JOHANNES KEPLER
UNIVERSITÄT LINZ

Author
Omid Mir

Submitted at
Institute of Networks and Security

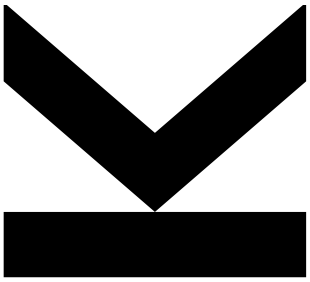
Supervisor and
First Evaluator
Univ.-Prof. Renè Mayrhofer

Second Evaluator
Univ.-Prof. Vanessa Teague

Co-Supervisor
Dr. Daniel Slamanig

August, 2023

Privacy Preserving Credentials via Novel Primitives



Doctoral Thesis

to obtain the academic degree of

Doktor der Technischen Wissenschaften

in the Doctoral Program

Technische Wissenschaften

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Abstract

Users or devices regularly need to demonstrate who they are on the Internet to enable decisions like whether they can access a certain resource such as a service. This often involves dedicated issuing authorities or identity providers (IdP), like Facebook or Google, who issue digital credentials for this purpose. Such credentials are important in securing access to traditional online services such as banking or email. However, they are becoming increasingly important in other non-digital areas such as travel (digital passports and driving licenses), physical door access (building or car keys), or digital health/vaccination credentials. However, in addition to creating single points of failure, relying on large and centralized identity providers raises concerns about the privacy of user data and the required trust in those central points. In particular, users lose control over their digital identity and disclose private data to authorities, which increases the severity of data breaches.

In this thesis, we investigate privacy-enhancing technologies to achieve the benefits of digital identity while preserving user privacy. By leveraging efficient cryptographic tools, e.g., signatures, zero-knowledge proofs, commitments, and encryption schemes, we propose secure protocols that safeguard user privacy while remaining practical. In particular, we focus on Anonymous Credentials (AC) as a basis for authentication and authorization, which have emerged as a promising solution for proving possession of credentials and attributes while preserving user privacy. Additionally, ACs can enable individuals to control their personal information and limit its collection and use by third parties. We develop and extend AC schemes regarding various properties while optimizing their efficiency as follows:

Issuer-Hiding Multi-Authority AC: We introduce the concept of Issuer-Hiding Multi-Authority Anonymous Credentials IhMA, which provides the Multi-Authority (**MA**) and Issuer Hiding (**IH**) critical concerns that have not yet been adequately addressed so far. **MA** means proving possession of attributes from multiple independent credential issuers more efficiently than showing multiple independent credentials. **IH** allows users to prove the validity of their credentials by only revealing that they have been issued by some the set of acceptable issuers but not the exact issuers. This protects the user's privacy, especially in decentralized settings where many issuers are involved, as verifying a user's credential may require knowledge of the Issuer's public key, which could inadvertently disclose private information about the user. Our proposed solution involves the development of two new primitives which are independent of interest:

- Aggregate Signatures with Randomizable Tags and Public Keys called AtoSa, where the aggregation and tag are useful for **MA**, and the latter feature (randomizable public keys) is essential for realizing the **IH** feature.
- Aggregate Tag based Mercurial Signatures called ATMS, which extend AtoSa to additionally support the randomization of messages and achieve equivalence class signatures (SPSEQ) and thus obtain a version of mercurial signatures that are aggregatable and have randomizable tags in other to provide the issuing hiding and unlinkability in multi-authority.

Delegatable AC: We present a novel delegatable anonymous credential (DAC) scheme that allows the owners of credentials to delegate the obtained credential to other users. It supports attributes, provides anonymity for delegations, allows the delegators to restrict further delegations, and comes with efficient construction. In particular, our DAC credentials do not grow with delegations, i.e., they are of constant size. Our approach builds on a new primitive:

- Structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC). The high-level idea is to use a special signature scheme to sign vectors of set commitments which can be extended by additional set commitments. Signatures additionally include a user's public key, which can be switched. Similar to conventional SPSEQ signatures, the signatures and messages can be publicly randomized and thus allow unlinkable showings in the DAC system.

Threshold Delegatable AC: We present a novel AC system with threshold issuance that additionally provides credential delegation and thus represents the first decentralized and delegatable AC. We provide a rigorous formal framework for such threshold delegatable anonymous credentials (TDAC). Our concrete approach departs from previous delegatable ACs and is inspired by the concept of predicate encryption and, in particular, functional credentials and builds upon the following primitive:

- A threshold delegatable subset predicate encryption (TDSPE) scheme, in which partial decryption keys are issued in a threshold way by multiple authorities and from which users then can generate decryption keys.

We also show how one can use any existing AC system (not necessarily from the above-developed AC systems) with login credentials (e.g., password and biometric) to provide privacy-preserving single sign-on.

Privacy-Preserving Single Sign-On: We construct a novel decentralized privacy-preserving single sign-on mechanism using a combination of existing AC systems and OPRF schemes with Multi-Factor Authentication, where the process of user authentication no longer depends on a single trusted third party (i.e., the IdP) in control of the whole authentication process. Also, it permits services where authenticating users remain anonymous within a group of users. Moreover, our scheme does not require the IdP to be online during the verification (passive verification).

Recovery of Encrypted Mobile Device Backups (eID): We propose a secure protocol for users to recover their electronic identity (eID) data in case of smartphone loss or malfunction. We leverage biometric authentication and auxiliary devices to allow clients to recover their secret keys from partially trusted servers using a Fuzzy Extractor.

We formalize all concepts and provide rigorous security definitions for all our proposed primitives and AC protocols. To validate the efficacy of our proposed solutions, we present efficient instantiations of the primitives/protocols. We also conduct performance benchmarking based on a prototype implementation made available as an open source python package to demonstrate the practical efficiency of our protocols and also primitives.

Acknowledgment

I would like to express my deepest appreciation and gratitude to the following individuals and organizations who have played significant roles in the completion of my Ph.D. thesis:

First and foremost, I am deeply grateful to my supervisor, René Mayrhofer, for believing in me, and for granting me the freedom to pursue my own research direction. I thank him for his unwavering support and invaluable guidance. I am happy to have had such a goodhearted and inspiring supervisor.

I'd also like to express my sincere thanks to my second supervisor, Daniel Slamanig. His inspiration and expert guidance have been pivotal in improving my research and igniting my interest in the world of cryptography. I am deeply thankful for the invaluable knowledge he shared, the myriad of things I learned from him, and for all our fruitful discussions.

I am grateful to my co-authors, Balthazar Bauer, Scott Griffy, Michael Hölzl, Anna Lysyanskaya, René Mayrhofer, Michael Roland, and Daniel Slamanig, for many interesting discussions and invaluable contributions, without whom my research would have been far less productive.

Enjoyable activities, socializing, and coffee-drinking breaks with colleagues have been integral to my Ph.D. experience. I am deeply grateful to my supportive colleagues, both past and present, from INS and LIT, for making this journey more enjoyable. The cherished memories we created together in these beautiful cultures will forever remain unforgettable.

To my dear friends and family, I feel blessed to have such a wonderful support system. My heartfelt gratitude goes to my parents for their unwavering love and support, which made all of this possible. I am deeply grateful to them beyond words. To my siblings, your love and belief in my abilities have been a constant source of inspiration. Thank you all for everything.

I dedicate this thesis to the memory of my father, whose role in my life remains immense, and to my mother for her constant encouragement and kindness.

Last, and above all, I'd like to thank you, Elham for being by my side and for all the efforts you have made to bear with my tendency to constantly talk about crypto stuff :)

Contents

List of Figures

List of Tables

1 Introduction

The digital world has brought tremendous benefits that have made our daily lives more comfortable than ever before. We can now perform a wide range of tasks online, from reading news on websites and socializing with friends on social media platforms to shopping, banking, and renewing our driver's licenses, among many others. However, for these activities to be secure, robust authentication is essential. It ensures that we buy products from genuine manufacturers, communicate with intended recipients, transfer money to the valid individuals, etc.

Authentication can be broadly categorized into two types: *(human-factors) identity based* and *attributes based authentication*. 1) The former involves traditional methods that require users to enter a combination of login credentials (using a password or biometric) or authenticate through a third party using a single sign-on (SSO) authentication protocol. The first option requires the user to maintain authentication credentials (e.g., password) for all services, while in the second case, the third party can track the user's activities. Meanwhile, attributes-based authentication uses a digital identity as a central concept, which can be seen as a collection of attributes (e.g., name, age, nationality, gender, etc.) representing a (real-world) entity in the digital realm. Instead of authenticating through a password (or biometric), one typically uses signatures as credentials, which certified the attributes. In a traditional sense, signatures allow a holder of a secret key to sign messages (attributes). Meanwhile, the resulting signature can be verified the related public key and all messages (attributes). These signatures are often issued by identity providers (IdP). On the Internet, a widely adopted practice is to have a centralized IdP, e.g., Google or Meta, to maintain the digital identity of users.

From a privacy perspective, however, the use of centralized identity providers poses significant challenges as users lose control over their digital identity (all their attributes reside at the IdP), and the IdP learns all the services a user consumes on the Internet (and data related to the use). Moreover, during the verification, users are often required to disclose more information than necessary, which can be problematic from a privacy standpoint. Such disclosures can lead to users' privacy being significantly compromised, and attackers may misuse or benefit from any personal data they can obtain.

Privacy-preserving technologies can overcome these privacy issues and offer private access control. More precisely, privacy-preserving technologies provide the protection of private data (that means not disclosing any privacy-sensitive information beyond what is required to be disclosed) while maintaining control over it. We also need to show that the certified data derives from a member of some permitted group while not displaying any other details about the actual user's identity. This is achieved through *anonymity* (hiding the

user's identity) and *unlinkability* (preventing anyone from linking transactions to a user or tracking their activities) in all interactions. Our focus is on both aspects of privacy, considering situations where the user's identity and attributes can reveal privacy-sensitive information. Guaranteeing strong privacy is a complex task that often requires leveraging cryptographic security guarantees. Our goal is to ensure the integrity of the data while simultaneously preserving the privacy of the users.

In this thesis, we investigate privacy-enhancing (attribute-based) credentials to achieve the benefits of digital identity while preserving user privacy. Our focus is on developing secure protocols that ensure user privacy while remaining practical. In particular, we focus on anonymous credentials as a basis for authentication and authorization, which have emerged as a promising solution for proving possession of credentials and attributes while preserving user privacy. Designing privacy-preserving schemes and ensuring their security is a delicate task that requires balancing strong security definitions, privacy protection, efficiency, and practical feasibility to develop usable privacy-preserving solutions. By doing so, we can ensure that users can enjoy the benefits of the digital world while protecting their privacy.

1.1 Background

We start by providing background on cryptographic approaches that form the basis of our work. Subsequently, we delve into the main concepts underlying the cryptographic schemes and AC protocols that we aim to present in this thesis.

1.1.1 Anonymous Credentials

An application of private data protecting techniques is an Anonymous Credential (AC) scheme. ACs play an essential role in privacy-preserving applications in which users can authenticate while disclosing minimal information [Cha85, CL01, CL03, CL04]. ACs consist of user(s), issuer(s), and verifier(s). An issuer issues credentials (*signature* on users' identity) to users, certifying specific attributes of the user. A user can then authenticate by proving possession of certain attributes as authorized in its credential.

Generally, to design AC systems in the digital world, a common approach is to combine signatures with zero-knowledge proofs, which enables proof of a signature without disclosing information about it and thus provides *anonymity*. Users can also access various services by selecting which attributes they want to disclose from the credential (*selective disclosure*). More precisely, one can use commitment and signature schemes combined with efficient protocols as follows: 1) a protocol in which we can prove knowledge of a committed message, 2) another protocol in which we can prove knowledge of the signature on a committed message, and optionally 3) a protocol for signing a committed message without revealing the committed message to the signing party (a blind signature). Moreover, these approaches usually provide the *multi-show* property such that several showings of the same credential cannot be linked. The key feature is that the user can choose which attributes

from a credential to reveal to a verifier, and the verifier can cryptographically verify the disclosed part. Meanwhile, the verifier should not learn anything more than that the disclosed attributes were certified by issuers. Also, multiple authentications are unlinkable, meaning that a verifier cannot distinguish a returning user from a new user. *The user has anonymity in the set of all users possessing the disclosed attributes (or even more powerful meaning prove predicated over attributes).*

In addition to the traditional AC schemes as described above, another popular type of AC system (also called sometimes self-blindable credentials) that is both simple and efficient is based on structure-preserving signatures on equivalence classes (SPSEQ) [FHS19, CL19]. SPS-EQ avoids the need for potentially costly zero-knowledge proofs and uses more efficient show protocols. This simplifies the construction process and eliminates the need for knowledge of the signature on committed (hidden) attributes, making it a straightforward approach to AC systems. However, they only support selective disclose attributes but not more expressive like proving predicates over attributes. In this thesis, we focus on this type of AC system.

1.1.1.1 Decentralizing Anonymous Credentials

All approaches discussed above rely on a central authority issuing credentials. Such a trusted party represents a single point of trust and failure and a valuable target of attack. Also, as shown in [GGM14a], compromise or issuer misbehavior can be quite difficult to notice in ACs. Moreover, in distributed settings spanning different contexts and jurisdictions, finding a dedicated party that is trusted by all participants in a system is usually non-trivial and often not desirable in practice. An increasingly prominent approach in conventional identity management is to take advantage of a distributed setting and in particular self-sovereign identity (SSI) frameworks like Sovrin[1] that use distributed ledger technologies (DLTs) and distributed public-key infrastructures (DPKIs) to mitigate this problems[2].

In SSI users are collecting certified attributes (called verifiable credentials) from *different sources* and then presenting (subsets of) verifiable credentials from this collection. There is an increasing push towards standardization of this verifiable credentials concept within W3C[3] and large efforts such as the future European data infrastructure (Gaia-X)[4] or the European Blockchain Services Infrastructure (EBSI)[5] are adopting this approach. Within the verifiable credential initiative in W3C it is also observed that privacy related features are important. In particular well-known features from AC systems such as supporting selective disclosure and proving predicates about attributes[6]. To realize this functionality

[1] https://sovrin.org/
[2] Sovrin also supports conventional anonymous credentials [KL17].
[3] https://www.w3.org/TR/vc-data-model/
[4] https://gaia-x.eu/
[5] https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home
[6] https://www.w3.org/TR/vc-data-model/#privacy-considerations

within W3C it is intended to base this upon the BBS+ signature scheme[7], a well-known building block for ACs currently being standardized by the CFRG of the IETF[8] (Privacy Pass [DGS+18] or the PrivateStats proposal by Facebook[9] to privately collect client-side telemetry from WhatsApp).

The aforementioned approach allows to preserve privacy in a setting where a user wants to show a single verifiable credentials issued by a single party. However, for a decentralized setting, where typically a subset of a collection of verifiable credentials from different issuers needs to be shown, the question of how to efficiently realize this arises. A naive way is to conduct a parallel credential showing of all the required verifiable credentials. However, apart from reduced efficiency this also has privacy implications. In particular, every verifiable credential reveals the exact issuer providing a lot of contextual partial information, e.g., a passport issued from a certain country or a driving license issues by a certain state reveal geographic information. This can be highly privacy intrusive in many settings and undermining the very objective of SSI systems [BFGP22]. And it is not possible to show that a credential comes from one of a larger set of issuers that would be accepted by a verifier. A set of independent recent works introduced a property called issuer-hiding [BEK+21, CLPK22, BFGP22] for AC systems. More specifically, users can prove that their credential is issued by one accepted issuer – without revealing which one. While this is a step towards countering the above privacy issues, these works only consider single issuers and are thus not yet suitable for a decentralized setting.

1.1.1.2 Delegatable Anonymous Credentials

In our everyday use of credentials, a common challenge arises: *delegation*. We often need to delegate our tasks, responsibilities, and permissions to others, or simply sharing access to resources and services with different individuals or electronic devices. Indeed, in practice, credentials are usually issued in a hierarchical manner, e.g., there is a chain of certificates between the user certificate and a trusted root authority. An easy way to achieve this goal is credential sharing. However, giving away the full and unlimited power of a credential is often not what we want. Consider the following example: The manager of a company wants to delegate a task that requires filling and signing documents to her secretary. Since the documents need a valid signature, the manager shares her digital signature key with the secretary. Unfortunately, this also gives the secretary the power to sign arbitrary documents in the name of the manager – which is obviously not what the manager intended. Even worse, since the actual signing key was shared, the only way to revoke these powers from the secretary is to revoke and invalidate the whole credential. As also highlighted in [BCC+09, CDD17, CL19], in a critical privacy context where AC systems are typically used traditional AC assumes that the verifying party knows the public key of the credential issuers. While such a hierarchical structure is desirable, this chain of issuers may reveal

[7] https://w3c-ccg.github.io/ldp-bbs2020/
[8] https://datatracker.ietf.org/wg/privacypass/about/
[9] https://research.fb.com/privatestats

sensitive information about the issuer's organizational structure or the credential holder.

Delegatable anonymous credentials (DACs), introduced by Chase and Lysyanskaya [CL06], solve this problem and support hiding the full delegation (issuance) chain and provide privacy during the delegation and the selective showing of attributes. This can be modeled via levels (cf. Belenkiy et al. [BCC+09]), e.g., any user can delegate a level $L = 1$ credential to another user. This level L credential can be used to derive a level $L + 1$ credential for another user. Also, the user can authenticate (i.e., prove possession of their credential) without disclosing their identity among the group of delegatees to a verifying party. Only the identity (i.e., public key) of the root issuer (not intermediate issuers) is revealed during the verification, while with traditional AC systems, the identity of all intermediate issuers will be revealed. Consequently, DACs provide stronger privacy guarantees than traditional AC systems.

1.1.2 Human-Factors Authentication

Authenticated Key Exchange (AKE) is one of the most broadly used cryptographic primitives which enables two parties to create a shared key over a public network. Typically, the parties need to have authentication tokens, e.g., cryptographic keys (asymmetric or symmetric high-entropy keys) or short secret values (low-entropy passwords). They also securely store these authentication tokens in a trusted service provider during the registration phase. There are various types of authentication factors such as knowledge, possession, and physical presence in practice low-entropy passwords are widely present. An example of an authentication protocol that relies on passwords is Password-based Authenticated Key Exchange (PAKE) [BPR00a].

However, passwords are usually vulnerable to both online and offline attacks [Cam17, WCW+17]. An adversary can compromises the data stored with the service provider (user account data, consisting of usernames and associated (potentially salted) password hashes) and run an offline dictionary attack on that data. Such an attack leads to the disclosure of user accounts and has happened several times in the past, cf. [Cam17, GP16, Gem15]. Even if low-entropy passwords are correctly salted and hashed, they still do not resist brute force attacks using modern hardware. Already in 2012, a rig of 25 GPUs could test up to 350 billion guesses per second in an offline dictionary attack [Pau12]. Multi-Factor Authentication (MFA) schemes overcome this risk by adding additional authentication factors. MFA combines (low-entropy) passwords with, e.g., secret values stored in physical tokens. Recent advancements in fingerprint readers and other sensors lead to the increased usage of smartphones and biometric factors in MFA schemes (e.g., the use of biometrics to securely retrieve private information [MMHN18]). These methods make it more difficult to guess the authentication factors. However, some MFA schemes incorporate password authentication and second-factor authentication as separate mechanisms and store a salted password hash (or biometric) on the server, leading to different vulnerabilities such as spoofing and offline attacks [OBM+18, JKSS18]. In other words, an adversary compromising the server is still able to recover the actual password (even if that password is no longer

usable without the additional associated factors). Moreover, mobile devices (smartphones, wearables, FIDO U2F , etc.) are considered more likely to be subject to loss or theft, and particularly smartphones and wearables open a large, high-risk attack surface for malware [MAR18, RPJ+18].

In general, authentication schemes are designed to uniquely identify a user. Consequently, they do not aim at protecting user privacy and users' activity in the digital world can easily be tracked and analyzed. Leakage of individual information may have serious consequences for users (including financial losses). To meet the increasing need of privacy protection in the digital world, multi-factor authentications are enhanced with privacy-preserving technologies. For instance, anonymous authentication schemes allow a member of a legitimate group, called a prover, to convince a verifier that it is a member of the group without revealing any information that would uniquely identify the prover within the group. Various schemes for anonymous password authentication have been proposed, e.g., [VYT05, Lin11, YZ08, SKI10]. In particular, anonymous password authentication guarantees unlinkability: The prover (e.g. the server of a service or identity provider) should not be able to link user authentications. Thus, for any two authentication sessions, the prover is unable to determine if they have been performed by the same user or two different users.

Centralized Authentication Architecture. An Identity Provider (IdP) with a centralized database of authentication data of all users could easily provide an MFA scheme and offer convenient single sign-on (SSO) to other services for its users [RR06]. SSO allows users to once receive a single token provided by the IdP and repeatedly authenticate themselves to service providers. Several initiatives such as PRIMA [ABS18], OAuth [HJ12], SAML [One19], and OpenID [RR06] let service providers take advantage of another centralized identity provider to authenticate users without becoming responsible for managing account passwords. In all these systems, the authentication follows a similar scheme (see Fig. 1.1) [AMMM18]:

- In the registration phase, the user creates credentials (e.g., a username/ID and a password) and passes them to the IdP (a trusted server) which stores the username together with the salted hash of the password.
- In the authentication phase, the IdP verifies the user-supplied sign-on credential by matching the username and password hash.
- After successful verification, the IdP issues an authentication credential (a digital signature or a message authentication code) using a secret key that authenticates the user to the service provider (e.g., a website) they want to visit.

However, this kind of centralized system poses several challenges:

- The IdP represents a single point of failure and an obvious target for attacks, such as:

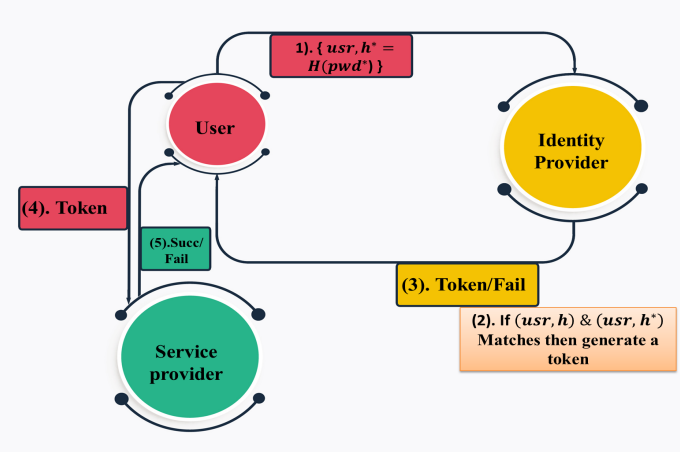


Figure 1.1: The generic flow diagram shows the authentication phase of a password-based token method. The figure does not include the registration phase where the user stores their username (usr) and hashed password (h) with the identity provider.

  - extraction of the secret key to forge tokens, which enable access to arbitrary services and data in the system;
  - capturing hashed passwords (or biometrics) to run offline dictionary attacks in order to recover user credentials;

  both potentially resulting in severe damage to the reliability of the system [AMMM18].

- The IdP is actively involved in each authentication session and can, therefore, track user activity, leading to serious privacy issues [AHS11, FKS17].
- The IdP takes a significant amount of control over the digital identity away from the user. Users cannot fully manage and store their identity by themselves but always need to rely on and interact with an always online IdP that offers the identity management system to them and the service provers they want to interact with.

Here we study decentralized options to overcome the above problems.

1.1.3 Recovery of Encrypted Mobile Device Backups (IDs)

Moving electronic identity (eID) into mobile devices is a growing trend. Examples include photo ID documents [HRM16], mobile payment wallets [GHR+15], and two-factor authentication tokens[10]. While this approach can increase both usability and security by transforming eID elements into hardware-backed software components on mobile devices, it

[10]E.g. by implementing the FIDO U2F or UAF protocols on smartphones with fingerprint sensors.

also creates a major single point of failure. Loss or theft of the smartphone becomes highly problematic when the user relies on it for identification, payment, and communication.

To address this issue, it is important for critical eID elements to be backed up, allowing owners to recover them on a new device if necessary - potentially under time pressure and outside of their normal, trusted environments. For example, assume Alice uses her smartphone as a digital identity and payment wallet, and her smartphone regularly creates a backup of all encrypted data, including payments and eID data. If her phone is stolen, she can acquire a new, compatible device and restore her private eID and payment data within a short time frame and under stress. Recovering her secret key to access her private eID data is also essential. While this backup system can help mitigate the risk of a single point of failure, other complex issues, such as locking/wiping/revoking the stolen phone or verifying the authenticity of a new device, are outside the scope of this thesis.

To date, the problem of backing up smartphones (not specifically eIDs) has been typically approached with implicitly trusted cloud services by the respective device or OS manufacturer. Although these services may potentially be made secure with significant technical effort (cf. the public presentation of the Apple cloud keystore [Krs16] or Android E2EE backup [11]), they still require complete trust in the operator. Although an organization may try to prevent itself from being able to extract previously stored key material with tamper-resistant hardware, the implementations and processes for new backups can always be changed without users being able to notice. Adding current issues of legal uncertainty in various countries concerning key escrow and encryption regulations, we argue that this level of trust in a for-profit company subject to a (potentially foreign) legal and political system is misplaced, especially with the implications of handling eIDs.

Ideally, we need to reduce the required trust in cloud services for backup and recovery of security- and privacy-critical data on smartphones, focusing on the use case of eIDs. The obvious (and naive) approach is to directly derive a cryptographic key from a user-provided password and locally encrypt/decrypt and sign/verify all backup data before sending it to the cloud service. In this case, the service provider would only need to be trusted for providing *availability*, but not for keeping *confidentiality* of the stored data (and *integrity* violations could at least be detected). Current approaches to full-device backup typically use such a method (including both the Android and iOS platforms at this time). However, the well-known difficulty of remembering passwords with high entropy [YBAG04] is even more of a problem for recovery of eID: such a recovery password would only be used very rarely (if at all) and often under stress. At the same time, it needs to be of higher entropy than typical login passwords, because it is the only element keeping a rogue (or legally compelled) service provider from violating the confidentiality of the backup data by simple brute-forcing a weak password. Therefore, a simple password-based key derivation function (PKDF) does not seem to be an appropriate solution, and on-device encryption methods have already been extended by including a hardware-based key part in the derivation function (first on iOS, now also on Android platforms). Unfortunately, for decrypting

[11] https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html

backup data on a new device during recovery, we cannot rely on a trusted execution environment (TEE) or other secure hardware to be in possession of a key part to contribute during key derivation, as we assume the original device (from which the backup had been created) to be completely unavailable (Alice's phone was stolen).

1.2 Related Works

We present related work on the topics discussed above.

1.2.1 Anonymous Credentials

Chaum proposed the concept of anonymous credentials (ACs) [Cha85] to address the *privacy problem*, which can be extended to provide privacy of service use even if credential issuers and verifiers collude [CL03, CL04, PS16a, ILV11, San20, FHS19, HS21]: by avoiding unique identifiers during authentication and providing only the relevant information from a credential, multiple different interactions of the same user cannot be linked. This *unlinkability* property must be considered a core privacy requirement for digital identity systems to avoid trivial mass surveillance. An important feature of modern anonymous credentials (often called attribute-based credentials or ABCs) is that they encode *attributes*, e.g., age or nationality, and allow users to selectively reveal some attributes while hiding others and thus supporting both fine-grained access control and data minimization[12] at the same time. Prominent AC systems are Idemix[13] by IBM and U-Prove[14] by Microsoft. A recent significant real world application is the integration of a more recent AC system [CMZ14a] into the popular Signal Messenger [CPZ20]. The most widely deployed technology related to ACs is the direct anonymous attestation (DAA) protocol [BCC04, CCD+17] supported by commonly integrated Trusted Platform Modules (TPMs). Nowadays, there are numerous approaches to construct ACs. The largest class of AC systems (and also DAA) builds upon re-randomizable signatures on commitments [CL03, CL04, PS16a, LMPY16] and related approaches such as equivalence class signatures [HS14, FHS19, HS21, CLK21] or redactable signatures [CDHK15, San20] (or more generic malleable signatures [CKLM14a]). Besides this signature-based approach, where a credential is essentially a signature from some issuer, a fundamentally different approach was recently proposed by Deuber et al. [DMM+18]. They construct an AC system based on predicate encryption [KSW08] and a user credential represents the decryption key for some predicate issued by the credential issuer (see Section 5.1).

[12] Unlinkability can only be guaranteed if neither the set of revealed attributes nor any associated meta data can be uniquely linked to a user.

[13] https://hyperledger-fabric.readthedocs.io/en/release-2.0/idemix.html

[14] https://www.microsoft.com/en-us/research/project/u-prove/

1.2.1.1 Decentralizing Anonymous Credentials

Garman et al. [GGM14a] introduced a solution in which there exists no credential issuers and thus no signatures. Users make claims about their identity attributes in form of commitments, which are submitted to a public transaction ledger, i.e., a blockchain. These registered commitments can then be used as a basis to compute Non-Interactive Zero-Knowledge Proof (NIZK) proofs, representing showings. While interesting, this cannot be considered a general solution as for many types of common credentials, e.g., passport, driving license, academic degrees, there is the need for explicit issuers.

Secondly, there is the concept of threshold issuance anonymous credentials, e.g., Coconut by Sonnino et al. [SAB+19] and threshold BBS+ by Doerner et al. [DKL+23]. Such a system thresholdizes a single issuer among a set of parties. While this helps to make AC systems more robust, it does not efficiently support multiple issuers and also does not support issuer-hiding.

Thirdly, Rosenberg et al. [RWGM22] present a framework to build ACs from existing identity documents, e.g., passports, driving license, whose contents (attributes) are registered in lists of "issuers". In particular, they are represented as commitments organized in Merkle trees and users obtain the authentication paths to their credentials. Then, users can use succinct NIZK proofs (zk-SNARKs) to prove statements about the attributes encoded in potentially multiple credentials. Here the zk-SNARK proofs for the single credentials are linked via NIZK proofs. This approach is very generic and avoids the lack of issuers in [GGM14a]. However, it is very complex and their credential showing due to computing Merkle membership proofs and linking of zk-SNARKS via NIZK can add significant costs (showing times in the order of seconds).

Recently, Hébant and Pointcheval introduced the concept of (traceable) Multi-Authority Anonymous Credentials (MA-ACs) [HP22]. Their approach to realize MA-ACs is based on so called aggregate signatures with randomizable tags and allows to aggregate showings of credentials of different issuers (but with respect to the same tag) into one compact showing. Due to randomizability of signatures and tags, it is possible to produce unlinkable showings. Moreover, the tag component has a secret part representing the user secret. While this is an interesting concept, it does not provide an efficient way of providing the issuer-hiding (IH) feature [BEK+21, CLPK22, BFGP22]. There is an obvious generic way to use a succinct NIZK, i.e., a zk-SNARK, and prove that the aggregated signature verifies for the given attributes under a subset of issuer keys without revealing which ones. While this can lead to an asymptotically compact solution, the prover will concretely be very expensive due the size of the verification keys, i.e,. they are of size $\mathbb{G}_2^{3+2n}$ each with n being the maximum number (types) of attributes, and thus the complexity of the verification equation in [HP22] to be proven with the zk-SNARK. Switching to non-succinct Schnorr-type NIZK obtained via Fiat-Shamir as done in [BEK+21] (in Construction 2), however, will result in a non-compact showing of size $O(n \cdot K)$ where K represents the number of issuers used in the aggregated output, and n represents the maximum number of attributes (even when ignoring the size of the proof corresponding to the non-shown

attributes). We consider this an important concept and aim to propose the first concretely efficient issuer-hiding MA-AC system.

1.2.1.2 Delegatable Anonymous Credentials

Chase and Lysyanskaya in [CL06] introduced the notion of delegatable anonymous credentials (DAC). DAC schemes, later improved in [BCC+09], are particularly interesting for applications such as (physical) access control [MSM+18], root of trust[15] [CL19], or authorizing transactions in permissioned blockchains [CDD17, BCET21a]. Belenkiy et al. [BCC+09] define their construction through levels. In their scheme, any user can issue a level $L = 1$ credential to another user. This level L credential can be used to derive a level $L + 1$ credential for another user. Also, the user can authenticate (i.e., prove possession of their credential) without disclosing their identity among the group of delegatees or any of the attributes of the credential to a verifying party. Only the identity (i.e., public key) of the root issuer is revealed during the verification. The foundation of [BCC+09] is based on commitment and signature schemes that incorporate randomizable NIZK proofs. Furthermore, they demonstrate the applicability of their method using Groth-Sahai (GS) commitments and GS NIZK proofs [GS08], illustrating that these can be instantiated with a size that scales linearly in relation to the chain length L. However, using such heavy tools like the GS proofs makes their scheme inefficient for practical use as the quite expensive statements result in poor performance and large credential size. Several other DAC constructions have been proposed afterwards, e.g. [CKLM13, Fuc11, CKLM14b], which follow roughly the same techniques as [BCC+09], i.e., using malleable proof systems (based on GS) as the main building block, and thus have similar performance characteristics.

Camenisch et al. [CDD17] propose a DAC scheme that is efficient and practical. In this scheme, we can prove possession of a credential in a privacy-preserving way, but we cannot obtain credentials anonymously. Indeed, credential holders can see all attributes and public keys on all levels in plain, i.e., not offering an anonymous delegation phase. They present an efficient instantiation of their DAC scheme based on the structure-preserving signature (SPS) by Groth [Gro15]. Later Blömer and Bobolz [BB18] proposed another practical DAC construction using dynamically malleable signatures (DMS) and NIZK proofs, which conceptually is similar to the approach in [CKLM14b]. In DMS a set of allowed transformations is not static but can be modified for each signature. Thus, one can derive signatures that are more restricted. Unfortunately, [BB18] does not describe a full instantiation of their generic protocol, but according to [BCET21a], it appears less efficient than [CDD17].

Crites and Lysyanskaya [CL19] provide probably the most efficient and conceptually most simple construction of delegatable anonymous credentials. Their approach doesn't rely on complex tools like NIZK proofs. At the same time, they assert enhanced security features compared to those of [CDD17], accomplished through the inclusion of an anonymous

[15] https://www.gradient.tech

delegation phase. The main building block of their construction is a new type of signature scheme, called mercurial signature. A mercurial signature extends structure-preserving signatures on equivalence classes (SPSEQ) [FHS19] to equivalence classes on the key space. SPSEQ in addition to randomizing signatures also provides randomization of signed messages (modeled as equivalence classes). Thus, SPSEQ allow similar applications as SPSs, but unlike the latter they do not need NIZK proofs on top, thereby yielding more efficient schemes. Mercurial signatures extend SPSEQ in the sense that they add the property of transforming public keys into an equivalent one, i.e., additionally supports randomization of public keys.

Unfortunately, the DAC scheme derived from mercurial signatures in [CL19] has some drawbacks: 1) It does not support attributes when used in a DAC which are often the key values that we use to prove things (such as "Programmer and PhD"). This makes their scheme unsuitable for many applications. Supporting attributes is left as an open problem in [CL19]. 2) [BF20] demonstrate a drawback of their weak form of anonymity when applying their mercurial signature in the DAC context. In essence, when Alice delegates a credential to Bob, it allows her to identify Bob each time he uses the credential for authentication. This situation presents a significant violation of Bob's privacy. It arises from the fact that, during the delegation process, Alice uses her secret key to sign Bob's pseudonym using her own pseudonym, which is represented by the randomized public key. This signed information becomes an integral component of Bob's credentials (for further elaboration, refer to [BF20]). 3) Similar to [CDD17], the credential size depends on the delegation chain length L, and thus, the size of signature grows linearly with L.

Summarizing the state-of-affairs in existing DAC schemes, efficiency in DAC schemes still remains as a major challenge. In fact, the ones that are conceptually simple and practically efficient do not provide all the desirable properties of supporting arbitrary attributes, being compact, and providing strong anonymity guarantees at the same time. Additionally, the previously proposed schemes do not support restricting capabilities during the delegation between users. Applying such a restriction during the delegation phase would allow restricting the purpose of a credential (cf. [MSM+18]), e.g., for performing a specific task during a specific time frame, or a user can only delegate a certain subset of their capabilities to another user.

1.2.2 Human-Factors Authentication

1.2.2.1 Single-Factor (Password) Authentication Key Exchange

For a long time, knowledge was (and still is) used as a primary means of authentication. Single-factor authentication based on passwords and PINs is a mechanism that is well-studied. Bellovin and Merritt [BM92] proposed Encrypted Key Exchange (EKE) where a client and a server share a password and use it to exchange encrypted information to agree on a common session key. EKE was followed by several enhancements (cf. [BM93, BMP00a, GL06]). Bellare et al. [BPR00a] introduced a comprehensive and formal provable model

for Password Authentication Key Exchange (PAKE). Building upon this work, Gennaro and Lindell [GL03] and Groce and Katz [GK10] proposed two generic PAKE schemes. These schemes are considered among the most efficient methods for constructing PAKE protocols in the standard model, which avoids relying on additional idealizing assumptions and ensures the strongest security guarantees.

Benhamouda and Pointcheval [BP13] proposed an extension to the traditional authenticated key exchange, incorporating a verifier into the process. The verifier is represented as a hash value or transformation $V = H(s, pw)$, where pw is the secret password and s is a public salt. Each user's entry in the server's database consists of the pair (s, V).

1.2.2.2 Multi-Factor Authentication

A single knowledge-based authentication factor has the disadvantage that an adversary needs to only compromise that single factor. Multi-factor authentication (MFA) overcomes this by combining multiple different factors. The widely-used combination is long-term passwords with secret keys, possibly stored in tokens (e.g., FIDO U2F). Shirvanian et al. [SJSN14] introduce a framework to analyze such two-factor authentication protocols. In their framework, the participants are a user, a client (e.g., a web browser), a server, and a device (e.g., a smartphone). In the authentication phase, the user sends a password and some additional information provided by the device. In most existing solutions, including [SJSN14, BCL16], during the registration process, the user gets a value called the "token", while the server records a hashed password. During the authentication phase, the two required factors (the password and the token) are sent to a verifier.

Jarecki et al. [JKSS16] proposed a password-authenticated key exchange protocol with device enhancement, utilizing mobile device storage as a token. The scheme achieves two primary objectives: Firstly, to thwart offline dictionary attacks, an adversary would need to compromise both the login server and the mobile device storage. Secondly, the user is required to confirm access to the mobile device storage during the login process, adding an extra layer of security.

Another popular factor used to authenticate users to remote servers is biometrics [HW15, HXB+14, PZ08]. Fleischhacker et al. [FMA14] also propose a modular framework called MFAKE which models biometrics following the liveness assumption of Pointcheval and Zimmer [PZ08]. However, Zhang et al. [ZXSM17] demonstrate that their scheme does not adequately protect privacy. Indeed, biometric authentication becomes a weak point when the framework directly uses the biometric template for authentication. In addition, it requires to execute a lot of sub-protocols which makes the scheme inefficient.

1.2.2.3 Anonymous Authentication

To better understand the potential dangers of online data collection and anonymous authentication, let us consider a loyalty programs example [BJDF16]. Imagine a grocery store that offers a loyalty card to customers. This loyalty card allows customers to obtain

discounts on specific items, but it also tracks their purchases and builds a profile of their shopping habits. If the customer only uses the card occasionally and does not mind the store having access to their purchasing history, they may not be bothered by this data collection. However, if the store were to sell this data to a third party or use it to discriminate against customers, the customer's right to privacy would be compromised. Blanco et al. [BJDF16] and Bobolz et al. [BEK+20] propose a privacy-aware loyalty program based on blind signatures and AC schemes and generalization of products that allows vendors and consumers to enjoy the benefits of loyalty while allowing consumers to stay anonymous.

Another approach towards user authentication is the anonymous password authentication protocol proposed by Viet et al. [VYT05]. They combine an oblivious transfer protocol and a password-authenticated key exchange scheme. Further enhancements were proposed by [YZ08, SKI10]. An anonymous authentication protocol permits users to authenticate themselves without disclosing their identity and becomes an important method for constructing privacy-preserving authenticated public channels. Zhang et al. [ZXSM17] presented a new anonymous authentication protocol that relies on a fuzzy extractor. They consider a practical application and suggest several authentication factors such as passwords, biometrics (e.g., fingerprint), and hardware with reasonably secure storage (e.g., smartphone).

1.2.3 Recovery of Encrypted Mobile Device Backups (IDs)

An approach to secretly backing up the credentials of smartphones has been presented by Ivan Krstić, Head of Security Engineering and Architecture at Apple [Krs16]. In their concept of a cloud key-store, they encrypt the credentials with a random backup ("escrow") key and further protect it with a user-defined *iCloud Security Code* (iCSC). This escrow key is stored inside a tamper-resistant device, a so-called Hardware Security Module (HSM), on the Apple server infrastructure. As this key never leaves this tamper-resistant hardware, decryption of the smartphone credentials can only be done within this HSM and by providing the correct iCSC. To further protect the credentials and the escrow key from being disclosed, Apple destroys the access keys for the administration of these HSMs (i.e. keys to program the HSM), locking even themselves out. However, this system still requires some trust in the operator. Even though the organization tries to prevent itself from being able to extract previously stored key material with tamper-resistant hardware and delete the access keys to that hardware, the implementations and processes for new backups can always be changed without users being able to notice. Recently, secret sharing protocols with password protection have been introduced as a way to solve this problem and remove the tamper-resistant hardware. The first Password Protected Secret Sharing (PPSS) scheme was proposed by Bagherzandi et al. [BJSL11]. Their scheme allows a user to distribute a secret key among different servers, and then reconstruct it from a single password, by communicating with at least $t + 1$ honest servers (among n possible ones). Also the public information (to enable the reconstruction key) is stored on each of the servers. The scheme has an initialization phase where the user communicates with each

of a set of n servers $S_1, ..., S_n$. After that each server S_i stores some public information associated with the user, the public information is a function of the secret key sk, the password pw and the server names S_i. When a user needs to retrieve the secret key sk, she runs a reconstruction protocol by interacting with a subset of at least $t + 1$ servers where the only input from the user is her password pw. However, the authors assumed an additional PKI. Furthermore, if an adversary can catch the key pair of one server, he has the ability to run an offline attack [ACNP16]. After that, Camenisch et al. [CLLN14] introduce a PPSS protocol for Threshold Password-Authenticated Secret Sharing (T-PASS), that does not require PKI authentication during the reconstruction phase. However, their scheme is still expensive. The cost of their scheme is 14 client exponentiations per server and 7 exponentiations for each server. It also requires 10 messages between a user and each server in the secret reconstruction phase. Yi et al. [YHCL15] propose a lightweight TPASS based on distributing a password, a secret and a digest of the secret. However, in the reconstruction protocol, at least t servers perform a broadcasting protocol to obtain and return the ElGamal encryptions of both the secret and the digest. Then users can verify the secret key. Camenisch et al. [CLN15] propose an efficient protocol that is not based on robust secret sharing scheme or zero-knowledge. Nevertheless, it is not able to detect which shares are valid. Since that if a password is incorrect, the user's failure will happen at the end of the verification step and they need to restart again with a different set of servers, which leads to DoS attack. Jarecki et al. [JKK14] present a PPSS scheme that uses a Verifiable Oblivious Pseudorandom Function (VOPRF) to avoid simple DoS attacks. Indeed, it guarantees that the user detects which server has tried to cheat or which communication has been changed. Jarecki et al., in [JKKX16] further improve the cost of this password-only PPSS by giving up of the robustness property. but, this can be a good method with a few servers. Also, the user is unable to detect the cheating servers. Abdalla et al. [ACNP16] propose two efficient Oblivious Pseudorandom Random Function (OPRF) constructions to overcome this drawback: The first one is based on the One-More Gap Diffie-Hellman assumption. The second scheme is on oblivious evaluation of the Naor Reingold PRF, based on the sole DDH assumption. Their main contribution is the efficient realization of the robustness in only one round of communication with each server. They also avoid any complex zero-knowledge proof. Although their scheme is much more efficient than the other schemes, it is still expensive (because it requires communication with many servers). Moreover, if the user enters a wrong password, the protocol confront with failure which leads to extra computations.

1.3 Contribution

Our main goal is to present practical/efficient protocols and schemes that are smoothly deployable in various privacy applications while focusing on cases where authentication is a crucial property. This thesis contains the following contributions, which split into several chapters. More precisely, in chapters 3, 4, and 5, we develop AC schemes in a so-called

self-blindable paradigm (i.e., the credential can be randomized between showings, effectively making the credential function as a zero-knowledge proof and no need for knowledge of the signature on committed attributes) regarding various properties while optimizing their efficiency. Chapter 6 shows how to combine AC with human factor authentication to provide a privacy-preserving single-sign authentication. In Chapter 7, we show how to securely restore the backup of this digital identity information. Finally, In Chapter 8, we provide a prototype implementation of our ACs in python. In more detail, the contributions are as follows:

1.3.1 Chapter 3: Issuer-Hiding Multi-Authority Credentials

Our goal here is to formalize and present a construction of (Issuer-Hiding) Multi-Authority Credentials that mitigate the aforementioned problems. This chapter is based on the paper [MBG+23] which has been accepted at ACM CCS 2023. Our contribution in this paper is twofold:

Aggregate signatures with randomization features. To achieve our goals, we present a fundamental approach by introducing tag-based aggregate signatures featuring randomizable tags and public keys. We then enhance these signatures to support message randomization, akin to the functionality of equivalence class signatures (SPSEQ) [FHS19]. Rigorous formal security models are provided for both types of schemes, along with provably secure instantiations within these models. In detail, our contributions include:

Aggregate signatures with randomizable keys and tags (AtoSa[16] for short). We introduce a novel scheme called *Aggregate Randomizable Tag-based Signatures* (AtoSa), where signatures are associated with tags that consist of private and public parts, allowing for aggregation of signatures sharing the *same* tag. Moreover, we enable the randomization of verification keys and tags, which are defined with respect to equivalence classes. This can be seen as an extension of aggregate signatures with randomizable tags, as previously introduced in [HP22], with the additional feature of randomizable keys and appropriate signature adaptation. In our scheme, existing signatures can be adapted to verify under the randomized public keys and tags. We build AtoSa based on the well-known Pointcheval-Sanders (PS) signatures [PS16a], which have been widely used as the foundation for various privacy-preserving primitives, such as group signatures and anonymous credentials [PS16a], redactable signatures [San20, San21], and dynamically malleable signatures [BB18].

*Aggregate Mercurial Signatures with Randomizable Tags (*ATMS*).* We introduce an advanced extension known as *Aggregate Mercurial Signatures with Randomizable Tags* (ATMS), which significantly enhances the capabilities of AtoSa by supporting the randomization of messages, enabling the use of equivalence classes of messages similar to (SPSEQ). In addition to AtoSa, ATMS allows existing signatures to be adapted for verification under

[16] The (ancient) Greek transliteration of the old Persian name U*tauθa*. Atossa means "bestowing very richly" or "well trickling" or "well granting". It refers to an Achaemenid empress who was the daughter of Cyrus the Great, and the wife of Darius the Great.

randomized messages, meaning that different representatives of the same message class can be used for verification. As a result, we achieve a version of mercurial signatures [CL19] that combines both aggregatability and randomizable tags. This represents a pioneering instance of an aggregate structure-preserving signature (and, consequently, SPSEQ). We present an ATMS construction inspired by the message-indexed SPS in [CKP+22], which itself is a variant of Ghadafi's SPS [Gha16] scheme.

Restrictions of our Constructions. It is important to note that our constructions differ from standard aggregate signatures in two aspects: 1) Firstly, they require all aggregated messages and corresponding verification keys to be known before requesting the first signature. 2) Alternatively, our constructions can be adapted to make the same assumption as synchronized aggregate signatures [AGH10, HW18]. In this adapted setting, every issuer ensures that only a single signature is issued for each tag. We will present our results based on the first approach, but we will also discuss adaptations for the second approach. It is worth mentioning that these adaptations do not alter any of the interfaces, security definitions, or proofs. As our main application involves anonymous credentials, the choice between the first and second approaches depends on the specific application scenario. However, it remains an intriguing open question to achieve fully dynamic signatures without relying on any of the above assumptions.

Like other types of signatures with randomization features, we also expect that our schemes will find applications beyond the one presented here.

Issuer-Hiding Multi-Authority Anonymous Credentials. We introduce a formal model for issuer-hiding multi-authority anonymous credentials (IhMA) and present two efficient constructions based on AtoSa and ATMS, denoted as $\mathsf{IhMA}_{\mathsf{AtoSa}}$ and $\mathsf{IhMA}_{\mathsf{ATMS}}$, respectively. These constructions address the challenges of user privacy and scalability in multi-authority settings, making them significant contributions to the field of ACs. In our $\mathsf{IhMA}_{\mathsf{AtoSa}}$ and $\mathsf{IhMA}_{\mathsf{ATMS}}$ constructions, acquiring a credential involves obtaining signatures on desired attributes from a group of issuers, all under the same tag (which can be considered the user's identity in credential schemes). During the showing phase, signatures are randomized from the relevant issuers, along with the tags, and then aggregated. Finally, the user presents the aggregated signature, and optionally, opens subsets of attributes or proves predicates over them, while also providing a proof of knowledge of the secret tag part. Both $\mathsf{IhMA}_{\mathsf{AtoSa}}$ and $\mathsf{IhMA}_{\mathsf{ATMS}}$ are highly efficient, but they do involve some trade-offs, which we thoroughly discuss below.

Enabling the issuer-hiding feature [CLPK22] operates in the following manner: Each verifier creates a "key-policy," which specifies a group of issuers (identified by their verification keys) from whom the verifier will accept an (aggregated) credential. This policy is a collection of SPSEQ signatures on the verification keys of either the AtoSa or ATMS scheme. As the equivalence classes of the SPSEQ (representing the message space) align with the key equivalence class of AtoSa and ATMS, the process of showing a credential remains similar to what was described before. However, in this case, all verification keys of the AtoSa or ATMS are randomized, and the corresponding SPSEQ signatures in the

key-policy are adjusted accordingly.

In the $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme, instead of directly signing attributes, we adopt the framework introduced by Fuchsbauer et al. [FHS19]. In this scheme, the signature is used to sign set commitments to attribute sets. However, incorporating this approach is not straightforward, as it requires ensuring that set commitments are compatible with the message space of our ATMS. As an additional contribution, we introduce a generalization of the decisional uber assumption family by Boyen [Boy08], along with an interactive version, to prove the anonymity of this construction. While both $\mathsf{IhMA}_{\mathsf{AtoSa}}$ and $\mathsf{IhMA}_{\mathsf{ATMS}}$ share a common objective, the differences in their constructions give rise to certain trade-offs in terms of functionality and efficiency:

- Credential size: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme can yield a fixed-sized credential, while the $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme does not achieve this without utilizing Zero Knowledge Proof of Knowledge (ZKPOK) of signatures.
- Efficiency: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme is more efficient at showing and verifying credentials compared to the $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme.
- Need for a trusted party: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme requires a trusted party, while the $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme does not. This is because $\mathsf{IhMA}_{\mathsf{ATMS}}$ relies on a trusted party to hold a trapdoor to generate set commitments, whereas $\mathsf{IhMA}_{\mathsf{AtoSa}}$ does not require such a trusted party.
- Expressiveness: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ supports revealing a subset of attributes from a set of attributes per issuer, i.e., selective disclosure per issuer. The $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme only supports a single attribute for each credential. Consequently, it only supports selective disclosure over all issuers. However, both schemes allow for proving arbitrary predicates over signed messages.

Overall, the choice of the concrete construction depends on the specifics of the use case or application and priorities set in the overall system.

1.3.2 Chapter 4: Efficient Delegatable Anonymous Credentials

Our goal in this work is to formalize and present a construction and implementation of DAC that mitigate the aforementioned problems. Along the way, we therefore design a novel cryptographic building block that we call structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC) and might be of independent interest. This chapter is based on the paper [MSBM23]. We summarize our main contribution below:

Delegatable Anonymous Credentials. We propose a novel delegatable anonymous credentials scheme (DAC). Our scheme provides the following key characteristics: *i*) It represents a *simple and practical* construction without requiring zero-knowledge proofs (for complex statements), which makes it well-suited for real-world applications. *ii*) It

is *constant-size* in two aspects. First, The bandwidth needed for the credential showing protocol remains unaffected by the number of attributes involved, but only depends on the delegation depth. Second, unlike the schemes in [CL19, CDD17] and similar to [BB18] the credential size is independent of the length of the credential (delegation) chain. *iii*) Credentials are *attributes-based* in a sense that every level in the delegation chain is associated to a set of attributes that are certified by the respective delegator. A credential holder can then decide for every level whether and which attributes should be selectively revealed during a showing of a credential. Moreover, every delegator can *restrict delegation* in how many further levels can be delegated and whether attributes associated to previous levels should be valid (showable) or invalidate them (making them unshowable). *iv*) It provides *full privacy* which means not only support an anonymous showing phase but also provide an anonymous delegation phase. Finally, *v*) our DAC comes with a prototypical *implementation* and evaluation that demonstrates its practical efficiency.

Novel Building Block. Our DAC scheme is based on a novel cryptographic building block that we call structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC). This primitive draws inspiration from structure-preserving signatures on equivalence classes [FHS19] as well as the set commitment scheme used in the aforementioned work to construct conventional anonymous credentials. Loosely speaking their idea is to use SPSEQ to sign a randomizable set commitment and showing a credential amounts to randomizing the message (set commitment), randomizing and adapting the signature to the new message and providing the signature, randomized commitment, and an opening to the randomized commitment. While in SPSEQ the message space is simply group element vectors, in SPSEQ-UC the message space is viewed as a vector (of length at most ℓ) of randomizable set commitments. This concept is somewhat similar to signatures on randomizable ciphertexts (SoRC) [BFPV11, BF20]. However, in contrast to SoRC which does not allow to reveal a subset of the encrypted message, here it is also possible to reveal only a subset of the committed values of each commitment in the vector while guaranteeing the privacy of the non-revealed ones. Thereby, SPSEQ-UC needs to be unlinkable, which means the same commitment-signature pair can be revealed multiple times without being linkable to each other. One key feature is that SPSEQ-UC allow to extend signed vectors by additional set commitments. More precisely, in SPSEQ-UC signing of a commitment vector of length k also produces an update key $\mathsf{uk}_{k'}$ corresponding to an integer k' with $k \leq k' \leq \ell$. Given the update key $\mathsf{uk}_{k'}$ one can update a commitment vector $\mathbf{C}$ to a vector $\mathbf{C}'$ (i.e., extending it). Another key feature is that in a SPSEQ-UC scheme the signing process is tied to a user public key. It allows a signer to produce a signature under her secret key for a given user public key such that this signature can be adapted into another valid signature for a new user public key by anyone knowing the related old user secret key.

We provide a rigorous security model for SPSEQ-UC which carefully crafts privacy notions similar to SPSEQ [FHS19] in order to guarantee that adapted (i.e., re-randomized) signatures, signatures after extending commitment vectors as well as signatures after

switching user public-keys are distributed identically to new signatures and thus are all unlinkable to fresh signatures. This is important for our application in DAC and other potential application in privacy-preserving protocols. Moreover, we provide a provably secure construction of an SPSEQ-UC. It is based on the SPSEQ scheme in [FHS15] and the set commitment scheme in [FHS19], but requires significantly new ideas to provide all the desired functionality.

1.3.3 Chapter 5: Threshold Delegatable Anonymous Credentials

Our overall goal is to introduce, formalize, and present the first construction and implementation of threshold delegatable anonymous credentials (TDACs) to mitigate the problems with existing constructions. This chapter is based on the paper [MSM23].

Building block. We start from a subset predicate encryption (SPE) scheme [KMMS17]. In such scheme, a user with a secret key for a set s can decrypt a ciphertext to obtain plaintext message M (which in DACs can, e.g., be challenges set by a verifier) if the ciphertext has been produced with respect to a set s' if and only if s defines a subset of s', i.e., $s \subseteq s'$. Moreover, we need to recall hierarchical predicate encryption (HPE) [OT09], which allows the delegation of secret keys to lower levels in a hierarchy. Building upon these concepts, we introduce the notion of *threshold delegatable subset predicate encryption* (TDSPE), whose features can be summarized as follows: 1) it extends SPE to support *delegation* and 2) it supports a *threshold issuance of decryption keys*. The latter means that we divide the trust by having multiple authorities. In order to generate a decryption key, a user must receive a threshold number of partial decryption keys by interacting with an authorized subset of authorities. We construct this process in a *non-interactive* way from the authorities' perspective, which means authorities do not need to interact with each other to create decryption keys during the key issuing steps. This is akin to multi-authority approaches in attribute-based encryption [Cha07, LW11]. However, in contrast to existing multi-authority approaches, our threshold mechanism allows some authorities to not be available and a user no longer needs to get the partial secret key from *all* authorities. Moreover, as long as the number of the corrupted authorities is not more than the threshold t, the system still works despite the fact that there are corrupted authorities. As a result, the system is able to provide secret keys for new users unless more than the *threshold* of authorities in the system are malfunctioning, which is especially useful in distributed applications.

Ultimately, we provide a *simple and efficient construction* of a TDSPE scheme based on an SPE scheme in [KMMS17]. In addition, our construction supports an *unbounded attribute* universe and *constant size public parameters* and can be proven secure under a well known assumption in the random oracle model in prime order bilinear groups.

Threshold delegatable anonymous credentials. We introduce the concept of threshold delegatable anonymous credentials (TDAC), thereby combining the advantages of

anonymous credentials with threshold issuance features and the delegation capability of delegatable anonymous credentials. In decentralized settings, TDAC can cope with a threshold number of dishonest or faulty nodes (Byzantine fault tolerance), which provides robustness and availability properties while at the same time supporting multi-level controlled delegation. In particular, we propose a formal model of TDAC capturing desirable security properties (i.e., unforgeability, unlinkability, and anonymity). As our main contribution, we propose an instantiation of TDAC that we show to be provably secure in our model. It is based on our TDSPE and yields an efficient construction with practical efficiency. Our TDAC construction is the first scheme that supports threshold issuing and delegatable credentials with credentials that are short and aggregatable in a multi-authority setting.

We present performance benchmarks based on a prototype implementation and provide a concrete efficiency comparison with the most relevant works Coconut [SAB+19] and Functional Credential (FC) [DMM+18], including the number of communication rounds, communication complexity, and run time. In this comparison, we show the attribute and parameters domain in the related schemes. In particular, in our approach we can encode the attributes and support an unbounded attribute universe while at the same time having compact (fixed size) public parameters independent of the number of supported attributes. This is a feature that Coconut and FC do not provide. Moreover, our TDAC scheme is non-interactive such that there is no need for interaction between issuers in the issuing of credentials.

We stress that TDAC is designed for a controlled delegation model. That is, root issuers define a fixed set of attributes that can be used and delegated (cf. Section 5.3.1 for encoding of attributes). While this can be a limitation for some applications, this level of control is beneficial for some relevant applications (see below and Section 5.5.3 for more comparisons between TDAC and the related schemes).

1.3.4 Chapter 6: Privacy-Preserving Single Sign-On

This chapter is based on the publications [MRM20, MRM22].

In this chapter, we construct a novel decentralized privacy-preserving single sign-on scheme using a new Decentralized Anonymous Multi-Factor Authentication scheme (DAMFA), where the process of user authentication no longer depends on a single trusted third party. Instead, it is fully decentralized onto a shared ledger to preserve user privacy while maintaining the single sign-on property. That is, users do not need to register their credentials with each service provider individually. The scheme also permits services where authenticating users remain anonymous within a group of users. Subsequently, our scheme does not require the IdP to be online during the verification (*passive verification*). Moreover, since there is no single third party (i.e., the IdP) in control of the whole authentication process, user and usage tracking by the IdP is inhibited.

The passive verification property of our scheme allows service providers to authenticate users at any time without requiring additional interaction with an IdP except what is available on the shared ledger. This property removes the cost of running secure channels

between the service provider and the identity provider. Simultaneously, the IdP is eliminated as a single point of failure and attack within the authentication process.

The scheme relies on personal identity agents as auxiliary devices that assist the user in the authentication process. The personal identity agents participate in a threshold secret sharing scheme to store the distributed private key of their users. in the authentication phase, the user unlocks their private key through a combination of biometrics and a password, combining biometric, knowledge, and possession factors. The distributed architecture prevents offline attacks against data extracted from compromised agents, as long as only a set of agents below the threshold is compromised or corrupted.

We define the ideal functionality and real-world definitions for the security of our DAMFA scheme. We prove our construction's security via ideal-real simulation. Finally, we demonstrate that our protocol is efficient and practical through a prototypical implementation and through a comparison of our scheme with other SSO works.

1.3.5 Chapter 7: Recovery of Encrypted Mobile Device Backups (IDs)

This chapter is based on the paper [MMHN18].

As mentioned early, the growing trend of moving electronic identity (eID) components into mobile devices, such as passports, credit card wallets, and loyalty cards, has created a major single point of failure for individuals. A smartphone loss, theft, or malfunction becomes problematic when users rely on it for identification, payment, and communication. To address this issue, we propose a solution for the backup and recovery of security- and privacy-critical data on smartphones, particularly for eID components.

The approach presented in this chapter offers a novel and practical solution to the challenge of securely backing up and recovering sensitive data on smartphones. It not only overcomes the difficulty of remembering passwords with high entropy but also reduces the necessary level of trust in cloud services. We leverage biometric authentication and auxiliary devices to allow clients to recover their secret keys from partially trusted servers using a fuzzy extractor to increase the entropy of the cryptographic key. This solution can potentially improve both the usability and security of eID components on mobile devices and could be extended to other security- and privacy-critical data on smartphones. Overall, our approach therefore relies on the following two aspects to enable authenticated recovery from partially trusted cloud services:

The owner authenticates biometrically, and these biometric identifiers are part of the key derivation function based on a fuzzy extractor. Individuals therefore do not have to remember strong passwords. However, we do not assume biometric identifiers (specifically fingerprint data within the scope of this paper) to be confidential against sufficiently dedicated adversaries.

To increase the entropy of the resulting cryptographic key, an additional key part is added in the key derivation, akin to the device-specific, hardware-based keys currently used for on-device encryption. As we cannot rely on a single secure hardware component

to be available, we split this key into shares that need to be combined during recovery. For instance, one can keep one of these shares online in a cloud service, carry a second one printed as a QRcode with her during traveling or on an auxiliary device.

1.3.6 Chapter 8: Practical Realization (Implementation)

As part of this thesis, we have developed a Python package that implements our new anonymous credentials, primitives, and protocols. The package includes all the necessary building blocks to construct these new protocols and showcase their practical applications and proof of concept.

To provide a standardized and cohesive interface for these credentials, we designed an interface called AC. Each new AC (chapter) of the thesis implements this interface and adds new features to it. This approach enables easy integration and comparison of the various anonymous credentials schemes developed throughout the thesis. The resulting Python package showcases the practicality of the proposed ACs and provides a useful tool for others to build upon and further advance this area of research.

This package can serve as a tangible demonstration of the author's contributions to the field and provides a valuable tool for further research in this area. Moreover, including the package in the thesis ensures consistency and clarity in presenting the new anonymous credentials and allows future researchers to build upon and advance this work.

1.3.7 Publication History

The majority of the material in this thesis has either already been published or accepted and will soon be published in the following papers, where the author is the main contributor. Indeed, the author's contributions encompass conceptualization and initial ideas, the majority of the design of the schemes, properties, security models, applications, security proofs, and writing a large part of the papers. We should also mention that, in the first paper, the proofs of the ATMS's unforgeability, issuer hiding, and public key class hiding were fully carried out by the co-authors. Moreover, in the second paper, the proof of the unforgeability of the SPSEQ-UC was fully done by one of the co-authors.

1. Omid Mir, Balthazar Bauer, Scott Griffy, Anna Lysyanskaya. Daniel Slamanig "**Aggregate Signatures with Versatile Randomization and Issuer-Hiding Multi-Authority Anonymous Credentials**", 30th ACM Conference on Computer and Communications Security- ACM CCS 2023 (accepted). Cryptology ePrint https://eprint.iacr.org/2023/1016.

2. Omid Mir, Daniel Slamanig, Balthazar Bauer, René Mayrhofer, "**Practical Delegatable Anonymous Credentials From Equivalence Class Signatures**", Proceedings on Privacy Enhancing Technologies (PETS) 2023, https://petsymposium.org/popets/2023/popets-2023-0093.pdf

3. Omid Mir, Daniel Slamanig, and René Mayrhofer. **Threshold delegatable anonymous**

credentials with controlled and fine-grained delegation. IEEE Transactions on Dependable and Secure Computing, pages 1–16, doi: 10.1109/TDSC.2023.3303834, 2023.

4. Omid Mir, Michael Roland, René Mayrhofer, "**Decentralized, Privacy-Preserving, Single Sign-On**", Security and Communication Networks, vol. 2022, Article ID 9983995, 18 pages, 2022.

5. Omid Mir, Michael Roland, and René Mayrhofer. **DAMFA: Decentralized Anonymous Multi-Factor Authentication**. 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '2020) page 10–19, 2020.

6. Omid Mir, René Mayrhofer, Michael Hölzl, and Thanh-Binh Nguyen. **Recovery of encrypted mobile device backups from partially trusted cloud servers**. 13th International Conference on Availability, Reliability and Security (ARES 2018): 38:1-38:10

1.3.8 Other Contribution

In addition to the publications included in this thesis, the following works are noteworthy contributions by the author: [HRMM20, HRMM18], where the author defined the privacy model of anonymity and unlikability for users, and provided proof for them.

1.4 Structure of this Thesis

The thesis is organized into the following chapters. The technical details presented in the subsequent chapters have been extracted (mainly verbatim) from the author's aforementioned publications.

Chapter 2: This chapter provides the preliminaries and general definitions required for this thesis.

Chapter 3: This chapter introduces new primitives, namely Aggregate Signatures with Randomizable Tags and Public Keys (AtoSa) and Aggregate Mercurial Signatures (ATMS). Based on these primitives, the chapter presents the concept of Issuer-Hiding Multi-Authority Anonymous Credentials IhMA, which provides Multi-Authority (MA) settings and Issuer Hiding (Ih).

Chapter 4: This chapter introduces new primitives called equivalence classes signatures on updatable commitments (SPSEQ-UC), which serve as the main building blocks for delegatable anonymous credentials.

Chapter 5: This chapter proposes another new primitive called threshold delegatable subset predicate encryption scheme and then uses it to build threshold delegatable anonymous credentials.

Chapter 6: This chapter presents a privacy-preserving, single sign-on method that allows users to authenticate with service providers using their password and biometrics in a privacy-preserving way.

Chapter 7: This chapter proposes a secure backup of identity data which includes storing encrypted backups on cloud servers using a novel secret key reconstruction protocol. This protocol enables clients to recover their secret keys from servers by leveraging biometric authentication (e.g., fingerprint) and auxiliary devices.

Chapter 8: This chapter provides more details of a Python package that implements the proposed anonymous credential schemes, primitives, and protocols.

Chapter 9 This chapter concludes the thesis and discusses open issues that could be addressed in future research.

2 Preliminaries

In this chapter, we provide the basic preliminaries used throughout the thesis. Most of the definitions are standard and can be seen as required-basis and less well-known (or more specific) primitives are presented as building blocks when we use them in the respective chapters.

2.1 Notation

For a relation $\mathcal{R}$ over strings, we write $[x]_{\mathcal{R}}$ to denote representative x of the equivalence class for given relation $\mathcal{R}$. We mention that a relation $\mathcal{R}$ is parameterized if it is well-defined as long as some other parameters are well-defined. Given a set S, we show $x \leftarrow S$ as uniformly samples an element at random in S. With $\lambda \in N$ the main security parameter is denoted. Likewise, 1^λ is the string of λ. All algorithms, and the adversary $\mathcal{A}$, receive 1^λ as an, often implicit, input. We omit to mention the λ-input and assume that all algorithms take λ as input. $\mathcal{A}^B$ shows $\mathcal{A}$ has oracle access to B. We use $\mathcal{O}$ to denote oracles defined in games and show a negligible function as ϵ. For a positive integer N, we denote the set $\{1, \ldots, N\}$ by $[N]$ and also show the vector $\mathbf{v} = (v_1, \ldots, v_n)$. Given two vectors $\mathbf{v} = (v_1, \ldots, v_n)$ and $\mathbf{w} = (w_1, \ldots, w_m)$, we represent vector appending by writing $(\mathbf{v}, \mathbf{w})$ e, g. $\mathbf{v} = (1, 2, 3)$, $w = (4)$, and then $(\mathbf{v}, w) = (\mathbf{v}, 4) = (1, 2, 3, 4)$. Whenever we have vectors $\mathbf{w}$ and $\mathbf{v}$ of identical dimension whose components are sets, then by $\mathbf{v} \subseteq \mathbf{w}$ we mean that the relation is applied componentwise. We have a closed interval $[a, b]$ that represents the set of all natural numbers greater or equal to a and less or equal to b. We assume all algorithms are polynomial-time (PPT) unless otherwise specified and public parameters are an implicit input to all algorithms in a scheme.

2.2 Computational Assumptions

The security of many cryptographic building blocks is rooted in the hardness of well-established computational problems. Note we do not consider post-quantum assumptions in this thesis.

A cyclic group. Regarding the book [BS22], we consider the GGen algorithm that takes a security parameter λ as input and produces a group $(\mathbb{G}, g, q)$. Here, the group order of $\mathbb{G}$ is a prime q with a bit length of λ, and g serves as a generator of $\mathbb{G}$. We consider the following assumptions:

Definition 1 (Discrete Logarithm assumption (DL) [BS22])**.** *We say the* DL *holds, if for every efficient adversary $\mathcal{A}$, there exists a negligible function ϵ such that:*

$$\Pr[(\mathbb{G}, g, q) \leftarrow \mathsf{GGen}(1^\lambda), x \leftarrow Z_q, x' \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^x) : x = x'] \leq \epsilon(\lambda)$$

Definition 2 (Computational Diffie-Hellman Assumption (CDH) [BS22])**.** *Let $(\mathbb{G}, g, q)$ be a cyclic group, the* CDH *assumption holds, if for all PPT adversaries* A *there exists a negligible function ϵ such that:*

$$\Pr\left[(\mathbb{G}, g, q) \leftarrow \mathsf{GGen}(1^\lambda), (y, x) \leftarrow Z_p, h' \leftarrow \mathcal{A}(\mathbb{G}, g^x, g^y) : h' = g^{xy}\right] \leq \epsilon(\lambda).$$

Definition 3 (Decisional Diffie-Hellman Assumption (DDH) [BS22])**.** *The decisional Diffie-Hellman assumption holds, if for every efficient adversary $\mathcal{A}$, there exists a negligible function ϵ such that:*

$$\Pr[\mathsf{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathsf{A}(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1] \leq \epsilon(\lambda),$$

where either $z = xy$ or a random element.

2.3 Bilinear Pairing

Bilinear groups are a set of three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p along with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- ***Bilinearity***. for all $P \in \mathbb{G}_1$, $\hat{P} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(P^a, \hat{P}^b) = e(P, \hat{P})^{a \cdot b}$
- ***Non-degeneracy.*** for $P \in \mathbb{G}_1/\{1\}$ and $\hat{P} \in \mathbb{G}_2/\{1\}$, $e(P, \hat{P}) \neq 1$;
- ***Efficiency***. the map e is efficiently computable.

Pairings can be categorized into three types:

- Type 1: $\mathbb{G}_1 = \mathbb{G}_2$.
- Type 2: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is an efficient isomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, but no efficient one in the other direction.
- Type 3: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no efficient isomorphism, for neither $\mathbb{G}_1 \to \mathbb{G}_2$ nor $\mathbb{G}_2 \to \mathbb{G}_1$.

Note that the signature scheme and other constructions are based on type 3 bilinear groups. We will use $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \leftarrow \mathsf{BGGen}(1^\lambda)$ to denote a bilinear group generator where p is a prime of bit length λ. We use P (or g_1) and $\hat{P}$ (or g_2) as generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively throughout the thesis.

Definition 4 (External Diffie-Hellman assumption (XDH))**.** *We say that the* XDH *holds corresponding to* BGGen *if the* DDH *holds in* $\mathbb{G}_1$

Definition 5 (Symmetric External Diffie-Hellman assumption (SXDH))**.** *We say that the* SXDH *holds corresponding to* BGGen *if the* DDH *holds in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *.*

Definition 6 (Decisional Bilinear Diffie-Hellman Assumption (DBDH))**.** *The* DBDH *holds relative to* BGGen*, for all PPT algorithms* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\lambda)$ *such that*

$$\left| \begin{array}{l} \Pr\left[1 \leftarrow \mathcal{A}(P, \hat{P}, P^a, P^b, P^c, \hat{P}^a, \hat{P}^b, \hat{P}^c, e(P,\hat{P})^{abc})\right] \\ -\Pr\left[1 \leftarrow \mathcal{A}(P, \hat{P}, P^a, P^b, P^c, \hat{P}^a, \hat{P}^b, \hat{P}^c, e(P,\hat{P})^{z})\right] \end{array} \right| \leq \epsilon(\lambda)$$

PS assumption. The PS assumption is an interactive assumption, defined by Pointcheval and Sanders [PS16a] to construct a short randomizable signature known as PS signature. We take the PS definition from [PS16a] as follows:

> PS Assumption [PS16a] The PS assumption holds if no PPT adversary $\mathcal{A}$, who takes asymmetric pairing $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e)$, a tuple $(\hat{P}^x, \hat{P}^y) \in \mathbb{G}_2^2$ where x and y are random scalars in $\mathbb{Z}_p$, and unlimited access to PS oracle $\mathcal{O}^{\mathsf{PS}}(m)$ s.t. on input $m \in \mathbb{Z}_p^*$ that chooses a random $h \in \mathbb{G}_1$ and outputs the pair (h, h^{x+my}), can efficiently generate a tuple $(h^*, s^*, m^*) \in \mathbb{G}_2^n \times \mathbb{Z}_p$ such that (1) $h^* \neq 1_{\mathbb{G}_1}$ (2) $s^* = h^{x+ym^*}$, (3) $m^* \notin Q$, where Q is the list of queried messages to the $\mathcal{O}^{\mathsf{PS}}(m)$ oracle.
>
> The validity of the PS assumption tuple (h^*, s^*, m^*) can be checked as: $e(s^*, \hat{P}) = e(h^*, \hat{P}^x(\hat{P}^y)^{m^*})$.

Generalization of PS assumption (GPS). GPS is introduced by Kim et al. [KLAP20], it splits the PS oracle into two: the first oracle provides basis h picked uniformly at random and the second oracle part gets the message and h and generates the PS tuple. We take the GPS definition from [KLAP20] as follows:

> Generalized PS Assumption [KLAP20]. Given a tuple $(P^x, \hat{P}^y) \in \mathbb{G}_2^2$ and two oracles $\mathcal{O}_0^{\mathsf{GPS}}()$ and $\mathcal{O}_1^{\mathsf{GPS}}(m, h)$ such that: $\mathcal{O}_0^{\mathsf{GPS}}() \rightarrow h$, where $h \in \mathbb{G}_1$ is uniformly distributed $\mathcal{O}_1^{\mathsf{GPS}}(m, h) \rightarrow s$, where $h \in \mathbb{G}_1$, $m \in \mathbb{Z}_p$, and $s = h^{x+ym} \in \mathbb{G}_1$ as output and if $h \notin Q_0 \vee (h, \star) \in Q_1$ return $\perp$. The GPS assumption holds if no PPT adversary, $\mathcal{A}$, can find a tuple $(h^*, s^*, m^*) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that, (1) $h^* \neq 1_{\mathbb{G}_1}$, (2) $s^* = (h^*)^{x+ym^*}$, (3) $m^* \notin Q$, where $Q_1 = Q_1 \cup (h, m)$ is the list of queried to $\mathcal{O}_1^{\mathsf{GPS}}$ oracle by the adversary.

2.4 Basic Cryptographic Primitives

Here, we mention the needed basic cryptographic building blocks.

2.4.1 Digital Signature Schemes

One of the main primitives we use to construct our protocols for anonymous credentials is a digital signature. Indeed, (in most cases) credentials are digital signatures on a set of attributes and a secret user key. Consider the following definition:

Definition 7 (Digital Signature Scheme)**.** *A signature scheme with message space $\mathcal{M}$ is a tuple of the following algorithms:*

$\mathsf{Setup}(1^\lambda)$: Take as input security parameter λ, output public parameters pp.

$\mathsf{KeyGen}(\mathsf{pp})$: Take as input public parameters pp, output a key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: Take as input secret key sk and a message $m \in \mathcal{M}$, output a signature σ.

$\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: Take as input public key pk, a message m and a signature σ, output $b \in \{0, 1\}$. We show 1 as valid signature and 0 otherwise.

Security Notions. The main security goal for a signature scheme is known as existential unforgeability under chosen message attacks (EUF-CMA). In simpler terms, this means that even if an attacker can access a signing oracle, it remains challenging to produce a valid signature (m, σ) for a message m that was never previously requested from the signing oracle.

Definition 8 (EUF-CMA)**.** *A signature is said to be (EUF-CMA), if for every PPT adversary* A *there is a negligible function ϵ such that* $\Pr[\mathsf{EUF\text{-}CMA}_{\mathcal{A}}(\lambda) = 1] \leq \epsilon(\lambda)$*, where the experiment* $\mathsf{EUF\text{-}CMA}_{\mathcal{A}}(\lambda)$ *is defined in Figure 2.1.*

$\underline{\mathsf{EUF\text{-}CMA}_{\mathcal{A}}(\lambda)}$

- $Q = \emptyset$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- $(\sigma^*, m^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}}(\mathsf{pp}, \mathsf{pk})$
- Output is 1 if $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m^*, \sigma^*) = 1 \wedge m^* \notin Q$.

$\underline{\mathcal{O}^{\mathsf{Sign}}(m)}$

- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$
- $Q \cup \{m\}$, return (σ, m)

Figure 2.1: Existential Unforgeability under a Chosen-Message Attack (EUF-CMA)

2.4.1.1 Pointcheval-Sanders (PS) Signatures

The PS signature [PS16a] works on asymmetric (type 3) bilinear groups. It is EUF-CMA-secure under the PS assumption (cf. Def. 2.3). For a single scalar message it is defined as follows:

$\mathsf{Setup}(1^\lambda)$: Take the security parameter λ as input and return the public parameters $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$, where P and $\hat{P}$ are generators.

$\mathsf{KeyGen}(\mathsf{pp})$: Take pp as input, sample two random numbers $(x, y) \xleftarrow{\$} \mathbb{Z}_p^*$, and return the verification key $\mathsf{vk} = (\hat{X} = \hat{P}^x, \hat{Y} = \hat{P}^y)$ and the secret key $\mathsf{sk} = (x, y)$.

$\mathsf{Sign}(\mathsf{sk}, m)$: Take the secret key sk and a message $m \in \mathbb{Z}_p$ as input. Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$ uniformly at random and then compute $\sigma = (h, s) = (P^r, h^{x+my})$ and return the signature σ as output.

$\mathsf{Verify}(\mathsf{vk}, \sigma, m)$: To verify a signature σ, it takes the verification key vk and message m as input. If $h \neq 1$ and the pairing product equation $e(h, \hat{X} \cdot \hat{Y}^m) = e(s, \hat{P})$ holds, then it returns 1 (accept), otherwise 0 (reject).

2.4.1.2 Ghadafi SPS

Structure-preserving signatures (SPSs) [AFG+16] are signatures where public keys and the signatures are source group elements of a bilinear group, and verification will be done only using group-membership tests and pairing-product equations. SPSs are compatible with the Groth-Sahai [GS08] proofs system defined on bilinear groups. Moreover, due to the structure of signatures, it is easy to randomize and integrate them into other primitives such as ElGamal encryption. We recall the SPS scheme by Ghadafi [Gha16] which is a structure-preserving variant of PS signatures [PS16a]. Ghadafi's SPS construction is defined over a Diffie-Hellman message space $\mathcal{M}_{\mathsf{DH}}$ as:

Diffie-Hellman Message Space. Over an asymmetric bilinear group, a pair $(M, N) \in \mathbb{G}_1 \times \mathbb{G}_2$ is called a Diffie-Hellman (DH) message $\mathcal{M}_{\mathsf{DH}}$ [AFG+16] if there exists $m \in \mathbb{Z}_p$ s.t. $M = P^m$ and $N = \hat{P}^m$. One can efficiently verify whether $(M, N) \in \mathcal{M}_{\mathsf{DH}}$ by checking $e(M, \hat{P}) = e(P, N)$. Thus, the message space is a vector of Diffie-Hellman pairs $(\mathbf{M}, \mathbf{N}) = (M_1, \ldots, M_n, N_1, \ldots, N_n)$ s.t., $(M_i, N_i) = (P^{m_i}, \hat{P}^{m_i}) \in \mathcal{M}_{\mathsf{DH}}$ for $m_i \in \mathbb{Z}_p$.

The construction is defined as follows:

$\mathsf{Setup}(1^\lambda)$: Generate a bilinear group $\mathsf{pp} = (q, P, \hat{P}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$ and output pp.

$\mathsf{KeyGen}(\mathsf{pp})$: Take the pp, choose two randoms $(x, y) \xleftarrow{\$} \mathbb{Z}_p^*$, and return the verification key $\mathsf{vk} = (\hat{P}^x, \hat{P}^y)$ and the secret key $\mathsf{sk} = (x, y)$.

$\mathsf{Sign}(\mathsf{sk}, (M, N))$: Take the sk and DH message $(M, N) \in \mathcal{M}_{\mathsf{DH}}$ such that $e(M, \hat{P}) = e(P, N)$ as input. Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$ and compute the signature as

$$\sigma = (R, S, T) = (R = P^r, S = M^r, T = R^x \cdot S^y)$$

$\mathsf{Verify}(\mathsf{vk}, \sigma, (M, N))$: Take the pp, vk, signature $\sigma = (R, S, T)$ and a message $(M, N) \in \mathcal{M}_{\mathsf{DH}}$. If the following equations hold, returns 1 and 0 otherwise:

$$e(R, N) = e(S, \hat{P}) \wedge e(T, \hat{P}) = e(R, \hat{X})e(S, \hat{Y}) \wedge h \neq 1$$

2.4.1.3 Message-Indexed Ghadafi SPS

To construct our ATMS scheme, we use the version of Ghadafi SPS that is presented in [CKP+22] for signing elements of an Indexed Diffie-Hellman message space $\mathcal{M}^H_{\mathsf{iDH}}$. More precisely, Crites et al. [CKP+22] adapt the DH message space $\mathcal{M}_{\mathsf{DH}}$ to a tuple $(id, M, N) \in I \times \mathbb{G}_1 \times \mathbb{G}_2$ called Indexed Diffie-Hellman message space $\mathcal{M}^H_{\mathsf{iDH}}$, which uses a random basis h computed using a random oracle instead of P, as follows:

> The Indexed Diffie-Hellman Message Space $\mathcal{M}^H_{\mathsf{iDH}}$ is taken verbatim from [CKP+22] as follows:
>
> **Definition 9** (Indexed Diffie-Hellman Message Space $\mathcal{M}^H_{\mathsf{iDH}}$ [CKP+22])**.** *Given a bilinear group* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, \hat{g}) \leftarrow \mathsf{BGGen}(1^\lambda)$*, an index set* $\mathcal{I}$*, and a random oracle* $H : \mathcal{I} \to \mathbb{G}_1$*,* $\mathcal{M}^H_{\mathsf{iDH}}$ *is an indexed Diffie-Hellman (DH) message space if* $\mathcal{M}^H_{\mathsf{iDH}} \subset \{(id, \tilde{M}) \mid id \in \mathcal{I}, m \in \mathbb{Z}_p, \tilde{M} = (H(id)^m, \hat{g}^m) \in \mathbb{G}_1 \times \mathbb{G}_2\}$ *and the following index uniqueness property holds: for all* $(id, \tilde{M}) \in \mathcal{M}^H_{\mathsf{iDH}}$*,* $(id', \tilde{M}') \in \mathcal{M}^H_{\mathsf{iDH}}$*,* $id = id' \Rightarrow \tilde{M} = \tilde{M}'$*.*
>
> *One can define the equivalence class for each message* $\tilde{M} = (M, N) \in \tilde{\mathcal{M}}^H_{\mathsf{iDH}}$*, as* $\mathsf{EQ}_{\mathsf{iDH}}(M, N) = \{(M^r, N) \mid \exists\ r \in \mathbb{Z}_p\}$*.*

One can check the first condition by checking $e(M, \hat{P}) = e(h, N)$. The second condition guarantees that no two messages use the same index, which needs to be ensured by signers; otherwise, one can not guarantee the unforgeability of the signature. The construction is defined as follows:

$\mathsf{Setup}(1^\lambda)$: Let $\mathsf{pp} = (q, \hat{P}, P, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$. Output parameters pp.

$\mathsf{KeyGen}(\mathsf{pp})$: Choose two random $(x, y) \xleftarrow{\$} \mathbb{Z}_p^*$, returns the verification key as $\mathsf{vk} = (\hat{P}^x, \hat{P}^y)$ and the secret signing key $\mathsf{sk} = (x, y)$.

$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, (id, M, N))$: On input sk and $(id, M, N) \in \mathcal{M}^H_{\mathsf{iDH}}$ such that $e(M, \hat{P}) = e(h, N)$, where $h = H(id) = P^r$. Computes the signature as $\sigma = (h, s) = (h, s = h^x \cdot M^y)$.

$\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, \sigma, (M, N))$: Check if the following equations hold, returns 1 (verify a signature σ), otherwise it returns 0 as: $e(h, N) = e(M, \hat{P}) \wedge e(s, \hat{P}) = e(h, \hat{X})e(M, \hat{Y}) \wedge h \neq 1$.

2.4.1.4 Signatures on Equivalence Classes

The concept of Structure-Preserving Signatures on Equivalence Classes (SPSEQ) [FHS19, HS14] offers an efficient and simultaneous method for randomizing messages and signatures publicly. This is applicable when the message space comprises group-element vectors $\mathbf{M} \in (\mathbb{G}^*)^\ell$.

The SPSEQ scheme facilitates the randomization of both messages and signatures by utilizing a change of message representatives along with a corresponding signature update. To achieve this, $(\mathbb{G}^*)^\ell$ is divided into classes based on a specific equivalence relation:

$$\mathcal{R} = \{(\mathbf{M}, \mathbf{M'}) \in (\mathbb{G}^*)^\ell \times (\mathbb{G}^*)^\ell \mid \exists\, s \in \mathbb{Z}_p^* : \mathbf{M'} = s \cdot \mathbf{M}\} \subseteq (\mathbb{G}^*)^{2\ell}$$

Definition 10. *An* SPSEQ *for equivalence relation* $\mathcal{R}$ *on* $\mathbb{G}_i^*$ *(for* $i \in \{1,2\}$*) consists of the following PPT algorithms.*

We take this definition of [FHS19], but adapted to our setting.

$\mathsf{BG}_{\mathcal{R}}(1^\lambda)$: Take λ and return a bilinear group BG.

$\mathsf{KeyGen}(\mathsf{BG}, \ell)$: Take BG and a length $\ell > 1$, return a key pair $(\mathsf{sk}, \mathsf{vk})$: $\mathsf{sk} \leftarrow (x_i)_{i\in[\ell]}$, and the public key $\mathsf{vk} \leftarrow (\hat{X}_i)_{i\in[\ell]} = (\hat{P}^{x_i})_{i\in[\ell]}$.

$\mathsf{Sign}(\mathsf{sk}, \mathbf{M})$: Take a representative $\mathbf{M} \in (\mathbb{G}_i^*)^\ell$ and sk, output a signature σ for the equivalence class $[\mathbf{M}]_{\mathcal{R}}$ as follows: $\sigma = (Z \leftarrow (\prod_{i\in[\ell]} M_i^{x_i})^y, Y \leftarrow P^{\frac{1}{y}}, \hat{Y} \leftarrow \hat{P}^{\frac{1}{y}})$ for a random $y \xleftarrow{\$} \mathbb{Z}_p^*$.

$\mathsf{ChangRep}(\mathbf{M}, \sigma, \mu, \mathsf{vk})$: Take a representative $\mathbf{M} \in (\mathbb{G}_i^*)^\ell$ of class $[\mathbf{M}]_{\mathcal{R}}$, signature σ, scalar μ and public key vk, output a new signature-message pair $(\mathbf{M}', \sigma')$, where $\mathbf{M}' = \mathbf{M}^\mu$ is the new representative and σ' its updated signature as follows: pick $r \xleftarrow{\$} \mathbb{Z}_p^*$ and return $\sigma' \leftarrow (Z^{r\mu}, Y^{\frac{1}{r}}, \hat{Y}^{\frac{1}{r}})$.

$\mathsf{Verify}(\mathbf{M}, \sigma, \mathsf{vk})$: This algorithm on input a $\mathbf{M} \in (\mathbb{G}_i^*)^\ell$, signature σ, public key vk, outputs a bit 1 if $\prod_{i\in[\ell]} e(M_i, \hat{X}_i) = e(Z, \hat{Y}) \wedge e(Y, \hat{P}) = e(P, \hat{Y})$, 0 otherwise.

Within the realm of EUF-CMA security, an adversary should generate a legitimate message-signature pair associated with an unrequested equivalence class, then it is classified as a forgery.

Definition 11 (EUF-CMA)**.** *An* SPSEQ *is EUF-CMA if for all* $\ell > 1$ *and all PPT* $\mathcal{A}$*, we have:*

$$\Pr\left[\begin{array}{l|rl} \mathsf{BG} \leftarrow \mathsf{BGGen}(1^\lambda), & & \forall\, \mathbf{M} \in Q : [\mathbf{M}^*]_{\mathcal{R}} \neq [\mathbf{M}]_{\mathcal{R}} \ \wedge \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{BG}, 1^\ell), & : & \mathsf{Verify}_{\mathcal{R}}(\mathbf{M}^*, \sigma^*, \mathsf{pk}) = 1 \\ (\mathbf{M}^*, \sigma^*) \leftarrow \mathsf{A}^{\mathsf{Sign}(\cdot,\mathsf{sk})}(\mathsf{pk}) & & \end{array}\right] \leq \epsilon(\lambda) \ ,$$

Q is the queries set that $\mathcal{A}$ has asked to Sign *oracle.*

In addition to EUF-CMA, an additional security definition for SPSEQ was introduced by [FHS19] as follows:

We take this definition form [FHS19]:

Definition 12 (Signature adaptation [FHS19])**.** *Let* $\ell > 1$*. An SPS-EQ scheme* SPSEQ *on* $(\mathbb{G}_i^*)^\ell$ perfectly adapts signatures *if for all tuples* $(\mathsf{sk}, \mathsf{pk}, \mathbf{M}, \sigma, \mu)$ *with*

$$\mathsf{VKey}(\mathsf{sk}, \mathsf{pk}) = 1 \qquad \mathbf{M} \in (\mathbb{G}_i^*)^\ell \qquad \mathsf{Verify}(\mathbf{M}, \sigma, \mathsf{pk}) = 1 \qquad \mu \in \mathbb{Z}_p^*$$

$\mathsf{ChangRep}(\mathbf{M}, \sigma, \mu, \mathsf{pk})$ *and* $\mathsf{Sign}(\mathbf{M}^\mu, \mathsf{sk})$ *are identically distributed.*

2.4.1.5 Mercurial Signatures

Mercurial signatures [CL19] and signatures with flexible public keys (SFPK) [BHKS18] extend the capabilities of SPSEQ by introducing the ability for signatures to adapt not only to randomized messages but also to randomized verification keys, thereby allowing for relations $[\mathsf{vk}]_{\mathcal{R}}$ on public keys. Additionally, Mercurial signatures offer support for message randomization, similar to SPSEQ.

An additional property, which distinguishes it from SPSEQ, is called public key class-hiding. This property makes it computationally hard to differentiate between a random public key and a different public key within the relation. Similar to the message space in [FHS19], the space of public keys consists of vectors of group elements from $\mathbb{G}_2^*$. Consequently, we can define the following equivalence relations:

$$\mathcal{R}_{\mathsf{vk}} = \{(\mathsf{vk}', \mathsf{vk}) \in (\mathbb{G}_2^*)^\ell \times (\mathbb{G}_2^*)^\ell | \exists r \in \mathbb{Z}_p^* \; st. \; \mathsf{vk}' = \mathsf{vk}^r\}$$

$$\mathcal{R}_{\mathsf{sk}} = \{(\mathsf{sk}, \mathsf{sk}') \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell | \exists r \in \mathbb{Z}_p^* \; st. \; \mathsf{sk}' = \mathsf{sk} \cdot r\}$$

Formally, in addition to the SPSEQ algorithms, we also require the following algorithms, which for the scheme in [CL19] (which extends the SPS-EQ scheme from [FHS19] presented above) are as follows:

$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \to \mathsf{sk}'$: Take as input sk and a random element $\omega \in \mathbb{Z}_p^*$, output a new secret key $\mathsf{sk}' = \mathsf{sk}^\omega$.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: Take as input $\mathsf{vk} = (\hat{X}_i)_{i \in [\ell]}$ and a random element $\omega \in \mathbb{Z}_p^*$, output a new public key $\mathsf{vk}' \in [\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$ as $\mathsf{vk}' = \mathsf{vk}^\omega$.

$\mathsf{ConvertSig}(\mathsf{vk}, m, \sigma, \omega) \to \sigma'$: Take as input vk, a message $m \in \mathbb{Z}_p^*$, a signature $\sigma = (Z, Y, \hat{Y})$, and random element $\omega \in \mathbb{Z}_p$, pick $r \xleftarrow{\$} \mathbb{Z}_p^*$, this ConvertSig returns a new signature σ' s.t $\mathsf{Verify}(\mathsf{vk}', m, \sigma') = 1$ as: $\sigma' \leftarrow (Z^{r\omega}, Y^{\frac{1}{r}}, \hat{Y}^{\frac{1}{r}})$.

Mercurial signatures are origin-hiding [CL19] if in addition to the origin-hiding of ChangRep (cf. Def. 35) the following property holds. We take this definition from [CL19] as:

> **Definition 13** (Origin-hiding of ConvertSig [CL19])**.** *For all λ, for all $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all vk, for all M, σ, if $\mathsf{Verify}(\mathsf{vk}, M, \sigma) = 1$, if $\omega \leftarrow \mathbb{Z}_p$, then $\mathsf{ConvertSig}(\mathsf{vk}, M, \sigma, \omega)$ outputs a uniformly random σ and $\mathsf{ConvertVK}(\mathsf{vk}, \omega)$ outputs a uniformly random element of $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$.*

2.4.2 Public-Key Encryption Schemes

Using public-key encryption, a message m can be encrypted using a specific public key pk. The resulting ciphertext reveals no information about the content of the message as long as the related secret key sk remains secret.

Definition 14 (Public Key Encryption)**.** *A public key encryption scheme includes the following algorithms:*

$\mathsf{PPGen}(1^\lambda)$: Given a security parameter λ, this algorithm generates public parameters denoted as pp.

$\mathsf{KeyGen}(\mathsf{pp})$: Using the public parameters pp, this algorithm produces a decryption key sk and a corresponding public encryption key pk.

$\mathsf{Enc}(\mathsf{pk}, m)$: With a public encryption key pk and a message m chosen from the message space $\mathcal{M}$, this algorithm creates a ciphertext CT.

$\mathsf{Dec}(\mathsf{sk}, CT)$: Given the secret decryption key sk and the ciphertext CT, this algorithm outputs the original message m or $\perp$ if the decryption fails.

In addition to the evident correctness property, we expect a public key encryption scheme to satisfy IND-CPA security, which is formally stated below:

Definition 15 (IND-CPA)**.** *A public key encryption scheme is IND-CPA, if for all PPT* A*, there exists a negligible function* $\epsilon(\cdot)$*:*

$$\Pr[\mathsf{IND\text{-}CPA}_{\mathsf{A}} = 1] \leq \frac{1}{2} + \epsilon(\lambda)$$

The related game is depicted in Figure 2.2.

$\underline{\mathsf{IND\text{-}CPA}_{\mathcal{A}}(\lambda)}$

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- $(m_0, m_1) \leftarrow \mathsf{A}(\mathsf{pp}, \mathsf{pk})$
- $(b') \leftarrow \mathsf{A}^{\mathcal{O}^{\mathsf{Enc}}}(\mathsf{pp}, \mathsf{pk}, CT)$

$\underline{\mathcal{O}^{\mathsf{Enc}}(m_0, m_1)}$

- If $|m_0| \neq |m_1|$, return $\perp$.
- Otherwise $CT \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b)$,
- return CT

Figure 2.2: IND-CPA Security

2.4.2.1 Predicate Encryption

Definition 16 (Predicate Encryption Schemes [KSW08])**.** *A predicate encryption scheme includes the following algorithms:*

$\mathsf{Setup}(1^\lambda)$: The Setup algorithm generates a master secret key denoted as dk and its corresponding public key pk.

$\mathsf{KeyGen}(\mathsf{dk}, f)$: Given the master secret key dk and a predicate $f \in F$, the KeyGen algorithm generates a specific key denoted as dk_f.

$\mathsf{Enc}(\mathsf{pk}, \mathsf{A}, m)$: With a public key pk, a set of attributes A, and a message m from a certain message space, the Enc algorithm produces a ciphertext CT.

$\mathsf{Dec}(\mathsf{dk}_f, CT)$: Given the secret key dk_f and the ciphertext CT, the Dec algorithm decrypts the ciphertext and returns the original message m.

Correctness requires that for all $\lambda \in N$, $m \in \mathcal{M}$, $(\mathsf{pk}, \mathsf{dk}) \in \mathsf{Setup}(1^\lambda)$, $f \in F$, and $\mathsf{sk}_f \in \mathsf{KeyGen}(\mathsf{dk}, f)$, and all A:

- If $f(\mathsf{A}) = 1 \Rightarrow \mathsf{Dec}(\mathsf{dk}_f, \mathsf{Enc}(\mathsf{pk}, \mathsf{A}, m)) = m$.
- If $f(\mathsf{A}) = 0 \Rightarrow \mathsf{Dec}(\mathsf{dk}_f, \mathsf{Enc}(\mathsf{pk}, \mathsf{A}, m)) = \perp$ with

We consider the following predicate encryption in this thesis.

Subset Predicate Encryption (SPE) scheme. We take the second construction of the SPE paper [KMMS17] as follows:

$\mathsf{Setup}(1^\lambda, n)$: Setup generates a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ with generators $(P, \hat{P})$, selects a random $\alpha \in \mathbb{Z}_p^*$ and sets $h = \hat{P}^\alpha$. Samples a random secret vector $(x_1, \ldots, x_n) \in (\mathbb{Z}_p^*)^n$ and sets for all $i \in \{1, \ldots, n\}$: $X_i = P^{x_i}$. The pk and the master secret key msk are as $pk = (P, X_1, \ldots, X_n, \hat{P}, h)$, and $msk = P^\alpha$.

$\mathsf{KeyGen}(msk, pk, s)$: The KeyGen algorithm picks a random $r \in \mathbb{Z}_p$ and defines the private key associated with the set s as: $sk_s = (R = \hat{P}^r, K = P^\alpha \prod_{i \in s} X_i^r)$.

$c \leftarrow \mathsf{Enc}(pk, m, s)$: Given set s, the Enc of a message $m \in \mathbb{G}_T$ picks a random $\rho \in \mathbb{Z}_p^*$ and returns the following:

$$c = (m \cdot e(P, h)^\rho, \hat{P}^\rho, \forall i \in s : X_i^\rho)$$

$m \leftarrow \mathsf{Dec}(sk_s, pk, c)$: A ciphertext $c = (A, B, C_1, \ldots, C_\ell)$, for an integer $\ell \leq n$, can be dencrypted using the private key $sk_s = (K, R)$, return

$$m = A \cdot e(\prod_{i \in s} C_i, R)/e(K, B)$$

2.4.3 Commitment Schemes

A commitment scheme (simply commitment henceforth) allows a sender to commit to a value without revealing it and send the commitment to a receiver. From this commitment, the receiver does not learn anything about the actual value (*hiding*). The sender can then send an opening to the receiver which will reveal the committed value, but the sender cannot change their mind by opening to a value different from the one used during computing the commitment (*binding*).

Definition 17. *A commitment scheme is defined as a triple of algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$*, comprising the following components:*

- $\mathsf{Setup}(1^\lambda)$: Takes the security parameter λ and outputs the public parameter pp.
- $\mathsf{Commit}(\mathsf{pp}, m)$: Given the public parameter pp and a message m from the message space $\mathcal{M}$, this algorithm produces a commitment C along with an open value d in the form of a pair (C, d).
- $\mathsf{Open}(\mathsf{pp}, C, d)$: This algorithm takes the public parameter pp, a commitment C, and an open value d as input and deterministically outputs either the original message m or a failure symbol $\perp$.

We refer to [BS22] for the formal definitions of hiding and binding properties.

2.4.3.1 Equivocable and Extractable Commitments

Sometimes, we require a commitment to be *equivocable*, i.e., there is a trapdoor that allows cheating during the opening of a commitment, i.e., a commitment can be opened to any value (known as EQTDC). Finally, we require *extractability*, which allows the holder of this trapdoor to extract the message from any valid commitment without knowing the opening value [CF01, DMM+18].

Definition 18 (Commitment)**.** *It consists of three algorithms* $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Commit}, \mathsf{VerCom})$*:*

- $\mathsf{EQTDC.Setup}_{\mathsf{com}}(1^\lambda) \rightarrow \mathsf{crs}$: Takes the security parameter as input, and outputs some public parameters, passed via a crs to all other algorithms.
- $\mathsf{EQTDC.com}(\mathsf{crs}, m) \rightarrow (\mathsf{com}, \mathsf{decom})$: Takes a message m, public parameters crs and picks a random integer r and returns a commitment com and the corresponding opening decom.
- $\mathsf{EQTDC.VerCom}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}) \rightarrow (m, \perp)$ Takes a commitment com and the opening value decom; returns the message m, or $\perp$.

We take the formal definitions of equivocable and extractable properties from [DMM+18]:

> **Correctness** requires that for all $\mathsf{crs} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, $m \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}(\mathsf{crs}, m)$ there exists a negligible function s.t,: $\Pr[m \leftarrow \mathsf{VerCom}(\mathsf{crs}, \mathsf{com}, \mathsf{decom})] \geq 1 - \epsilon(\lambda)$.
>
> **Definition 19.** *(Equivocable and Extractable Commitments [DMM+18]). A commitment scheme is equivocable and extractable (*EQTDC*) if there exists two algorithms* $(\xi, \xi^0_{\mathsf{Eq}}, \xi^1_{\mathsf{Eq}}, \xi_{\mathsf{Ext}})$ *s.t, for all* $m \in \{0,1\}^{\mathsf{poly}(\lambda)}$*,* $(\mathsf{crs}, a) \in \xi(1^\lambda)$*,* $\mathsf{com} \in \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a)$*, and* $\mathsf{decom} \in \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}, m)$*, it holds:* $m \leftarrow \mathsf{VerCom}(\mathsf{crs}, \mathsf{com}, \mathsf{decom})$ *and the random variables*

$$\left\{\begin{array}{l}\mathsf{crs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}; (\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}(\mathsf{crs}, m) : \\ (\mathsf{crs}, \mathsf{com}, \mathsf{decom}) \ \textit{and} \\ (\mathsf{crs}, a) \leftarrow \xi(1^\lambda); \mathsf{com} \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a); \\ \mathsf{decom} \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}, m) : (\mathsf{crs}, \mathsf{com}, \mathsf{decom}) \end{array}\right\}$$

are indistinguishable. Moreover, there exists a negligible function such that for all algorithms Commit^*, for all $m \in \{0,1\}^{\mathsf{poly}(\lambda)}$:

$$\Pr\left[\begin{array}{l}(\mathsf{crs}, a) \leftarrow \xi(1^\lambda); \\ (\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}^*(\mathsf{crs}, m); \\ m' \leftarrow \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}) : \mathsf{VerCom}(crs, \mathsf{com}, \mathsf{decom}) = m'\end{array}\right] -$$

$$\Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}; \\ (\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}^*(\mathsf{crs}, m) : \\ \mathsf{VerCom}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}) = m\end{array}\right] \leq \epsilon(\lambda).$$

2.4.3.2 Pedersen Commitments

Pedersen commitments [Ped91] have a group $\mathbb{G}$ of prime order q and generators $(g_0, \ldots, g_m)$ as public parameters. For committing to the value $(z_1, \ldots, z_m) \in Z_q$, a user picks a random $r \in Z_q$ and sets $C = \mathsf{PedCom}\,(z_1, \ldots, z_m; r) = g_0^r \prod_{i=1}^m g_i^{z_i}$.

2.4.3.3 Set Commitment

Fuchsbauer et al., [FHS19] present a novel commitment that enables commitment to sets and opening of arbitrary subsets. A crucial aspect of their scheme is its support for commitment randomization, which aligns with the randomization of messages in the SPSEQ scheme, involving multiplication by a scalar.

Definition 20 (Set commitment [FHS19])**.** *A set commitment* SC *scheme includes the following algorithms. We take this definition from [FHS19]:*

$\mathsf{SC.Setup}(1^\lambda, 1^t) \rightarrow \mathsf{pp}_{\mathrm{sc}}$: The Setup algorithm takes a security parameter λ and an upper bound t for the maximum size of committed sets. It outputs public parameters denoted as $\mathsf{pp}_{\mathrm{sc}}$, which will be used as an implicit input for all subsequent algorithms.

$\mathsf{SC.Commit}(S) \rightarrow (C, O)$: Given a set S from the message space, the Commit algorithm generates a commitment C to the set S and an associated opening value O.

$\mathsf{SC.Open}(C, S, O) \to 0/1$: The Open algorithm takes a commitment C, a set S, and an opening value O. It outputs 1 if O is a valid opening of C corresponding to the set $S \in S_{\mathsf{pp_{SC}}}$, and 0 otherwise.

$\mathsf{SC.OpenSubset}(C, S, O, T) \to W$: Given a commitment C, a set S from the message space $S_{\mathsf{pp_{SC}}}$, an opening value O, and a set T, the $\mathsf{OpenSubset}$ algorithm returns $\perp$ if T is not a subset of S. Otherwise, it returns a witness W for the set T, indicating that T is a subset of the set committed to in C.

$\mathsf{SC.VerifySubset}(C, T, W) \to 0/1$: The $\mathsf{VerifySubset}$ algorithm takes a commitment C, a set T, and a witness W. If W is a valid for the set T, it outputs 1, and 0 otherwise.

We refer the reader to [FHS19] for formal definitions of the correctness, binding, hiding, and subset-soundness notions and their proofs.

On computing commitments. We note that with the knowledge of the trapdoor α, we can compute a commitment when externally provided with the randomness ρ in the group as P^ρ. If required, we will therefore modify the commitment computation to $\mathsf{SC.Commit}(S, \alpha, P^\rho)$, which then computes $C \leftarrow (P^\rho)^{f_S(\alpha)}$ and sets $O \leftarrow \perp$.[1]

2.4.4 Zero-Knowledge Proofs of Knowledge

We are exploring zero-knowledge proofs of knowledge (ZKPoK), specifically focusing on protocols for proving knowledge of a discrete logarithm. Fortunately, there exists a commonly used pattern for demonstrating knowledge of a discrete logarithm, known as the Σ-protocol. This protocol represents a three-round public-coin honest-verifier zero-knowledge proof of knowledge. In the context of ZKPoK, Σ-protocols are highly efficient implementations that can be transformed into (malicious-verifier) zero-knowledge proofs of knowledge through the application of Damgard's Technique [CDM00].

For formal definitions of zero knowledge, soundness, and completeness, we recommend referring to [BS22].

The Fiat-Shamir heuristic provides a way to convert a Σ-protocol into a non-interactive zero-knowledge proof of knowledge, denoted as NIZK. The core concept behind this heuristic is to eliminate the need for a verifier to choose a challenge and, instead, let the prover compute the challenge herself using a hash function applied to the announcement.

Camenisch and Stadler Notation [CS97]. We use the common Camenisch and Stadler

[1] Here we assume that ρ is honestly chosen, i.e., the discrete logarithm w.r.t. P is known. This can be enforced by requiring to provide a ZKPoK of the discrete logarithm ρ w.r.t. element P.

[CS97] notation for ZKPOK (or ZKPoK) as follows:

$$\mathsf{ZKPOK}\left\{(\alpha,\beta) : y = P^{\alpha} \wedge z = P^{\beta} \cdot h^{\alpha}\right\},$$

denotes an (non-) interactive proof of knowledge of discrete logarithms (α, β) (the witness) meeting the right-hand side statement about the public values y, P, z, h.

2.4.5 Secret Sharing

Shamir's Secret Sharing scheme (SSS) [Sha79a] is a randomized algorithm that on input four integers (n, t, p, s), where p is a prime, $0 < t \leq n < p$ and a secret $s \in \mathbb{Z}_p$, outputs n shares $(s_1, \ldots, s_n) \in \mathbb{Z}_p^n$ such that the following two conditions hold for any set $\mathsf{ST} = \{i_1, \ldots, i_\tau\}$:

- If $\tau \geq t$, there exists fixed (i.e., independent of s) integers $(\lambda_1, \ldots, \lambda_\tau) \in \mathbb{Z}_p^\tau$ (a.k.a. Lagrange coefficients) such that $\sum_{j=1}^{\tau} \lambda_j s_{i_j} = s \mod p$;
- If $\tau < t$, the distribution of $(s_{i_1}, \ldots, s_{i_\tau})$ is uniformly random.

Specifically, Shamir's secret sharing performs as: Choose $(a_1, \ldots, a_{t-1})$ uniformly from $\mathbb{Z}_p$. Set $p(x) := s + a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_{t-1} \cdot x_{t-1}$ and let s_i be $p(i)$ for all $i \in [n]$.

2.5 Computational Models.

We can conduct security proofs using different computational models as follows. We briefly describe them below.

Standard Model. Provable security involves establishing a formal relationship between the security of cryptographic schemes and the difficulty of solving well-known hard problems. This is achieved through reductionist security proofs, which demonstrate that breaking a cryptographic scheme is as hard as solving a well-studied problem. The standard model does not need any additional idealizing assumptions. So, it is the most promising model.

Random-Oracle Model. The random oracle model (ROM) [BR93] is a cryptographic framework that treats hash functions as if they were idealized random functions. In this model, hash functions are represented as oracles that respond with uniformly random values for each query, while consistently repeating the same responses for previously asked queries.

Universal Composition. The Universal Composition (UC) framework [Can00] allows the modular design of cryptographic protocols using "ideal functionalities," eliminating the need for explicit reductions. Ideal functionalities act as trusted third parties, defining adversary capabilities. Simulators demonstrate protocol security by making real and ideal worlds indistinguishable.

3 Issuer-Hiding Multi-Authority Credentials

In this chapter, we introduce the concept of Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA). It allows a compact and efficient showing of multiple credentials from different issuers. Another important property is called issuer hiding (IH). This means that showing a set of credentials is not revealed which issuer has issued which credentials but only whether a verifier-defined policy on the acceptable set of issuers is satisfied. This issue becomes particularly acute in the context of MA, where a user could be uniquely identified by the combination of issuers in their showing. Our proposed solution involves the development of two new signature primitives with versatile randomization features which are independent of interest: 1) Aggregate Signatures with Randomizable Tags and Public Keys (AtoSa) and 2) Aggregate Mercurial Signatures (ATMS), which extend the functionality of AtoSa to additionally support the randomization of messages and yield the first instance of an aggregate (equivalence-class) structure-preserving signature. These primitives can be elegantly used to obtain IhMA with different trade-offs but have applications beyond.

We formalize all notations and provide rigorous security definitions for our proposed primitives. We present provably secure and efficient instantiations of the two primitives as well as corresponding IhMA systems. Finally, we provide benchmarks based on an implementation to demonstrate the practical efficiency of our constructions.

3.1 Comparison of IhMA with Previous Work

We have already discussed that there is only one dedicated MA-AC scheme [HP22]. This is however not issuer-hiding (IH) and as mentioned, adding IH comes with a significant overhead. In Table 6.3.5, we compare our IhMA approaches to other schemes in the literature that provide the IH feature [BEK+21, BFGP22, CLPK22] and for comparison we use the naive approach to achieve MA, i.e., parallel showings of single credentials, which we indicate by $\approx$. We compare them in terms of the size of credential $|\mathbf{Cred}|$, communication cost of showing $|\mathbf{Show}|$, and computational cost of showing **Show** for user **(P)** and verifier **(V)**. We provide concrete analysis for our schemes' communication cost in Section 3.5.1. To ensure a fair comparison between the schemes, we consider a typical case where k out of n attributes come from K out of N issuers where n is the total number of attributes given to the user by N issuers, and k is the number of attributes involved in the showing (and K the number of issuers indicated in the showing).

With respect to credential size $|\mathbf{Cred}|$, the naive approach to **MA** leads to $O(K)$ complexity. Our $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme maintains a constant credential size even when there

Table 3.1: Comparison of AC schemes in **MA** setting (n: Attributes; k: Disclosed attributes, u: Undisclosed attributes, N: Total issuers in policy, K: issuers in showing)

	[CLPK22] ‡	[BFGP22]⋆⋆	[BEK+21]⋆⋆	$\mathsf{IhMA}_{\mathsf{AtoSa}}$	$\mathsf{IhMA}_{\mathsf{ATMS}}$
IH	✓	✓	✓	✓	✓
MA	≈	≈	≈	✓	✓
|Cred|	$O(N)$	$O(N)$	$O(N)$	$O(N)^\star$	$O(N)^\star$
|Show|	$O(K \cdot N)$	$O(k \cdot K)$	$O(k \cdot 2K)$	$O(K)$	$O(K)$
Show (P)	$O(KuN)$	$O(k \cdot K)$	$O(k \cdot 2K)$	$O(K)^\dagger$	$O(u \cdot K)$
Show (V)	$O(KkN)$	$O(k \cdot K)$	$O(k \cdot 2K)$	$O(k)$	$O(k \cdot K)$

⋆ We present the scheme in a way that supports ad-hoc attribute/issuer aggregation, but for fixed signatures, a constant size credential is achievable. For ATMS we will show how to achieve this in Section 3.4.3.
⋆⋆ K refers to proving knowledge of K credentials and K signatures of key policy in Showing.
† Since the ad-hoc aggregation cost is negligible, it is skipped here. Also, without considering IH, it becomes $O(1)$.
‡ This scheme uses standard assumptions in the ROM while other schemes use the GGM.

are $K > 1$ issuers, while our $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme has $O(K)$ credentials. However, we can aggregate credentials and then during showing apply a ZKPOK of a PS signature, which allows us to reduce the credential size to a constant size. In contrast, others have a credential size linear in the number of issuers K.

In terms of communication cost in showing (|**Show**|), our schemes require sending the randomized vks of the K issuers, along with two signatures (one for the credential and one for the key policy), overall giving $O(K)$. In [BEK+21], the communication size is based on sending K blinded credentials and K blinded signatures in the key policy and provide a ZKPOK of having correctly done so. The scheme in [BFGP22] is similar to [BEK+21], but the size of the policy is fixed. Finally, in the scheme described in [CLPK22], one needs to prove knowledge of K out of N verification keys (a linear sized OR statement) and sends them along with K credentials. Note that the size of ZKPOK includes many group elements and significantly more than only transferring K verification keys, as it is the case for our constructions.

When it comes to the computational cost of showing, i.e., **Show (P)** and **Show (V)**, our $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme has a minimal computational cost for provers as they only need to perform a small/constant number of operations for aggregation, along with K exponentiations for randomizing the verification keys vk. Our $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme involves additional computation in the creation of a witness for set commitments corresponding to undisclosed attributes (a multi-exponentiation of $O(u)$). In [BEK+21], this cost includes proving knowledge of k signatures (in the key policy), K credentials, and k disclosed attributes. Similarly, [BFGP22] requires the computation of generating witness for their aggregator (accumulator) on K credentials, proving knowledge of k credential, but it does not need to prove knowledge of signatures in the policy. Moreover, in [CLPK22], proving knowledge of K-out-of-N verification keys is necessary, along with the computation of generating witness on undisclosed attributes for set commitments on K credentials. Again, the cost of ZKPOK for credentials or committed attributes is significantly more expensive than in our

case, which is needed only to prove a secret key and some multi-exponentiation for creating witness. We should mention here that by leveraging ZKPOK, arbitrary relationships can be proved on attributes.

In summary, while the efficiency of different schemes may appear to be close asymptotically, our IhMA approaches are significantly more efficient than existing approaches while providing both properties simultaneously. Indeed, we only need group operations on $\mathbb{G}_i$ at the cost of $O(k)$. In contrast, other schemes require proving knowledge of signatures or keys, which is significantly more expensive.

3.2 Aggregate Signatures with Randomizable Keys and Tags

Now we introduce a novel primitive named AtoSa where one can aggregate signatures of different messages under different keys only if they are associated with the same tag (consisting of a private and a public part). Moreover, apart from allowing randomizing signatures, verification keys as well as tags can be randomized. Unlike mercurial signatures, our AtoSa scheme does not allow for randomization of messages. Tags and verification keys are defined with respect to equivalence classes and randomization switches between representatives of these classes. We introduce a comprehensive formal model and a construction which as a starting point takes PS signatures [PS16a]. For our AtoSa scheme we show how to integrate tags into PS signatures, use the above discussed features to make them aggregatable, and show that the key-randomization features of PS signatures (cf. [CRS+21] with $\Delta_2 = 0$) applies to our modification.

3.2.1 Formal Definitions

The public key randomization is similar to that of mercurial signatures [CL19], which allow to define equivalence classes on the key space $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}, [\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$ (cf. Section 2.4.1.5). Let a tag be $(\tau, \mathbf{T})$, where τ and $\mathbf{T}$ are the secret and public parts of tag respectively. For the tag randomization, we define equivalence classes $[\mathbf{T}]_{\mathcal{R}_\tau}$ ($[\tau]_{\mathcal{R}_\tau}$ for secret parts) on the tag space $\mathcal{T}$ similar to $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$ and $[\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$ as:

$$\mathcal{R}_\tau = \begin{Bmatrix} (\mathbf{T}', \mathbf{T}) \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell | \ \exists \mu \in \mathbb{Z}_p^* : \ \mathbf{T}' = \mathbf{T}^\mu \\ (\tau', \tau) \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell | \ \exists \mu \in \mathbb{Z}_p^* : \ \tau' = \tau \cdot \mu \end{Bmatrix}$$

We denote the space of all tags as $\mathcal{T}$ and the messages space is $\mathbb{Z}_p$. In contrast to SPSEQ (and mercurial) signatures, we do not consider equivalence classes on the message space for AtoSa.

Definition 21 (Aggregate Signatures with Randomizable Public Keys and Tag (AtoSa))**.** *An* AtoSa *for parameterized equivalence relations $\mathcal{R}_\tau$, $\mathcal{R}_{\mathsf{sk}}$ and $\mathcal{R}_{\mathsf{vk}}$, consists of the following algorithms:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter λ, output the public parameters pp.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{vk})$: On input the public parameters pp, output a key pair $(\mathsf{sk}, \mathsf{vk})$.

$\mathsf{VKeyGen}(\mathsf{sk})$: On input a secret key sk, output a verification key vk.

$\mathsf{GenAuxTag}(S) \to (\{\mathsf{aux}_j\}_{j\in[n]}, (\tau, \mathbf{T}))$: Given a message-key set $S = \{(m_j, \mathsf{vk}_j)_{j\in[n]}\}$, output auxiliary data $\{\mathsf{aux}_j\}_{j\in[n]}$ correlated to (vk_j, m_j) and a tag pair $(\tau, \mathbf{T})$, where all vk_j should be distinct.

$\mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, m_j) \to \sigma_j$: On input a secret key sk_j, tag's secret τ, auxiliary data aux_j and message $m_j \in \mathbb{Z}_p$, output a signature σ_j for $(\tau, \mathbf{T})$ and m_j under the verification key vk_j.

$\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, m_j, \sigma_j) \to \{0, 1\}$: Given a verification key vk_j, tag's public $\mathbf{T}$, message m_j and signature σ_j, output 1 if σ_j is valid relative to vk_j, m_j and $\mathbf{T}$, and 0 otherwise.

$\mathsf{AggrSign}(\mathbf{T}, \{(\mathsf{vk}_j, m_j, \sigma_j)\}_{j=1}^{\ell}) \to \sigma$: Given ℓ signatures, $(\sigma_j)_{j\in[\ell]}$ for messages $(m_j)_{j\in[\ell]}$ under verification keys, $(\mathsf{vk}_j)_{j\in[\ell]}$ on the same tag $\mathbf{T}$, output an aggregate signature σ on all messages $\mathbb{M} = (m_j)_{j\in[\ell]}$ under the tag $\mathbf{T}$ and aggregated verification key $\mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}$.

$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma) \to \{0, 1\}$: Given an aggregated verification key avk, tag $\mathbf{T}$, messages $\mathbb{M}$ and signature σ, output 1 if σ is valid relative to avk, $\mathbb{M}$ and $\mathbf{T}$, and 0 otherwise.

$\mathsf{ConvertTag}(\mathbf{T}, \mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a new randomized tag $\mathbf{T}' \in [\mathbf{T}]_{\mathcal{R}_\tau}$.

$\mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, \sigma, \mu) \to (\sigma', \mathbf{T}')$: (Randomize Signature and Tag together) Given a signature σ on a message m under tag $\mathbf{T}$ and vk, and randomness μ. Return a randomized signature and tag $(\sigma', \mathbf{T}')$ s.t $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}', m, \sigma') = 1$, where $\mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)$.

$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \to \mathsf{sk}'$: On input a sk and key converter ω, output a new secret key sk'.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: On input a vk and key converter ω, output a new public key vk'.

$\mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega) \to \sigma'$: On input a vk, message m, tag $\mathbf{T}$, signature σ, and key converter ω, return a new signature σ' s.t $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}, m, \sigma') = 1$, where $\mathsf{vk}' \leftarrow \mathsf{ConvertVK}(\mathsf{vk}, \omega)$.

We note that VKeyGen is only required in the security definition and is never used in the construction. Although the signer receives the tag secret key τ, we replace this with a ZKPOK in our IhMA scheme.

3.2.2 Security Definitions

Correctness. As usual we require that honest signatures verify as expected, but need to consider all the randomizations as well as the aggregation.

Definition 22 (AtoSa correctness)**.** *An* AtoSa *is correct if it has the following three properties:*

Basic signature correctness:
For all $\{\mathsf{sk}_i, \mathsf{vk}_i\}_{i\in[\ell]} \leftarrow (\mathsf{KeyGen}(1^k))^n$, $\{m_i\}_{i\in[\ell]} \in \mathbb{Z}_p^{*\ell}$, $(\tau, \{\mathsf{aux}_j\}) = \mathsf{GenTagIdx}(\tau, \{m_i, \mathsf{vk}_i\}_{i\in[\ell]})$ $j \in [\ell]$, we have that $\sigma = \mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, m_j)$ and $\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, m_j, \sigma) = 1$

Randomizable signature correctness:
For all $(\mathsf{sk}, \mathsf{vk}, m, \mathbf{T}, \tau, \beta, \omega, \mu, \sigma', \mathsf{vk}', \sigma^*, \mathbf{T}^*, \mathbf{T}^\dagger)$ such that $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, m, \sigma) = 1$, $\beta, \omega \in \mathbb{Z}_p^*$, $\sigma' = \mathsf{ConvertSig}(\mathsf{vk}, \sigma, \omega)$, $\mathsf{vk}' = \mathsf{ConvertVK}(\mathsf{vk}, \omega)$, $\mathsf{sk}' = \mathsf{ConvertSK}(\mathsf{sk}, \omega)$, $(\sigma^*, \mathbf{T}^*) = \mathsf{RndSigTag}(\mathsf{vk}', \mathbf{T}, m, \sigma', \beta)$, $(\mathbf{T}^\dagger) = \mathsf{C}onvertTag(\mathbf{T}, \omega)$,
the following holds:
$\mathsf{ConvertSig}(\mathsf{vk}, \sigma, \omega) = \mathsf{Sign}(\mathsf{sk}', \mathsf{aux}, \tau, m)$ (for a valid aux), $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}^*, m, \sigma^*) = 1$ and $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}^\dagger, m, \sigma') = 1$.

We've combined the definitions of all randomization functions ($\mathsf{RndSigTag}, \mathsf{ConvertTag}, \ldots$) in this definition, but for ensuring that use of a single randomization function is value, we can set the other randomization factors (β, ω) to 1 indicating no randomization to see that all the randomization functions are correct independent of each other.

Aggregatable signature correctness:
For all $\{\mathsf{sk}_i, \mathsf{vk}_i, m_i, \sigma_i, \mathbf{T}\}_{i\in[\ell]}$ such that $\forall i, \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}, m_i, \sigma_i) = 1$.

Then, the following holds:
$\sigma' = \mathsf{AggrSign}(\mathbf{T}, \{\mathsf{vk}_i, m_i, \sigma_i\}_{i\in[\ell]})$, $\mathsf{VerifyAggr}\left(a\mathsf{vk}, \mathbf{T}, \{m_i\}_{i\in[\ell]}, \sigma'\right) = 1$, where $a\mathsf{vk} = (\mathsf{vk}_i)_{i\in[\ell]}$.

Randomizable of Aggregatable signature correctness:
For all $a\mathsf{vk} = (\mathsf{vk}_i)_{i\in[\ell]}, \mathbf{T}, \{m_i\}_{i\in[\ell]}, \sigma$ and $\beta, \omega \in \mathbb{Z}_p^*$ such that $\mathsf{VerifyAggr}\left(a\mathsf{vk}, \mathbf{T}, \{m_i\}_{i\in[\ell]}, \sigma\right) = 1$, $\sigma' = \mathsf{ConvertSig}(\mathsf{vk}, \sigma, \omega)$, $i \in [\ell]$: $\mathsf{vk}_i' = \mathsf{ConvertVK}(\mathsf{vk}_i, \omega)$, $\mathsf{sk}_i' = \mathsf{ConvertSK}(\mathsf{sk}_i, \omega)$ and $(\sigma^*, \mathbf{T}^*) = \mathsf{R}ndSigTag(\mathbf{T}, a\mathsf{vk}, (m_i)_{i\in[n]}, \sigma', \beta)$. $(\mathbf{T}^\dagger) = \mathsf{C}onvertTag(\mathbf{T}, \omega)$,
Then the following holds:
$\mathsf{VerifyAggr}\left(a\mathsf{vk}', \mathbf{T}^*, \{m_i\}_{i\in[\ell]}, \sigma^*\right) = 1$ and $\mathsf{VerifyAggr}\left(a\mathsf{vk}, \mathbf{T}^\dagger, \{m_i\}_{i\in[\ell]}, \sigma'\right) = 1$, where $a\mathsf{vk}' = (\mathsf{vk}_i')_{i\in[\ell]}$.

Unforgeability. We model unforgeability following the ideas in the chosen-key model [BGLS03, LMRS04], where the adversary $\mathcal{A}$ is given a single public key vk' and access to a signing oracle on the challenge key. The adversary wins if the aggregate signature, σ, is a valid aggregate signature on a vector of messages $\mathbb{M} = (m_1, \ldots, m_n)$ under keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$,

and σ is nontrivial, i.e., the adversary did not request a signature on a m_j for $\mathsf{vk}_j = \mathsf{vk}'$ or more precisely where vk_j is in the same equivalence class as the challenge key vk'. $\mathcal{A}$ has the power to choose all public keys except the challenger's public key vk'. For our instantiation, however, we have to work in a slightly weakened model which is equivalent to the certified-keys model [LLY13, LOS+06]. In this setting the $\mathcal{A}$ registers pairs of $(\mathsf{vk}, \mathsf{sk})$ with exception of the challenge key. To model this, we have the adversary output the secret keys of the verification keys they provide in our security games. In the real world, such a key registration can be realized by requiring issuers to prove knowledge of their sk, which in the formal analysis allows a reduction to extract the secret key.

Definition 23 (Unforgeability)**.** *An* AtoSa *signature is unforgeable if for all PPT algorithms $\mathcal{A}$ having access to the oracle $\mathcal{O}^{\mathsf{Sign}()}$, there exists a negligible function ϵ such that:* $\Pr[\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda) = 1] \leq \epsilon(\lambda)$ *where the experiment* $\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda)$ *is defined in Fig. 4.1 and Q is the set of queries that $\mathcal{A}$ has issued to the $\mathcal{O}^{\mathsf{Sign}}$.*

$\underline{\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda)}$:

- $Q := \emptyset; \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
- $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(\mathsf{pp})$;
- $(j', \mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}, \mathsf{ask} = (\mathsf{sk}_j)_{j\in[\ell]\setminus j'}, \mathbb{M}^* = (m_j^*)_{j\in[\ell]}, (\tau^*, \mathbf{T}^*), \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{vk}')$
- $(\mathsf{vk}_j^*) := (\mathsf{VKeyGen}(\mathsf{sk}_j))_{j\in[\ell]\setminus j'}$,

return:

$$\begin{pmatrix} \mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}^*, \sigma^*, \mathbb{M}^*) = 1 \ \wedge \forall j \in [\ell], j \neq j' : \\ [\mathsf{vk}_j^*]_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_j]_{\mathcal{R}_{\mathsf{vk}}} \wedge \ [\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_{j'}]_{\mathcal{R}_{\mathsf{vk}}} \\ \wedge \forall (m, \mathbf{T}) \in Q : m \neq m_j^* \vee [\mathbf{T}]_{\mathcal{R}_\tau} \neq [\mathbf{T}^*]_{\mathcal{R}_\tau} \end{pmatrix}$$

$\underline{\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{aux}, (\tau, \mathbf{T}))}$:

- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}', \tau, \mathsf{aux}, m)$
- $Q = Q \cup \{m, \mathbf{T}\}$,

return σ

Figure 3.1: Experiment $\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda)$

Privacy guarantees. Similar to mercurial signatures [CL19], we define the following privacy notion for randomized keys vk and tags:

Definition 24 (Public key class-hiding)**.** *For all PPT adversaries $\mathcal{A}$, and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *there exists a negligible ϵ such that:*

$$\Pr\begin{bmatrix} (\mathsf{vk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); (\mathsf{vk}_2^0, \mathsf{sk}_2^0) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \\ r \xleftarrow{\$} \mathbb{Z}_p; \mathsf{vk}_2^1 = \mathsf{ConvertVK}(\mathsf{vk}_1, r); \mathsf{sk}_2^1 = \mathsf{ConvertSK}(\mathsf{sk}_1, r); \\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}_1,\cdot),\mathsf{Sign}(\mathsf{sk}_2^b,\cdot)}(\mathsf{vk}_1, \mathsf{vk}_2^b) : b' = b \end{bmatrix} \leq \frac{1}{2} + \epsilon(\lambda)$$

Definition 25 (Tag class-hiding)**.** *For all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that*

$$\Pr\begin{bmatrix} b \leftarrow \{0,1\}, \mathsf{BG} \leftarrow \mathsf{BGGen}(1^\lambda), \mathbf{T} \leftarrow \mathcal{T}, \mathbf{T}^{(0)} \leftarrow \mathcal{T}, \\ \mathbf{T}^{(1)} \leftarrow [\mathbf{T}]_{\mathcal{R}}, b^* \leftarrow \mathcal{A}(\mathsf{BG}, \mathbf{T}, \mathbf{T}^{(b)}) : b^* = b \end{bmatrix} - \frac{1}{2} \leq \epsilon(\lambda)$$

The tag class-hiding property for $\mathcal{R}_\tau$ is implied by the DDH *assumption.*

The following definition guarantees that a signature with tag $\mathbf{T}$ on a message m under vkoutput by ConvertSig and fed into RndSigTag produces a uniformly random signature under a uniformly random tag (from the respective tag class) and uniformly random key (from the respective key class).

Definition 26 (Origin-hiding of ConvertSig)**.** *For all λ, and $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{vk}, m, \sigma, \mathbf{T})$, if $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, m, \sigma) = 1$, and $(\omega, \mu) \in \mathbb{Z}_p^*$, then $(\sigma', \mathbf{T}') \leftarrow \mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, \mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega), \mu)$ outputs uniformly random elements in signature space and $[\mathbf{T}]_{\mathcal{R}_\tau}$) such that $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}', m, \sigma') = 1$, and $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ is a uniformly random element of $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$.*

We also require a similar definition for ConvertTag and the tag randomization:

Definition 27 (Origin-hiding of ConvertTag)**.** *For all λ, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{vk}, m, \sigma, \mathbf{T})$, if $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, m, \sigma) = 1$, and $\mu \in \mathbb{Z}_p^*$, then $(\sigma', \mathbf{T}') \leftarrow \mathsf{RndSigTag}(\mathsf{vk}, \mathsf{ConvertTag}(\mathbf{T}, \mu), m, \sigma, \mu)$ outputs uniformly random elements in the signature space and $[\mathbf{T}]_{\mathcal{R}_\tau}$ such that $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}', m, \sigma') = 1$.*

3.2.3 Construction

We construct the AtoSa scheme based on the PS signature [PS16a]. We can observe that to make PS signatures (h_i, s_i) aggregateable, we need the h_i components to be identical for all signatures to be aggregated. While in the original PSvconstruction h is a random element independently chosen during signing, this can be emulated in AtoSa by generating h for all signatures via a hash function based on some common information embedded in aux. For example, aux, could be a concatenation of all the messages and the tag. This technique was implicitly used in Coconut [SAB+19] and Camenisch et al. [CDL+20], and has recently been formalized by Crites et al. in [CKP+22].

We note that we should be careful when computing h, i.e., in choosing aux, as in PS signatures one can forge signatures when obtaining two signatures on two different messages with respect to the same element h. To prevent forgeries when aiming to aggregate signatures, a unique base h for a set of messages signed under the same tag is required. Therefore, we compute h as a hash of a concatenation of the messages to be signed and corresponding verification keys, denoted as aux. This approach ensures that every signer computes signatures on the same base h. We also introduce a new definition and function:

Aux binding. To ensure this property of h while making our construction modular, we define a straightforward property of $\mathsf{GenAuxTag}(S)$, i.e., no adversary can "open" an aux to two messages for the same signer. This definition is paired with the function VerifyAux which is called by Sign.

Definition 28 (Aux binding)**.** *We split* aux *into a preimage and an opening:* (c, o)*. For all PPT* $\mathcal{A}$*, and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda)$ *there exists a negligible* ϵ *such that:*

$$\Pr\left[\begin{array}{l}(h, \mathsf{aux} = (c, o), \mathsf{aux} = (c', o'), \tau, m, \tau', m') \leftarrow \mathcal{A}(\mathsf{vk});\\ \mathsf{VerifyAux}(\mathsf{sk}, (c, o), \tau, m) = 1 \wedge \mathsf{VerifyAux}(\mathsf{sk}, (c', o'), \tau', m') = 1;\\ c = c' \wedge ([\tau]_{\mathcal{R}_\tau} \neq [\tau']_{\mathcal{R}_\tau} \vee m \neq m')\end{array}\right] \leq \epsilon(\lambda)$$

We will then hash the preimage, c in our construction to reduce to the GPS assumption effectively. The o value in this definition may seem unnecessary, but it will become useful when we introduce our IhMA construction in Section 3.4. We've left aux binding out of our definition and rather defined it in our construction in order to make our definition more generic as aux binding is simply a property we use in the proof to ensure that our construction satisfies the definition of AtoSa.

Synchronicity assumption. We note that when we do not want to fix messages and verification keys in aux beforehand, then we can make assumption as in synchronized aggregate signatures [AGH10,HW18] and require each signer to only issue *a single signature per tag*. In this case aux only contains the tag and in the construction below we set $c = P^{\rho_1}||P^{\rho_2}$ and Definition 28 is trivially satisfied.

We involve the tag in signatures by exponentiating the component h with the secret part of the tag h^ρ and compute the component s using this value, which clearly can be checked via a pairing with the tag's public part and verified like a standard PS signature. Moreover, AtoSa allows the randomization of tag, vk and signatures via a change of representatives tag and vk and a matching signature update.

Our construction. The construction is as follows:

$\mathsf{Setup}(1^\lambda)$: Run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$ with a prime number order p, where P is a generator of $\mathbb{G}_1$, $\hat{P}$ a generator of $\mathbb{G}_2$. Pick H as a hash function: $H : \{0,1\}^* \rightarrow \mathbb{G}_1$. Output public parameters $\mathsf{pp} = \{\mathsf{BG}, H\}$.

$\mathsf{KeyGen}\,(\mathsf{pp})$: Choose $(x, y_1, y_2) \xleftarrow{\$} \mathbb{Z}_p$ and set the secret key $\mathsf{sk} = (x, y_1, y_2)$ and verification key $\mathsf{vk} = (\hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{X} = \hat{P}^x)$.

$\mathsf{VKeyGen}\,(\mathsf{sk})$: On input a secret key $\mathsf{sk} = (x, y_1, y_2)$, output $\mathsf{vk} = (\hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{X} = \hat{P}^x)$.

$\mathsf{GenAuxTag}(S)$: Given a set $S = \{(m_j, \mathsf{vk}_j)_{j\in[\ell]}\}$, choose $(\rho_1, \rho_2) \xleftarrow{\$} \mathbb{Z}_p$, set $c = P^{\rho_1}||P^{\rho_2}||(m_j, \mathsf{vk}_j)_{j\in[\ell]}$. Next set all $\mathsf{aux}_j = (c, \perp)$. Compute $h = H(c)$ and output aux and a tag pair $(\tau = (\rho_1, \rho_2), \mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2}))$.

$\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, m_j)$ Parse aux as (c, o). Check that $\tau \in c$ (i.e., that c has the form $P^{\rho_1}||P^{\rho_2}||...$) and $(m_j, \mathsf{vk}) \in c$ where vk is a verification key related to sk (in the same equivalence class). Also check that no other vk_j in aux has the same equivalence

class as sk. This can be done by checking that $\hat{Y}_2 = \hat{Y}_1^{\frac{y_2}{y_1}}$ and that $\hat{X} = \hat{Y}_2^{\frac{x}{y_2}}$. If these checks pass, it means that this is in the same equivalence class as the verifier's key. If the check doesn't pass, it means the vk_j is not in the same equivalence class.

$\mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, m_j)$: Given a $\mathsf{sk}_j = (y_{1j}, y_{2j}, x_j)$, τ, aux_j and a message m_j. If $\mathsf{VerifyAux}(\mathsf{sk}_j, \mathsf{aux}_j, \tau, m_j) \neq 1$ return $\perp$. Else, parse aux as (c, o) and compute $h = H(c)$ and output:

$$\sigma_j = (h', s_j) = \left(h' = h^{\rho_1}, s_j = (h^{\rho_1})^{x_j + y_{1j} \cdot m_j} \cdot (h^{\rho_2})^{y_{2j}}\right)$$

$\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, m_j, \sigma_j)$: Given a vk_j, tag $\mathbf{T} = (T_1, T_2)$, message m_j and signature σ_j, parse σ_j as (h', s_j) and return 1 if the following checks hold and 0 otherwise:

$$e(h', \hat{X} \cdot \hat{Y}_1^{m_1}) e(T_2, \hat{Y}_2) = (s_j, \hat{P}) \ \wedge \ T_1 = h' \neq 1_{\mathbb{G}}$$

$\mathsf{AggrSign}(\mathbf{T}, \{(\mathsf{vk}_j, m_j, \sigma_j)\}_{j=1}^{\ell})$: Given ℓ valid signatures $\sigma_j = (h', s_j)$ for m_j under vk_j and the same tag $\mathbf{T}$, where $j \in [\ell]$, outputs an aggregate signature σ on the messages $\mathbb{M} = (m_j)_{j \in [\ell]}$ under the tag $\mathbf{T}$ and aggregated verification key $\mathsf{avk} = (\mathsf{vk}_j)_{j \in [\ell]}$ as: $\sigma' = \left(h', s' = \prod_{j=1}^{\ell} s_j\right)$.

$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma)$: Given an avk, tag $\mathbf{T}$, messages $\mathbb{M}$ and aggregate signature $\sigma = (h', s)$, it outputs 1 if the following checks holds and 0 otherwise:

$$e\left(h', \prod_{j \in [\ell]} \hat{X}_j \cdot \hat{Y}_{1j}^{m_j}\right) e\left(h^{\rho_2}, \prod_{j \in [\ell]} \hat{Y}_{2j}\right) = e\left(s, \hat{P}\right) \ \wedge \ T_1 = h' \neq 1_{\mathbb{G}}$$

$\mathsf{ConvertTag}(\mathbf{T}, \mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a randomized tag $\mathbf{T}' = \mathbf{T}^{\mu} = (T_1^{\mu}, T_2^{\mu})$.

$\mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, \sigma, \mu) \to (\sigma', \mathbf{T}')$: Given a signature σ on message m under a valid tag $\mathbf{T}$ and vk, and randomness μ. Return a randomized signature σ' and a randomized tag:

$$\sigma' = \left(h'^{\mu}, s^{\mu}\right), \quad \mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)$$

where is a valid signature for a new tag representative $\mathbf{T}' \in [\mathbf{T}]_{\mathcal{R}_\tau}$.

$\mathsf{ConvertSK}(\mathsf{sk}, \omega)$: On input sk and a key converter $\omega \in \mathbb{Z}_p^*$, output a new secret key sk' as $\mathsf{sk}' = \mathsf{sk} \cdot \omega$.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega)$: On input $\mathsf{vk}$and a key converter $\omega \in \mathbb{Z}_p^*$, output a new public key as $\mathsf{vk}' = \mathsf{vk}^{\omega}$.

$\mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega)$: On input a vk, message m, signature σ, tag $\mathbf{T}$, and key converter $\omega \in \mathbb{Z}_p^*$, return a new signature σ' s.t. $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}, m, \sigma') = 1$, where $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ as follows: $\sigma' = (h', s' = s^{\omega})$.

The correctness of our construction follows from inspection. We formally show the unforgeability and privacy notations.

Theorem 3.2.1 (Unforgeability)**.** *Our construction achieves the EUF-CMA security stated in Def 23, under the hardness of* GPS *assumption, stated in Def. 2.3 in the random oracle model.*

Proof. To simplify our proof and make it more readable, we split our proof via two lemmas such that Lemma 3.2.2 indicates that the aggregate signature with a randomizable tag is secure in the RO model (without considering randomizable keys). To realize Lemma 3.2.2, we modify the game in Definition 23 to only output one if the forgery is on the honest signer's exact key, $vk' = vk_j$. Lemma 3.2.3 stands for randomizable verification keys property and shows that if the aggregate signature with a randomizable tag is secure, meaning lemma 3.2.2 is correct, we can achieve an aggregate signature with randomizable verification keys. WLOG, we assume the game only outputs one if the forgery is on the honest signer's exact key $\mathsf{vk}' = \mathsf{vk}_{j'}$ for an index j'.

Lemma 3.2.2 (Aggregate Signatures with Randomizable Tags)**.** *Let $\mathcal{A}$ be an adversary against the EUF-CMA security of the aggregate signature scheme (Def. 23). If* GPS *assumption holds, then our construction in Section 3.2.3 is unforgeable when $\mathcal{A}$ outputs a forgery on an exact honest verification key instead of an equivalent one. This means that after interacting with the EUF-CMA challenger, no PPT adversary can produce $(\mathsf{avk} = (\mathsf{vk}_j), \mathsf{ask} = (\mathsf{sk}_j), \mathbb{M}^* = (m_i^*), \hat{\tau}^*, \sigma^*)_{j\in[\ell]}$ s.t:*

An $\mathcal{A}$ interacting with the EUF-CMA challenger produces: $(j', \mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}, \mathsf{ask} = (\mathsf{sk}_j)_{j\in[\ell]\setminus j'}, \mathbb{M}^* = (m_j^*)_{j\in[\ell]}, (\tau^*, \mathbf{T}^*), \sigma^*)$ That adversary has defeated the EUF-CMA game if their output satisfies:

$$\begin{pmatrix} \mathsf{VerifyAggr}\,(\mathsf{avk}, \mathbf{T}^*, \sigma^*, \mathbb{M}^*) = 1 \wedge \\ \forall j \in [\ell], j \neq j' : [\mathsf{vk}_j^*]_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_j]_{\mathcal{R}_{\mathsf{vk}}} \wedge \\ [\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_{j'}]_{\mathcal{R}_{\mathsf{vk}}} \wedge \forall (m, \tau) \in Q : m \neq m_j^* \vee [\mathbf{T}^*] \neq [\mathbf{T}] \end{pmatrix}$$

Proof. We construct a reduction $\mathcal{B}$ using $\mathcal{A}$ against the GPS assumption (Def. 2.3). The challenger of latter game will be denoted by $\mathcal{C}$. We answer to the random oracle $H(c)$ by calling $\mathcal{O}_0^{\mathsf{GPS}}(c)$ to generate base h where c is part of aux from Definition 28. This is a similar call to $\mathcal{O}_0^{\mathsf{GPS}_3}(id)$ in Definition 2.3 but replacing id with c.

The reduction will continue by using the given challenge key from the GPS challenger to sign either messages or tags. The insight for why this proof works comes from the fact that our signature is exactly a multi-message signature (from [PS16a]) on m and $\frac{\rho_2}{\rho_1}$ randomized by ρ_1. Our proof of security will be similar to the proof of multi-message security in [PS16a].

Setup: $\mathcal{B}$ receives from $\mathcal{C}$ values $(\hat{X} = \hat{P}^x, \hat{Y} = \hat{P}^y)$ and pp of BG. $\mathcal{B}$ then computes $\alpha_1, \beta_1, \alpha_2, \beta_2$ and values: $\hat{Y}_1 = \hat{Y}^{\alpha_1}\hat{P}^{\beta_1}, \hat{Y}_2 = \hat{Y}^{\alpha_2}\hat{P}^{\beta_2}$. $\mathcal{B}$ then computes the challenge key for AtoSa as $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2)$ and gives this to the adversary.

$\mathsf{Q}ueries$: When $\mathcal{A}$ asks a signature query on a tag τ, m, $\mathsf{aux} = (c, o)$, s.t $\mathsf{V}erifyAux(\mathsf{sk}, \mathsf{aux}, \tau, m) = 1$, $\mathcal{B}$ computes a signature as follows:

- $\mathcal{B}$ first requests from $\mathcal{C}$ a base $h = H(c)$, by calling $h \leftarrow \mathcal{O}_0^{\mathsf{G}PS}(c)$ which is also a RO response. $\mathcal{C}$ (or RO) response as follows: if $Q_0[c] = \perp$, pick $r \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $Q_0[c] \leftarrow h = P^r$: return $Q_0[c]$, where Q_0 is the list of queried messages to $\mathcal{O}_0^{\mathsf{G}PS}$ (or RO). Note that if $h \in Q_1$ we return $\perp$.
- $\mathcal{B}$ requests from $\mathcal{C}$ to compute s for a message $m^\dagger = \alpha_1 m + \alpha_2 \frac{\rho_2}{\rho_1}$ by calling $s \leftarrow \mathcal{O}_1^{\mathsf{G}PS}(m^\dagger, h)$. $\mathcal{C}$ computes this as $s = h^{x+(\alpha_1 m+\alpha_2 \frac{\rho_1}{\rho_2})y}$. $\mathcal{B}$ then computes $\sigma = \left(h' = h^{\rho_1}, s' = \left(s * h^{\beta_1 m+\beta_2 \frac{\rho_2}{\rho_1}}\right)^{\rho_1}\right)$ and returns this to the adversary. We can see that this verifies with the vk' we gave the adversary earlier.

$$\sigma = \left(h' = h^{\rho_1}, s' = \left(h^{\rho_1(x+(\alpha_1 y m+\alpha_2 \frac{\rho_2}{\rho_1})y+\beta_1 m+\beta_2 \frac{\rho_2}{\rho_1})}\right)\right)$$

Using the equations from Sign (removing the degeneracy check):

$$e(h', \hat{X} * \hat{Y}_1^m) e(T_2, \hat{Y}_2) = e(s', \hat{P})$$

$$= e(h^{\rho_1}, \hat{P}^x * \hat{P}^{(\alpha_1 y+\beta_1)m}) e(h^{\rho_2}, \hat{P}^{\alpha_2 y+\beta_2}) = e(s', \hat{P})$$

$$= e(h, \hat{P})^{\rho_1 * (x+(\alpha_1 y+\beta_1)m} e(h, \hat{P})^{\rho_2 * (\alpha_2 y+\beta_2)} = e(s', \hat{P})$$

$$= e(h, \hat{P})^{\rho_1 * (x+(\alpha_1 y+\beta_1)m)+\rho_2 * (\alpha_2 y+\beta_2)} = e(s', \hat{P})$$

$$= e(h, \hat{P})^{\rho_1 * (x+m\alpha_1 y+m\beta_1 m+\frac{\rho_2}{\rho_1}\alpha_2 y+\frac{\rho_2}{\rho_1}\beta_2)} = e(s', \hat{P})$$

If we rearrange s' we can see this is the same so the signature verifies correctly:

$$e(s', \hat{P}) = e(h, \hat{P})^{\rho_1(x+\alpha_1 y m+\beta_1 m+\alpha_2 \frac{\rho_2}{\rho_1} y+\beta_2 \frac{\rho_2}{\rho_1})}$$

- $\mathcal{A}$ then repeats a polynomial number of signing queries adaptively.

$\mathsf{O}utput$: Eventually, $\mathcal{A}$ outputs a forgery as $(j', h'^*, s^*, \tau^* = (\rho_1^*, \rho_2^*, h^{\rho_1^*}, h^{\rho_2^*}))$ on messages $\mathbb{M}^* = (m_1^*, \ldots, m_n^*)$ under the keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. From the definition, we know that for an index j', a tuple $(m_{j'}^*, \sigma_{j'}^*)$ should be the new signature-message pair under $\mathsf{vk}_{j'} = \mathsf{vk}'$ that is aggregated in σ^* such that $\mathcal{A}$ has never queried both $m_{j'}^*$ and τ^* together. The adversary has also output all other secret keys except for the challenge key. This allows us to isolate this key:

leftmirgin=* $\mathcal{B}$ cancels the tag out from the aggregate signature which is a new tuple as:

$$\sigma^*_{j'} = \left(h^*, s^*_{j'} = \frac{s^*}{\prod_{j \in [\ell] \setminus j'} (h^*)^{x_j + y_{1j} m_j + y_{2j} \frac{\rho^*_2}{\rho^*_1}}} (h^*)^{-\sum \beta_j m^*_j} \right)$$

This signature should now satisfy: $e(h^*, \hat{X}\hat{Y}^{\alpha_1 m^*_{j'} + \alpha_2 \frac{\rho^*_2}{\rho^*_1}}) = e(s^*_{j'}, \hat{P})$.

Send $(\sigma^*_{j'} = (h^*, s^*_{j'}), \alpha_1 m^*_{j'} + \alpha_2 \frac{\rho^*_2}{\rho^*_1})$ under $\mathsf{vk} = (\hat{X}, \hat{Y})$ as a valid GPS response to $\mathcal{C}$.

Because this verifies on the challenge key for a message $\alpha_1 m^*_{j'} + \alpha_2 \frac{\rho^*_2}{\rho^*_1}$, it will be a valid forgery if this message were never queried previously. We can see that, after fixing a challenge key $\hat{Y}$, then $\forall \alpha_1, \alpha_2, \hat{Y}_1, \hat{Y}_2 \in \mathbb{G}_2$, $\exists \beta_1, \beta_2$ s.t. $\hat{Y}_1 = \hat{Y}^{\alpha_1} \hat{P}^{\beta_1}, \hat{Y}_2 = \hat{Y}^{\alpha_2} \hat{P}^{\beta_2}$. This can be seen by setting $\hat{P}^{\beta_1}$ to be $\hat{Y}^{-1}\hat{Y}_1$ (and similar for $\hat{P}^{\beta_2}$). This value for β_1 isn't possible to compute in polynomial time, but it still exists and each choice of β_1 is just as likely to be chosen via random coins as any other element. Further, the distribution of vk values resulting from the choice of β_1, β_2 is uniform. Thus, because of β_1 and β_2, the adversary's view is independent of α_1 and α_2. In the space of message/tag pairs, we have only p^3 sets of pairs $((m, \frac{\rho_2}{\rho_1}), (m', \frac{\rho'_2}{\rho'_1}))$ that satisfy this equation:

$$\alpha_1 m + \alpha_2 \frac{\rho_2}{\rho_1} = \alpha_1 m' + \alpha_2 \frac{\rho'_2}{\rho'_1} \tag{3.1}$$

(where p is the order of the group). This is because for each combination of $m, \frac{\rho_2}{\rho_1}, m'$, there is a specific value for $\frac{\rho'_2}{\rho'_1}$ that completes the set. Thus there are only p^3 distinct sets that meet Equation 3.1. Notice that there are p^4 of these sets without the restriction in Equation 3.1. The adversary samples these sets at random when issuing queries since their view is independent of the chosen α_1, α_2. In the end, the reduction will only fail if a we find a set that satisfies Equation 3.1 in the adversary's signature queries. Note that the adversary cannot entirely benefit from his or her polynomial number of queries since the pair must contain the adversary's outputted forgery and the adversary's view is independent of α_1, α_2 so their choice of which message to output must be random. Thus, the adversary outputs a forgery $m^*_{j'}, \tau^*$ which forms a pair with each q query issued previously (where q is the polynomial number of signing queries). Thus, the chance that our adversary outputs a forgery that meets Equation 3.1 with a previous query (which would mean our reduction does not constitute a forgery) is $\frac{p^3 * q}{p^4}$ which is negligible since p is exponential and q is polynomial in the security parameter.

Note that we never ask $\mathcal{O}_1^{\mathsf{GPS}}$ for a second signature on any given h. This is because of Aux binding (Definition 28). The value we pass to $\mathcal{O}_0^{\mathsf{GPS}}$ is based on the messages and tags we sign. Thus, if the adversary asks for a second signature on a particular message/tag

pair, the resulting h will either be the same (meaning we can simply return the previous signature) or be a fresh result from $\mathcal{O}_0^{\mathsf{GPS}}$, meaning that this h has not been seen before.

Lemma 3.2.3 (Aggregate Tag based Signatures with randomizable Keys)**.** *An adversary cannot produce a valid forgery in Definition 38 without querying the corresponding randomization of the challenge verification key,* vk'*, thus allowing a reduction to extract this randomization and de-randomize the signature to verify with this key.*

To prove the randomization (flexible) public keys property, we follow proof of convert Mercurial signature [CL19]. Assume there exists a generic group, PPT algorithm $\mathcal{A}$ that can break the unforgeability of aggregate signature scheme randomizable tag and public keys; that is, when given an honest verifier key, vk', $\mathcal{A}$ is able to produce a forgery $(\mathsf{avk} = \{\mathsf{vk}_i^*\}, \mathsf{ask}, \tau^*, \mathbb{M}^*, \sigma^*)$ that satisfies the following conditions with non-negligible probability:

$$[\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_1^*]_{\mathcal{R}_{\mathsf{vk}}} \wedge \forall m \in Q, m_1^* \neq m \wedge$$
$$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}^*, \mathbb{M}^*, \sigma^*) = 1$$

Where WLOG, vk_1^* is in the same equivalence class as vk' (with this construction, the adversary's forgery can always be rearranged to produce a forgery like this). The fact that vk_1^* belongs to the same equivalence class as vk' implies that there exists some $\alpha \in \mathbb{Z}_p^*$ such that $\mathsf{vk}' = \mathsf{vk}_1^\alpha$. We can construct a PPT reduction, $\mathcal{B}$ that creates a forgery for an aggregate tag based signature scheme using $\mathcal{A}$, then use Lemma 3.2.2 to prove our construction secure. The challenger $\mathcal{C}$ in the tag based signature unforgeability game for $\mathcal{B}$ chooses values $(x, y_1, y_2) \xleftarrow{\$} \mathbb{Z}_p^*$, sets $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2) = (\hat{P}^x, \hat{P}^{y_1}, \hat{P}^{y_2})$, and forwards vk' to $\mathcal{B}$.

On input vk', $\mathcal{B}$ operates as follows:

- $\mathcal{B}$ forwards vk' to $\mathcal{A}$ and runs $\mathcal{A}(\mathsf{vk}')$. $\mathcal{B}$ forwards $\mathcal{A}$'s signature queries to the AtoSa (with inflexible public keys) challenger and forwards the results to $\mathcal{A}$. $\mathcal{B}$ also services and records $\mathcal{A}$'s GGM queries.
- $\mathcal{B}$ obtains $\mathcal{A}$'s forgery $(\mathsf{avk} = \{\mathsf{vk}_i^*\}, \mathbb{M}^* = \{m_i^*\}, \sigma^*, \tau^*)$.
- If, via this process, it is possible for $\mathcal{B}$ to obtain α, $\mathcal{B}$ can remove α and outputs $(\mathsf{avk}' = \{(\mathsf{vk}_i^*)^\alpha\}, \mathbb{M}^*, \sigma' = (h, (s^*)^\alpha), \tau^*)$ as his forgery; else, $\mathcal{B}$ outputs $\perp$.

Now, let us analyze this reduction. One of these vk_i^* is in the same equivalence class as vk'. WLOG, we'll say this is vk_1^*.

Claim 3.2.3.1. *If* $[\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_1^*]_{\mathcal{R}_{\mathsf{vk}}}$*, then the generic group model reduction* $\mathcal{B}$ *can obtain* $\alpha \in \mathbb{Z}_p^*$ *such that* $\mathsf{vk}' = (\mathsf{vk}_1^*)^\alpha$*.*

Proof. Initially, before any queries are made, the elements of $\mathbb{G}_2$ that $\mathcal{A}$ has seen are $\hat{P}$ and $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2)$. Any output in $\mathbb{G}_2$ by the adversary must be from a GGM query of the form:

$$P^{\alpha_1} * \hat{X}^{\alpha_x} * \hat{Y}_1^{\alpha_{y_1}} * \hat{Y}_2^{\alpha_{y_2}}$$

Where $\alpha_1, \alpha_x, \alpha_y \in \mathbb{Z}_p$. We can rewrite this as:

$$P^{\alpha_1 + x\alpha_x + y_1\alpha_{y_1} + y_2\alpha_{y_2}}$$

This must be the form of the adversary's output, $\mathsf{vk}_1^* = \hat{X}_1^*, \hat{Y}_1^*$.

$$\hat{X}_1^* = P^{\alpha_1^{(\hat{X}^*)} + x\alpha_x^{(\hat{X}^*)} + y_1\alpha_{y_1}{}^{(\hat{X}^*)} + y_2\alpha_{y_2}{}^{(\hat{X}^*)}}$$

$$\hat{Y}_{1,1}^* = P^{\alpha_1^{(\hat{Y_1}^*)} + x\alpha_x^{(\hat{Y_1}^*)} + y_1\alpha_{y_1}{}^{(\hat{Y_1}^*)} + y_2\alpha_{y_2}{}^{(\hat{Y_1}^*)}}$$

$$\hat{Y}_{2,1}^* = P^{\alpha_1^{(\hat{Y_2}^*)} + x\alpha_x^{(\hat{Y_2}^*)} + y_1\alpha_{y_1}{}^{(\hat{Y_2}^*)} + y_2\alpha_{y_2}{}^{(\hat{Y_2}^*)}}$$

Where, for example, $\alpha_{y_1}^{(\hat{X}^*)}$ is the adversary's coefficient for the secret value, y_1, when computing their forgery verification key, $\hat{X}^*$. We want to prove that $\alpha_1^{(\hat{X}^*)}$, $\alpha_1^{(\hat{Y_1}^*)}$, $\alpha_1^{(\hat{Y_2}^*)}$, $\alpha_x^{(\hat{Y_1}^*)}$, $\alpha_x^{(\hat{Y_2}^*)}$, $\alpha_{y_1}^{(\hat{Y_2}^*)}$, $\alpha_{y_1}^{(\hat{X}^*)}$, $\alpha_{y_2}^{(\hat{X}^*)}$, and $\alpha_{y_2}^{(\hat{Y_1}^*)}$ are zero and $\alpha_x^{(\hat{X}^*)}$ is equal to $\alpha_{y_1}^{(\hat{Y_1}^*)}$ and $\alpha_{y_2}^{(\hat{Y_2}^*)}$. If so, we will know that $\alpha_x^{(\hat{X}^*)} = \alpha_{y_1}^{(\hat{Y_1}^*)} = \alpha_{y_2}^{(\hat{Y_2}^*)} = \alpha$ and we can compute the forgery for the AtoSa game. We can think of the exponents as polynomials:

$p_{\hat{X}}^*(x, y) = \alpha_1^{(\hat{X}^*)} + x\alpha_x^{(\hat{X}^*)} + y_1\alpha_{y_1}^{(\hat{X}^*)} + y_2\alpha_{y_2}^{(\hat{X}^*)}$,
$p_{\hat{Y_1}}^*(x, y) = \alpha_1^{(\hat{Y_1}^*)} + x\alpha_x^{(\hat{Y_1}^*)} + y_1\alpha_{y_1}^{(\hat{Y_1}^*)} + y_2\alpha_{y_2}^{(\hat{Y_1}^*)}$,
$p_{\hat{Y_2}}^*(x, y) = \alpha_1^{(\hat{Y_2}^*)} + x\alpha_x^{(\hat{Y_2}^*)} + y_1\alpha_{y_1}^{(\hat{Y_2}^*)} + y_2\alpha_{y_2}^{(\hat{Y_2}^*)}$,

Signatures are exclusively in $\mathbb{G}_1$, so the adversary does not learn any more elements in $\mathbb{G}_2$. If the adversary outputs a vk_1^* where $\alpha_1^{(\hat{X}^*)} \neq 0$, $\alpha_{y_1}^{(\hat{X}^*)} \neq 0$, or $\alpha_{y_2}^{(\hat{X}^*)} \neq 0$, then $\hat{X}_1^*$ and $\hat{X}'$ (in vk_1^* and vk') are the result of queries to the GGM of distinct polynomials in $\mathbb{Z}_p[x, y_1, y_2]$ where p is the size of the groups of the bilinear pairing. There is a similar arguement for $\hat{Y}_{1,1}^*$ and $\hat{Y}_{2,1}^*$. Thus, according to the Schwartz-Zippel lemma, the chance that these two polynomials evaluate to the same value when x, y are chosen randomly, is negligible. Thus, if we have $\mathcal{B}$ output random encodings independent of x, y_1, y_2 and later define x, y_1, y_2, there is a negligble chance that $\mathcal{A}$ can compute a non-zero value for $\alpha_1^{(\hat{X}^*)}$, $\alpha_1^{(\hat{Y_1}^*)}$, $\alpha_1^{(\hat{Y_2}^*)}$, $\alpha_{y_1}^{(\hat{X}^*)}$, $\alpha_{y_2}^{(\hat{X}^*)}$, $\alpha_{y_1}^{(\hat{X}^*)}$, $\alpha_{y_1}^{(\hat{Y_2}^*)}$, $\alpha_{y_2}^{(\hat{X}^*)}$, or $\alpha_{y_2}^{(\hat{Y_1}^*)}$ where the resulting vk_1^* is still in the equivalence class of vk'. This proves claim 1.

After proving that only $\alpha_x^{(\hat{X}^*)}$, $\alpha_{y_1}^{(\hat{Y_1}^*)}$ and $\alpha_{y_2}^{(\hat{Y_2}^*)}$ are non-zero, it holds that $\alpha_x^{(\hat{X}^*)} = \alpha_{y_1}^{(\hat{Y_1}^*)} = \alpha_{y_2}^{(\hat{Y_2}^*)} = \alpha$ as, otherwise, $[\mathsf{vk}_1^*]_{\mathcal{R}} \neq [\mathsf{vk}']_{\mathcal{R}}$.

We can see from the Verify algorithm that, if the reduction can recover α such that $\mathsf{vk}_1^* = (\mathsf{vk}')^{\alpha}$, and $(\mathsf{avk}, m^*, \sigma^*, \tau^*)$ is a valid forgery for our AtoSa scheme with randomizable public keys, then $(\mathsf{avk}', m^*, \sigma', \tau^*)$ is a valid forgery for our AtoSa scheme without randomizable public keys:

$\mathsf{avk} = \{\hat{X}_j, \hat{Y}_{1,j}, \hat{Y}_{2,j}\}$,
$\mathsf{avk}' = \{\hat{X}_j^{\frac{1}{\alpha}}, \hat{Y}_{1,1}^{\frac{1}{\alpha}}, \hat{Y}_{2,1}^{\frac{1}{\alpha}}\}$,

$$e(h, \prod_{i\in[\ell]} \hat{X}_j * \hat{Y}_{1,j}^{m_i}) e(h^{\rho_2}, \prod_{j\in[n]} \hat{Y}_{2,j}) = e(s, \hat{\tau}^*),$$
$$e(h, \prod_{i\in[\ell]} \hat{X}_j^{\frac{1}{\alpha}} * \hat{Y}_{1,j}^{\frac{1}{\alpha} m_i}) e(h^{\rho_2}, \prod_{j\in[n]} \hat{Y}_{2,j}^{\frac{1}{\alpha}}) = e(s^{\frac{1}{\alpha}}, \hat{\tau}^*),$$

We know that $(\mathsf{vk}_1^*)^{\frac{1}{\alpha}} = \mathsf{vk}'$ and so we can use Lemma 1 with $\mathsf{avk}^\dagger = \mathsf{avk}^{\frac{1}{\alpha}}, \sigma^\dagger = (h, s^{\frac{1}{\alpha}})$ to reduce this to breaking the GPS.

Theorem 3.2.4 (Privacy)**.** *Our construction is origin-hiding of ConvertSig, origin-hiding of* RndSigTag*, tag class hiding and has public key class-hiding based on Def. 26, Def. 27, Def. 25, and Def. 24, respectively.*

The proof of tag class-hiding follows exactly from the message class hiding of the FHS scheme in [FHS19]. The proof of other properties is provided in our paper [MBG+23].

3.3 Aggregate Mercurial Signatures With Randomizable Tags

We now present an aggregate mercurial signature with randomizable tags (ATMS). Similar to AtoSa, (see Def. 21), one can aggregate mercurial signatures of different messages under different keys under the same tag and randomize those signatures, public keys, and tags. ATMS differs from AtoSa by in addition supporting equivalence classes on the message space. This further allows the randomization of messages, leading to a feature known from structure-preserving signature on equivalence classes (SPSEQ) and, more precisely, mercurial signatures.

To achieve the aggregation property, we follow the strategy presented by Crites et al. in context of threshold SPS [CKP+22], where the authors define a so called Indexed Diffie-Hellman message space $\mathcal{M}^H_{\mathsf{iDH}}$. But the main problem with this approach, as it is defined over both groups, is that we can not define indistinguishable equivalence classes over $\mathbb{G}_1^k \times \mathbb{G}_2^k$, since spanning both groups makes DDH easy and would yield trivial linkability. Note that given both $((M_1, M_2), (N_1, N_2))$ and $((M_1', M_2'), (N_1', N_2'))$, one can easily link them together by checking $e(M_1, N_2') = e(M_2, N_1')$ and $e(M_1', N_2) = e(M_2', N_1)$ holds. So we adapt $\mathcal{M}^H_{\mathsf{iDH}}$ and define a new message space called a Tag-based DH message space $\mathcal{M}^H_{\mathsf{TDH}}$ and its corresponding EQ relation. We essentially define one equivalence class per group and tie them together via the message, the tag, and an index obtained via some auxiliary information (similar to the aux in the case of AtoSa). Indeed we adapt the Diffie-Hellman message space $\mathcal{M}_{\mathsf{DH}}$ to a Tag-based DH message space $\mathcal{M}^H_{\mathsf{TDH}}$ for a tuple $(\mathsf{aux}, h, \mathbf{T}, M, N)$, which includes a tag $\mathbf{T}$ with auxiliary data aux (instead of the *id*).

This new message space then allows us to aggregate and define an equivalence (EQ) relation which gives an indistinguishable message space.

3.3.1 Formal Definitions

We begin our definitions by introducing Tag-based DH message space $\mathcal{M}^H_{\mathsf{TDH}}$ and give an instantiation in the random oracle model (ROM). Then we define a new EQ relation

regarding this message space $\mathcal{M}^H_{\mathsf{TDH}}$, and finally, we define our new primitive ATMS.

A Tag-based DH message space. We adapt the message indexing technique introduced by [CKP+22] (cf. Def. 9) to tags:

Definition 29 (A Tag-based DH message space ($\mathcal{M}^H_{\mathsf{TDH}}$))**.** *Let H be a random oracle. For the aux and tag $\mathbf{T} = (h^{\rho_i})_{i\in[k]}$, we define $\mathcal{M}^H_{\mathsf{TDH}}$ as a tag based DH message space, if the following property holds: For the messages vector $(\mathbf{M}, \mathbf{N}) = (M_1, \ldots, M_k, N_1, \ldots, N_k)$ there exists $m_i \in \mathbb{Z}_p$ s.t. for each tuple $(\mathsf{aux}, T_i = h^{\rho_i}, M_i = T_i^{m_i}, N_i = \hat{P}^{m_i})$, the following holds: $e(M_i, \hat{P}) = e(T_i, N_i)$.*

We provide an instantiation in Fig. 3.2. Let us assume WLOG a message vector with the length $k = 2$ as $\mathbf{m} = (m_1, m_2)$, this can be generalized to any length $k > 1$.

$\underline{\mathcal{M}^H_{\mathsf{TDH}}(\mathbf{T} = (h^{\rho_1}, h^{\rho_2}), \mathsf{aux}, \mathbf{m})}$:

- $h \leftarrow H\,(\mathsf{aux})$
- for $i \in [2]$:
  - $M_i \leftarrow h^{m_i \rho_i}$
  - $N_i \leftarrow \hat{P}^{m_i}$
- **return** $(\mathbf{M}, \mathbf{N})$

$\underline{H\,(\mathsf{aux})}$:

- If $Q_H[\mathsf{aux}] = \perp$:
- $r \xleftarrow{\$} \mathbb{Z}_p$
- $Q_H[\mathsf{aux}] \leftarrow P^r := h$
- **return** $Q_H[\mathsf{aux}]$

Figure 3.2: Tag based Diffie-Hellman message space in ROM

Equivalence relations (EQ) over $\mathcal{M}^H_{\mathsf{TDH}}$. Let the message space $\mathcal{M}^H_{\mathsf{TDH}}$ be defined as $(\mathbf{M}, \mathbf{N}) = (M_1, \ldots . M_k, N_1, \ldots, N_k) \in (\mathbb{G}_1^*)^k \times (\mathbb{G}_2^*)^k$ such that for $(h, \mathbf{T})$, and $i \in [k]$: $e(M_i, \hat{P}) = e(T_i, N_i)$. Now we can define a family of equivalence relations $\mathbb{R}^\ell$ so that for any ℓ with $1 < k \leq \ell$. We define the following equivalence relation $\mathcal{R}_{\mathsf{TDH}} \in \mathbb{R}^\ell$ and the equivalence class $[(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ of a message vector with size k. More concretely, for a fixed bilinear group BG and (k, ℓ), we define $\mathcal{R}_{\mathsf{TDH}} \in \mathbb{R}^\ell$ as follows:

Definition 30 (Equivalence relations of $\mathcal{M}^H_{\mathsf{TDH}}$ message spaces)**.** *If vectors of a pair $(\mathbf{M}, \mathbf{N}) \in (\mathbb{G}_1^*)^k \times (\mathbb{G}_2^*)^k$ is a message vector from $\mathcal{M}^H_{\mathsf{TDH}}$, then the equivalence relations $[(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ defined as*

$$\mathcal{R}_{\mathsf{TDH}} = \left\{ \begin{array}{l} (\mathbf{M}, \mathbf{N}), (\mathbf{M}', \mathbf{N}') \in (\mathbb{G}_1^* \times \mathbb{G}_2^*)^k \times (\mathbb{G}_1^* \times \mathbb{G}_2^*)^k \Leftrightarrow \exists (\mu, \upsilon) \in \mathbb{Z}_p^* : \\ \mathbf{M}' = \mathbf{M}^{\mu\upsilon}, \mathbf{N}' = \mathbf{N}^{\upsilon} \end{array} \right\}$$

Note that the EQ relation for an aggregate signature on a set of vectors $\mathbb{M} = ((\mathbf{M}_j, \mathbf{N}_j))_{j\in[\ell]}$ is the family (set) of relation as above, while all vectors use the same randomness $\mathbb{M} = ((\mathbf{M}_j^{\mu\upsilon}, \mathbf{N}_j^{\upsilon}))_{j\in[\ell]}$. For instance, the j'th message vector $(\mathbf{M}_j, \mathbf{N}_j) \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}^j_{\mathsf{TDH}}}$ is in the class $\mathcal{R}^j_{\mathsf{TDH}} \in \mathbb{R}^\ell$ and if one more signature-message pair is added to the set, we have $\mathcal{R}^{j+1}_{\mathsf{TDH}} \in \mathbb{R}^\ell$, where $j + 1 < \ell$. Moreover, we consider the EQ relation for verification keys

vk and Tag similar to AtoSa and indicate as $\mathcal{R}_{\mathsf{vk}}$ (see Def. 2.4.1.5) and $\mathcal{R}_\tau$ as stated in Def. 3.2.1. We again denote by $\mathcal{T}$ the space of all tags. We present our ATMS scheme in Definition 31.

Definition 31 (Aggregate Mercurial Signatures with Randomizable Tag (ATMS))**.** *An* ATMS *scheme, associated with the parameterized equivalence relations* $\mathbb{R}^\ell$*,* $\mathcal{R}_{\mathsf{TDH}}$*,* $\mathcal{R}_\tau$ *and* $\mathcal{R}_{\mathsf{vk}}$*, and also message space* $\mathcal{M}^H_{\mathsf{TDH}}$ *consists of the algorithms:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter λ, output the public parameters pp.

$\mathsf{KeyGen}\,(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{vk})$: On input the public parameters pp, output a key pair $(\mathsf{sk}, \mathsf{vk})$.

$\mathsf{VKeyGen}\,(\mathsf{sk})$: On input a secret key sk, output a verification key vk.

$\mathsf{GenIdxTag}(S) \to (\mathsf{aux}_j, (\tau, \mathbf{T}))$: Given a set $S = ((\mathbf{M}_j, \mathbf{N}_j)\,, \mathsf{vk}_j)_{j\in[n]}$ of messages and keys, output auxiliary data aux_j and a tag pair $(\tau, \mathbf{T})$ where τ is the secret part and $\mathbf{T}$ is the public part of tag and all vk_j should be distinct.

$\mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, (\mathbf{M}_j, \mathbf{N}_j)) \to \sigma_j$: On input a secret key sk_j, tag's secret τ, auxiliary data aux_j and message vector $(\mathbf{M}_j, \mathbf{N}_j) \in \mathcal{M}^H_{\mathsf{TDH}}$, output a signature σ_j under the τ, vk_j and $(\mathbf{M}_j, \mathbf{N}_j)$.

$\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, (\mathbf{M}_j, \mathbf{N}_j), \sigma_j) \to \{0, 1\}$: Given a verification key vk_j, tag's public $\mathbf{T}$, message vector $(\mathbf{M}_j, \mathbf{N}_j)$ and signature σ_j, output 1 if σ_j is valid relative to vk_j, $(\mathbf{M}_j, \mathbf{N}_j)$ and $\mathbf{T}$, and 0 otherwise.

$\mathsf{VerifyTag}(\mathbf{T}, \tau, \sigma) \to \{0, 1\}$: Given a tag's public $\mathbf{T}$, tag's secret signature σ, output 1 if $\mathbf{T}$ is valid relative to σ, and τ, and 0 otherwise.

$\mathsf{AggrSign}(\mathbf{T}, (\mathsf{vk}_j, (\mathbf{M}_j, \mathbf{N}_j), \sigma_j)_{j=1}^\ell) \to \sigma'$ Given ℓ signed messages $(\mathbf{M}_j, \mathbf{N}_j)$ in σ_j under vk_j for $j \in [\ell]$ and the same tag $\mathbf{T}$, output a signature σ on the messages $\mathbb{M} = ((\mathbf{M}_j, \mathbf{N}_j))_{j\in[\ell]}$ under the tag $\mathbf{T}$ and verification key $\mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}$.

$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma) \to \{0, 1\}$: Given a verification key avk, tag $\mathbf{T}$, messages $\mathbb{M}$ and signature σ, output 1 if σ is valid relative to avk, $\mathbb{M}$ and $\mathbf{T}$, and 0 otherwise.

$\mathsf{ConvertTag}(\mathbf{T}, \mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a randomized tag $\mathbf{T}' \in [\mathbf{T}]_{\mathcal{R}_\tau}$ (i.e., a new representative of tag).

$\mathsf{ChangRep}((\mathbf{M}, \mathbf{N}), \sigma, \mathbf{T}, (\mu, \upsilon)) \to (\sigma', \mathbf{T}')$: On input a representative $(\mathbf{M}, \mathbf{N}) \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$, $\mathbf{T} \in [\mathbf{T}]_{\mathcal{R}_\tau}$, signature σ and randomness (μ, υ), return a new signature $((\mathbf{M}', \mathbf{N}'), \mathbf{T}', \sigma')$, where $\mathbf{M}' = \mathbf{M}^{\mu\upsilon} \wedge \mathbf{N}' = \mathbf{N}^\upsilon \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ and $\mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)$ are the new representatives and σ' is valid for $(\mathbf{M}', \mathbf{N}')$ and $[\mathbf{T}]_{\mathcal{R}_\tau}$.

This will also apply for a set representative $\mathbb{M}$ such that one can get a new set representative $\mathbb{M}'$ by scaling all message with the same (μ, υ).

$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \to \mathsf{sk}'$: On input a sk and key converter ω, output a new secret key sk'.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: On input a $\mathsf{vk}$and key converter ω, output a new public key vk'.

$\mathsf{ConvertSig}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, \omega) \to \sigma'$: On input a vk, message vector $(\mathbf{M}, \mathbf{N})$, signature with tag $(\sigma, \mathbf{T})$, and key converter ω, return a new signature σ' such that $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma') = 1$, where $\mathsf{vk}' \leftarrow \mathsf{ConvertVK}(\mathsf{vk}, \omega)$.

The VerifyTag and VKeyGen are only used for the security game.

3.3.2 Security Definitions

Correctness. As usual we require that honest signatures verify as expected, but need to consider all the randomizations as well as the aggregation.

Definition 32 (ATMS correctness)**.** *For all:* S,λ,$(\mathbf{M}, \mathbf{N}) \in \mathcal{M}^H_{\mathsf{TDH}}$ *such that:*

$$\begin{aligned}
&\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)\\
&\mathsf{aux}, \mathbf{T}, \tau := \mathsf{GenAuxTag}(S)\\
&(\sigma_i)_{i=1}^{\ell} = (\mathsf{Sign}(\mathsf{sk}_i, \tau, \mathsf{aux}, (\mathbf{M}_i, \mathbf{N}_i)))_{i=1}^{\ell},\\
&\sigma' := \mathsf{AggrSign}(\mathbf{T}, (\mathsf{vk}_j, (\mathbf{M}_j, \mathbf{N}_j), \sigma_j)_{j=1}^{\ell})
\end{aligned}$$

Then:

$$\begin{aligned}
&\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i') = 1.\\
&\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}(\mathbf{M}_i, \mathbf{N}_i)_{i=1}^{\ell}, \sigma')
\end{aligned}$$

Further, for randomization, if $\forall$ τ *secret part of the tag,* $(m_1^{(1)}, m_2^{(1)})_{i=1}^{\ell} \in \mathbb{Z}_p^{2\ell}$, *keys honestly generated* $(\mathsf{sk}_i, \mathsf{vk}_i)_{i=1}^{\ell}$ *and for all randomness* (γ, β, ω), *for all* $\sigma, \{\sigma_i\}_{i\in[\ell]}, \mathbf{T}$, *if*

$$\begin{aligned}
&\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i) = 1.\\
&\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i)_{i=1}^{\ell}, \sigma)\\
&(\sigma_i', \mathbf{T}')_{i=1}^{\ell} = (\mathsf{ChangRep}(\mathsf{vk}_i, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i), (\gamma, \beta)))_{i=1}^{\ell},\\
&(\mathsf{vk}_i')_{i=1}^{\ell} := (\mathsf{ConvertVK}(\mathsf{vk}_i, \omega))_{i=1}^{\ell}\\
&(\sigma_i'')_{i=1}^{\ell} := (\mathsf{ConvertSig}(\mathsf{vk}_i, (\mathbf{M}_i, \mathbf{N}_i), \mathbf{T}', \sigma_i', \omega))_{i=1}^{\ell}\\
&(\mathbf{T}_i^{\dagger}) := \mathsf{ConvertTag}(\mathbf{T}, \omega)\\
&\sigma^* := \mathsf{AggrSign}(\mathbf{T}', (\mathsf{vk}_j, (\mathbf{M}_j, \mathbf{N}_j), \sigma_j'')_{j=1}^{\ell})
\end{aligned}$$

then

$$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}', (\mathbf{M}_i, \mathbf{N}_i)_{i=1}^{\ell}, \sigma^*)$$
$$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}^\dagger, (\mathbf{M}_i, \mathbf{N}_i)_{i=1}^{\ell}, \sigma^*)$$
$$\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}'_i, \mathbf{T}', (\mathbf{M}_i, \mathbf{N}_i), \sigma'_i) = 1.$$
$$\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}^\dagger, (\mathbf{M}_i, \mathbf{N}_i), \sigma''_i) = 1.$$

Also $\forall(\mathsf{sk}, \mathsf{vk})$ *honestly generated* $\mathbf{T}$ *and* $(\mathbf{M}, \mathbf{N})$ *then* $\mathsf{ChangRep}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, \cdot)$ *is a group morphism from* $\mathbb{Z}_p^{*2}$ *to* $\mathcal{SIG}$*, moreover, we have*

$$\omega \mapsto \begin{pmatrix} \mathsf{ConvertSK}(\mathsf{sk}, \omega), \mathsf{ConvertVK}(\mathsf{vk}, \omega), \\ \mathsf{ConvertSig}(\mathsf{vk}, (\mathbf{M}, \mathbf{N}), \mathbf{T}, \sigma'_i, \omega) \end{pmatrix}$$

is a group morphism from $\mathbb{Z}_p^{*2}$ *to* $\mathcal{SK} \times \mathcal{VK} \times \mathcal{SIG}$

Unforgeability. The unforgeability game follows the unforgeability definition of AtoSa (see Def. 23). It is slightly modified to fit with our additional EQ relation (Def. 30), i.e., unforgeability is defined with respect to message classes and in addition need to check VerifyTag.

Definition 33 (Unforgeability)**.** *An* ATMS *is unforgeable if for all PPT* $\mathcal{A}$ *having access to the oracle* $\mathcal{O}^{\mathsf{Sign}()}$ *there exists a negligible function* ϵ *s.t:* $\Pr[\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda) = 1] \leq \epsilon(\lambda)$ *where the experiment* $\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda)$ *is defined in Fig. 3.3 and* Q *is the set of queries that* $\mathcal{A}$ *has issued to* $\mathcal{O}^{\mathsf{Sign}()}$*.*

$\underline{\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda)}$:

- $Q := \emptyset; \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
- $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(\mathsf{pp})$;
- $(j', \mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}, \mathsf{ask} = (\mathsf{sk}_j)_{j\in[\ell]\setminus j'}, \mathbb{M}^* = ((\mathbf{M}_j^*, \mathbf{N}_j^*))_{j\in[\ell]}, \mathbf{T}^*, \tau^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{vk}')$
- $(\mathsf{vk}_j^* := \mathsf{VKeyGen}(\mathsf{sk}_j))_{j\in[\ell], j\neq j'}$ **Return:**

$$\begin{pmatrix} \mathsf{VerifyAggr}\,(\mathsf{avk}, \mathbf{T}^*, \sigma^*, \mathbb{M}^*) = 1 \ \wedge \mathsf{VerifyTag}(\mathbf{T}^*, \sigma^*, \tau^*) \wedge \forall j \in [\ell], j \neq j' : \\ [\mathsf{vk}_j^*]_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_j]_{\mathcal{R}_{\mathsf{vk}}} \wedge [\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_{j'}]_{\mathcal{R}_{\mathsf{vk}}} \wedge \\ \forall((\mathbf{M}, \mathbf{N}), \mathbf{T}) \in Q : [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}} \neq [(\mathbf{M}_j^*, \mathbf{N}_j^*)]_{\mathcal{R}_{\mathsf{TDH}}} \vee [\mathbf{T}]_{\mathcal{R}_\tau} \neq [\mathbf{T}^*]_{\mathcal{R}_\tau} \end{pmatrix}$$

$\underline{\mathcal{O}^{\mathsf{Sign}}((\tau, \mathbf{T}), \mathsf{aux}, (\mathbf{M}, \mathbf{N}))}$:

- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}', \tau, \mathsf{aux}, (\mathbf{M}, \mathbf{N}))$
- $Q = Q \cup \{(\mathbf{M}, \mathbf{N}), \mathbf{T}\}$, **Return** σ

Figure 3.3: Experiment $\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda)$

Privacy guarantees. Similar as in Section 3.2, we consider the privacy notations *Origin-hiding of ConvertSig*, and *Public key class-hiding* (it is the same as Def. 24). We note that all definitions can be updated due to $\mathcal{M}^H_{\mathsf{TDH}}$ message space (receptively EQ relations of $\mathcal{M}^H_{\mathsf{TDH}}$) instead of the vector $\mathbf{M}$. Origin-hiding of ConvertSig definition can be updated straightforwardly as follows:

Definition 34 (Origin-hiding of ConvertSig for ATMS)**.** *For all* λ*, and* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$*, for all* $(\mathsf{vk}, (\mathbf{M}, \mathbf{N}), \sigma, \mathbf{T})$*, if* $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma) = 1$*, and* $(\omega, \upsilon, \mu) \in \mathbb{Z}_p^*$*, then* $\sigma' \leftarrow \mathsf{ChangRep}((\mathbf{M}, \mathbf{N}), \mathsf{ConvertSig}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, \omega), \mathbf{T}, (\upsilon, \mu))$ *outputs a uniformly random in the respective spaces s.t.* $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}', (\mathbf{M}', \mathbf{N}'), \sigma') = 1$*, where* $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ *outputs a uniformly random element of* $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$.

However, since this is a variant of SPSEQ we consider the *adaption property* similar to [FHS19] below, an additional property which guarantees that signatures from ChangRep and Sign are identically distributed. This definition also covers Origin-hiding of ConvertTag.

Definition 35 (Perfect Adaption of Signatures)**.** *An* ATMS *scheme perfectly adapts signatures if for all* $(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, (\mu, \upsilon))$ *with* $(\mathbf{M}, \mathbf{N}) \in \mathcal{M}^H_{\mathsf{TDH}} \wedge \mathsf{Verify}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma) = 1 \wedge \ (\mu, \upsilon) \in \mathbb{Z}_p^*$ *we have that the output of* $(\sigma', \mathbf{T}') \leftarrow \mathsf{ChangRep}(\sigma, (\mathbf{M}, \mathbf{N}), \mathbf{T}, (\mu, \upsilon))$ *is a uniformly random element in the respective space, conditioned on* $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}^\mu, (\mathbf{M}^{\mu\upsilon}, \mathbf{N}^\upsilon), \sigma') = 1$.

3.3.3 Construction

Our construction is inspired by the message-indexed SPS by Crites et al. [CKP+22] (see Def. 2.4.1.3), which is a variant of Ghadafi's SPS [Gha16] (see Def. 2.4.1.2). We use the tag-based message definition $\mathcal{M}^H_{\mathsf{TDH}}$ (Def. 29) instead of the message-indexed (Def. 9). For simplicity, we assume a message vector with the length $k = 2$ as $(\mathbf{M}, \mathbf{N}) = ((M_1, M_2), (N_1, N_2))$, but this can be straightforwardly generalized to any length $k > 1$. Similar to the construction in Section 3.2.3, we again need aux binding to make this particular construction work.

Definition 36 (Aux binding for ATMS)**.** *We split* aux *into a preimage and an opening:* (c, o)*. For all PPT* $\mathcal{A}$*, and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda)$ *there exists a negligible* ϵ *such that:*

$$\Pr\left[\begin{array}{l}(\mathsf{aux} = (c, o), \mathsf{aux} = (c', o'), \tau, (\mathbf{M}, \mathbf{N}), \tau', (\mathbf{M}', \mathbf{N}')) \leftarrow \mathcal{A}(\mathsf{vk}); \\ \mathsf{VerifyAux}(\mathsf{sk}, (c, o), \tau, (\mathbf{M}, \mathbf{N})) = 1 \wedge \mathsf{VerifyAux}(\mathsf{sk}, (c', o'), \tau', (\mathbf{M}', \mathbf{N}')) = 1 \wedge \\ c = c' \wedge (\tau \neq \tau' \vee (\mathbf{M}, \mathbf{N}) \neq (\mathbf{M}', \mathbf{N}'))\end{array}\right] \leq \epsilon(\lambda)$$

Synchronicity assumption. Same as in Section 3.2.3, instead of fixing messages and verification keys in aux, we can make same assumption as in synchronized aggregate signatures and simply set $c = P^{\rho_1} || P^{\rho_2}$ in the construction below and Definition 28 is trivially satisfied.

Our construction. The construction is as follows:

$\mathsf{Setup}(1^\lambda)$: Run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$ with a prime number order p, where P a generator of $\mathbb{G}_1$, $\hat{P}$ a generator of $\mathbb{G}_2$ and H a hash function: $H : \{0,1\}^* \rightarrow \mathbb{G}_1$, output $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, H)$.

$\mathsf{KeyGen}(\mathsf{pp})$: Given pp, sample $\mathsf{sk} = (x, y_1, y_2, z_1, z_2) \xleftarrow{\$} (\mathbb{Z}_p^*)^5$, and $\mathsf{vk} = (\hat{X} = \hat{P}^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{Z}_1 = \hat{P}^{z_1}, \hat{Z}_2 = \hat{P}^{z_2})$.

$\mathsf{VKeyGen}(\mathsf{sk})$: Given $\mathsf{sk} = (x, y_1, y_2, z_1, z_2)$, return $\mathsf{vk} = (\hat{X} = \hat{P}^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{Z}_1 = \hat{P}^{z_1}, \hat{Z}_2 = \hat{P}^{z_2})$.

$\mathsf{GenIdxTag}(S)$: Given a set $S = \{(\mathbf{M}_j, \mathbf{N}_j, \mathsf{vk}_j)_{j\in[n]}\}$, choose $(\rho_1, \rho_2) \xleftarrow{\$} \mathbb{Z}_p$, set $\tau = (\rho_1, \rho_2), \mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$, and $c = \left(P^{\rho_1} || P^{\rho_2} || (\mathbf{N}_j, \mathsf{vk}_j)_{j\in[n]}\right)$, where $h = H(c)$ and $\mathsf{aux}_j = (c, o = \perp)$.

$\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, (\tau_1, \tau_2), ((M_1, M_2), (N_1, N_2)))$: Extract (T_1, T_2), parse aux as (c, o). Check that $((\mathbf{M}, \mathbf{N}), [\mathsf{VKeyGen}(\mathsf{sk})]) \in \mathsf{aux}$ (i.e., $c = ...||((\mathbf{M}, \mathbf{N}), [\mathsf{VKeyGen}(\mathsf{sk})])||...$) s.t no other vk in aux related to sk and check that $(T_1, T_2) = (h^{\tau_1}, h^{\tau_2})$. Compute $h := H(c)$. Output $\bigwedge_{i=1}^{2} e(M_i, \hat{P}) = e(h^{\tau_i}, N_i)$.

$\mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, (\mathbf{M}, \mathbf{N}))$: Given a sk_j, $\tau, \mathsf{aux}_j = (c, \perp)$, and message $(\mathbf{M}, \mathbf{N}) = ((M_1, M_2), (N_1, N_2)) \in \mathcal{M}^H_{\mathsf{TDH}}$. Parse τ as (ρ_1, ρ_2). Run $\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, (\mathbf{M}, \mathbf{N}))$ and verify that this outputs 1. If so compute $h = H(c)$ and output a signature as:

$$\sigma = (h, b = \prod_{j\in[2]} h^{\rho_j \cdot z_j}, s = (h^x \cdot \prod_{j\in[2]} M_j^{y_j})).$$

$\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma)$: Given a vk, tag $\mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$, message $(\mathbf{M}, \mathbf{N})$ and signature $\sigma = (h, b, s)$ return 1 if the following holds and 0 otherwise:

$$e(h, \hat{X}) \prod_{j\in[2]} e(M_j, \hat{Y}_j) = e(s, \hat{P}) \ \wedge \ e(b, \hat{P}) = \prod_{j\in[2]} e(T_j, \hat{Z}_j)$$
$$\bigwedge_{j=1}^{2} e(T_j, N_j) = e(M_j, \hat{P})$$

$\mathsf{VerifyTag}(\mathbf{T}, \tau, \sigma)$: Given $\tau = (\tau_1, \tau_2)$, $\sigma = (h, b, s)$, output 1 if $T_i = h^{\tau_i}$ for all $i \in \{1, 2\}$, and 0 otherwise.

$\mathsf{AggrSign}(\mathbf{T}, (\mathsf{vk}_i, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i)_{i=1}^{\ell})$: Given ℓ valid signatures $\sigma_i = (h, b_i, s_i)$ for $(\mathbf{M}_i, \mathbf{N}_i)$ under vk_i and the same tag $\mathbf{T}$ for $i \in [\ell]$, return $\perp$ if all h are not the same, else output a signature σ on the messages $\mathbb{M} = ((\mathbf{M}_i, \mathbf{N}_i))_{i\in[\ell]}$ under the tag $\mathbf{T}$ and aggregated verification key $\mathsf{avk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$ as follows: $\sigma = \left(h, b' = \prod_{i=1}^{\ell} b_i, s' = \prod_{i=1}^{\ell} s_i\right)$.

$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma)$: Given $\mathsf{avk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$, tag $\mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$, messages $\mathbb{M}$ and signature $\sigma = (h, b, s)$, check if the following checks holds and 0 otherwise:

$$\prod_{i\in[\ell]} e(h, \hat{X}_i) \prod_{j\in[2]} e(M_{ij}, \hat{Y}_{ij}) = e(s, \hat{P}) \ \wedge \ e(b, \hat{P}) = \prod_{i\in[\ell]} \prod_{j\in[2]} e(T_j, \hat{Z}_{ij})$$
$$\bigwedge_{j\in[2]\wedge i\in[\ell]} e(T_j, N_{ij}) = e(M_{ij}, \hat{P})$$

$\mathsf{ConvertTag}(\mathbf{T}, \mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a randomized tag $\mathbf{T}' = (h^{\rho_1\mu}, h^{\rho_2\mu})$.

$\mathsf{ChangRep}(\sigma, (\mathbf{M}, \mathbf{N}), \mathbf{T}, (\mu, \upsilon))$: On input a representative $(\mathbf{M}, \mathbf{N}) \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$, $\mathbf{T} \in [\mathbf{T}]_{\mathcal{R}_\tau}$, signature $\sigma = (h, b, s)$, and $(\mu, \upsilon) \in (\mathbb{Z}_p^*)^2$, output:

$$\sigma' = \left(h' \leftarrow h^{\mu\upsilon}, b' \leftarrow b^{\mu}, s' \leftarrow s^{\mu\upsilon}, \mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)\right),$$

which is a valid signature for new representatives $(\mathbf{M}^{\mu\upsilon} = \mathbf{M}', \mathbf{N}^{\upsilon} = \mathbf{N}') \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ and $\mathbf{T}' = (h^{\rho_1\mu}, h^{\rho_2\mu}) \in [\mathbf{T}]_{\mathcal{R}_\tau}$.

$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \to \mathsf{sk}'$: On input a sk and key converter $\omega \in \mathbb{Z}_p^*$, output a new secret key as $\mathsf{sk}' = \mathsf{sk} \cdot \omega$.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: On input a vkand key converter $\omega \in \mathbb{Z}_p^*$, output $\mathsf{vk}' = \mathsf{vk}^{\omega} = (\hat{X}^{\omega}, \hat{Y}_1^{\omega}, \hat{Y}_2^{\omega}, \hat{Z}_1^{\omega}, \hat{Z}_2^{\omega})$.

$\mathsf{ConvertSig}(\mathsf{vk}, (\mathbf{M}, \mathbf{N}), \sigma, \mathbf{T}, \omega) \to \sigma'$: On input a vk, message $(\mathbf{M}, \mathbf{N})$, signature σ with tag $\mathbf{T}$, and key converter $\omega \in \mathbb{Z}_p^*$, returns a new signature σ' as: $\sigma' = (h, b^{\omega}, s^{\omega})$.

Note that one can reduce the number of paring operations in $\mathsf{VerifyAggr}$ by using batching verification techniques (cf. [FGHP09]).

Theorem 3.3.1 (Privacy)**.** *Our construction is origin-hiding of ConvertSig (Def. 26), public key class-hiding (Def. 24), and provides perfect adaption of signatures (Def. 35).*

We refer to [MBG+23] for the proof of Theorem 3.3.1.

Theorem 3.3.2 (Unforgeability)**.** *Our construction is EUF-CMA secure regarding the definition 33 in the generic group model for Type-III bilinear groups.*

The proof of Theorem 3.3.2 is provided in our paper [MBG+23].

3.4 Application to AC

As our core application we present Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA). In a multi-authority setting [HP22], credentials come from ℓ different credential issuers. Naively, the showing of credentials requires ℓ independent credentials to be shown. This can be overcome [HP22] by leveraging aggregate signatures, obtaining a compact AC system with compact-size credentials, and showing costs. However, verifying a user's credentials needs knowledge of all issuers' verification keys, which might violate user privacy. Thus, in the vein of [BEK+21] we introduce the issuer-hiding property for multi-authority credentials. We recall that here the verifier can define a set of acceptable issuers in an ad-hoc manner. Then a user can prove that the subset of credentials shown were issued by acceptable issuers without revealing which credential corresponds to which issuer. This is an important feature, especially in multi-authority settings where disclosing issuer keys can reveal too much information compared to a single issuer setting and already lead to identification of the user.

3.4.1 Formal Definition

Our definition supports multiple users $(u_j)_{j\in[\ell]}$ and multiple credential issuers $(\mathsf{CI}_j)_{j\in[\ell]}$. An issuer can generate a key pair of secret and verification keys $(i\mathsf{sk}, i\mathsf{vk})$ via $\mathsf{IKeyGen}()$. Similarly, users runs $\mathsf{UKeyGen}()$ to generate a user key pair $(u\mathsf{sk}, u\mathsf{vk})$. Each issuer can then issue a credential (cred) on an attribute (a) or attribute-set (A) to a user who can verify the received credential locally. Indeed, when we use AtoSa, we consider an attribute a (i.e., the attribute set includes only one attribute); when we use ATMS, we consider an attribute set, A. We use the notation A, to define security and formal definitions for consistency of definitions.

Users can then use the $\mathsf{CredAggr}$ algorithm to aggregate all credentials and create a single credential valid for all attributes and verification keys. To define the set of accepted issuers, a verifier generates a key-policy pol using $\mathsf{GenPolicies}$ (it is known as Presentation policies in [BEK+21]), which can be checked for well-formedness by everyone. Finally, with an aggregate credential (disclosing a subset attributes D) and some key-policy pol from the verifier, a user uses Show to derive a proof, which a verifier can verify.

Definition 37 (Issuer-Hiding Multi-Authority Credentials (IhMA))**.** *An* IhMA *is defined by the following algorithms/protocols:*

- Setup: On input a security parameter λ, output public parameters pp (implicit input to all algorithms) .
- $\mathsf{IKeyGen}$: Generate a key pair $(i\mathsf{sk}, i\mathsf{vk})$ for an issuer i.
- $\mathsf{UKeyGen}$: Take a message-key set S, generate a user key pair $(u\mathsf{sk}, u\mathsf{vk})$ which acts as user's identity and auxiliary data aux.

- Issuance: In this protocol, an issuer i associated to $(i\mathsf{sk}, i\mathsf{vk})$ creates a credential cred on an attributes-set A to a user u associated to $(u\mathsf{sk}, u\mathsf{vk})$ as follows:

$$[\mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}) \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A})] \rightarrow \mathsf{cred}$$

- CredAggr: Take as input a $u\mathsf{sk}$ of user and a list of credentials $(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_i)$ for $i \in [\ell]$ and output an aggregated credential cred of attributes-set $\{\mathsf{A}_i\}_{i\in[\ell]}$:

$$\mathsf{CredAggr}\left(u\mathsf{sk}, \{(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_i)\}_{i\in[\ell]}\right) \rightarrow \mathsf{cred}$$

- GenPolicies: A verifier with the secret key $v\mathsf{sk}$ can define policies defining sets of issuers $\{\mathsf{ivk}\}_{i\in[n]}$ they are willing to accept for certain Show sessions, we have:

$$\mathsf{GenPolicy}(v\mathsf{sk}, \{\mathsf{ivk}\}_{i\in[n]}) \rightarrow pol, \text{ where } n \leq \ell$$

- Show: In this protocol, a user u with $(u\mathsf{sk}, u\mathsf{vk})$ runs CredShow and interacts with a verifier running CredVerify to prove that she owns a valid credential cred on disclosed attribute sets $D \subseteq \{\mathsf{A}_i\}_{i\in[\ell]}$ issued respectively by one or some credential issuers in pol:

$$\begin{bmatrix}\mathsf{CredShow}(u\mathsf{sk}, pol, \{(i\mathsf{vk}, \mathsf{A}_i)\}_{i\in[\ell]}, \mathsf{cred}, D) \leftrightarrow \\ \mathsf{CredVerify}(pol, (i\mathsf{vk}_i)_{i\in[\ell]}, D)\end{bmatrix} \rightarrow (0, 1)$$

3.4.2 Security Definitions

We define our security model based on the game-based framework in [FHS19, HP22], with some modifications to harmonize their definition with the one on IhMA and consider the use of multi-authority and issuer-hiding properties. The adversary $\mathcal{A}$ has access to the following oracles that describe the possible ways to interact with the IhMA. Moreover, we define some global lists that are shared among oracles as $\mathcal{HU}$ a list of honest users and $\mathcal{CU}$ a list of corrupted users, similarly we define $\mathcal{HCI}$ and $\mathcal{CCI}$ for the honest/corrupted credential issuers. Also, $\mathcal{L}_{uk}$ stands for a list of user's keys and $\mathcal{L}_{\mathsf{cred}}$ which is a list of user-credential pairs which includes issued credentials and attributes, and to which users they were issued. A credential in $\mathcal{L}_{\mathsf{cred}}$ can be empty ($\perp$) if the user has not received a credential on this attribute yet. For simplicity we assume a tag τ includes aux as well.

- $\mathcal{O}^{\mathsf{HCI}}(i)$: Create an honest issuer with identity i. If i already exists (i.e. $i \in \mathcal{HCI} \cup \mathcal{CCI}$), output $\perp$. Otherwise, run $(i\mathsf{sk}, i\mathsf{vk}) \leftarrow \mathsf{IKeyGen}(i)$, add $(i, i\mathsf{sk}, i\mathsf{vk}) \in \mathcal{HCI}$, and return $i\mathsf{vk}$.

- $\mathcal{O}^{\mathsf{CorruptCI}}(i)$: Corrupt a credential issuer i. If i does not exist (i.e. $i \notin \mathcal{HCI} \cup \mathcal{CCI}$), create a new corrupted issuer by appending i to $\mathcal{CCI}$. Otherwise, if $i \in \mathcal{HCI}$, remove i from $\mathcal{HCI}$, add it to $\mathcal{CCI}$ and output $i\mathsf{sk}$. Note that $\mathcal{A}$ does not allow to corrupt the challenge key vk'.

- $\mathcal{O}^{\mathsf{User}}(u, S)$: On input a user identity u and issuer/attributes pairs $\{(a_i, \mathsf{vk}_i)\} \in S$. If $u \in \mathcal{HU}$ or $u \in \mathcal{CU}$, return $\perp$. Else, create a fresh entry u by running $(u\mathsf{sk}, u\mathsf{vk}, \mathsf{aux}) \leftarrow \mathsf{UKeyGen}$ and adding u and $(u\mathsf{sk}, u\mathsf{vk}, \mathsf{aux})$ to the list $\mathcal{HU}$ and $\mathcal{L}_{uk}$, receptively. Then, for each $(a_i, \mathsf{vk}_i) \in S$, add $\mathcal{L}_S[i]$. Return $u\mathsf{vk}$.

- $\mathcal{O}^{\mathsf{CorruptU}}(u)$: On input a user identity u and a user public key $u\mathsf{vk}$. If $u \notin \mathcal{HU}$, register a new corrupt user with $u\mathsf{vk}$ and add $u \in \mathcal{CU}$. Else, move the entry corresponding to u from $\mathcal{HU}$ and add it to $\mathcal{CU}$, output $u\mathsf{sk}$ and all the related credentials items $(u, \mathsf{A}_i, \mathsf{cred}_i)$ of $\mathcal{L}_{\mathsf{cred}}[u]$.

- $\mathcal{O}^{\mathsf{ObtIss}}(u, i, \mathsf{A}_i)$: (Perform an honest issuing/obtaining) Take as input a user identity u, issuer identity i, and attribute(s) A_i. If $u \notin \mathcal{HU} \vee i \notin \mathcal{HCI}$, return $\perp$. Else, find entries $(u\mathsf{sk} \in \mathcal{L}_{uk}, i\mathsf{sk} \in \mathcal{HCI})$, and run the issuing protocols:

  $$[\mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow \mathsf{C}redIssue(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i)] \rightarrow \mathsf{cred}_i$$

  and add the entry $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$, where cred_i includes aux_i.

- $\mathcal{O}^{\mathsf{Obtain}}(u, i, \mathsf{A}_i)$: (Perform an honest obtaining of a credential with a malicious issuer) On input a user identity $u \in \mathcal{HU}$, issuer identity $i \in \mathcal{CCI}$ and attributes A_i. If $u \notin \mathcal{HU} \vee i \notin \mathcal{CCI}$, return $\perp$. Else, find $u\mathsf{sk} \in \mathcal{L}_{uk}$, and run the Obtain protocol with $\mathcal{A}$:

  $$[\mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow \mathcal{A}] \rightarrow \mathsf{cred}_i$$

  If $\mathsf{cred}_i = \perp$, return $\perp$. Else, append the resulting output $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$.

- $\mathcal{O}^{\mathsf{Issue}}(u, i, \mathsf{A}_i)$: (Perform a malicious obtaining of a credential with an honest issuer) On input a user identity $u \in \mathcal{CU}$, issuer $i \in \mathcal{HCI}$, and attributes A_i. If $u \notin \mathcal{CU} \vee i \notin \mathcal{HCI}$, return $\perp$. Else, find entries $i\mathsf{sk} \in \mathcal{HCI}$, and run Issuing with $\mathcal{A}$:

  $$[\mathcal{A} \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i)] \rightarrow \mathsf{cred}_i$$

  Append elements $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by a corrupted user u to get a credential from a honest issuer.

- $\mathcal{O}^{\mathsf{CredShow}}(j, pol, D)$: On input an index of an issuance j, key-policy pol and attributes-subset D. First parses $\mathcal{L}_{\mathsf{cred}}[j]$ as $(u, \mathsf{A}_i, \mathsf{cred}_i)$, where cred_i is the credential issued by an issuer $i\mathsf{vk}$ on A_i for a user u during the i-th query to $\mathcal{O}^{\mathsf{ObtIss}}$ or $\mathcal{O}^{\mathsf{Obtain}}$. If $i \notin \mathcal{HU}$ return $\perp$. Else, run (with the adversary):

  $\mathsf{CredShow}(u\mathsf{sk}, pol, \{(i\mathsf{vk}, \mathsf{A}_i)\}_{i \in [\ell]}, \mathsf{cred}, D) \leftrightarrow \mathcal{A}$.

($\mathcal{O}^{\mathsf{Obtain}}$) and ($\mathcal{O}^{\mathsf{Issue}}$) are defined specifically for the anonymity ExpAno and the unforgeability ExpUnf, respectively. The other oracles are common between ExpAno and ExpUnf.

Correctness We require that honestly issued credentials shown to honest verifiers always verify with a caveat. If a user does not specify a particular issuer and attribute when User

is called, then if that issuer is called to issue that attribute to the user, it is allowed to fail. I.e.: the user must include pairs for every attribute they wish to receive. Further, if the user specifies two attributes for the same issuer, we allow the scheme to return $\perp$ during issuing. This limitation can be overcome practically by having each issuer use a different key for each attribute.

Unforgeability. Unforgeability requires that no adversary can convince a verifier to accept a credential for a set of attributes for which he does not possess all the individual credentials (and related users' secret keys) from the accepted issuers set $I = \{i\mathsf{vk}\}_{i\in[\ell]}$. $\mathcal{A}$ can obtain $i\mathsf{vk} \in I$ using $\mathcal{O}^{\mathsf{HCI}}(i)$ and $\mathcal{O}^{\mathsf{CorruptCI}}(i)$. Intuitively, an adversary can win the unforgeability experiment if $\mathcal{A}$ is able to convince an honest verifier that he satisfies a certain attribute subset while he does not have an appropriate credential. To make the game non-trivial, we impose restrictions that for all corrupted users the disclosed attributes subset D should not pass verification (satisfy attributes credentials).

Definition 38 (Unforgeability)**.** *An* IhMA *is unforgeable if, for all* $\lambda \in \mathbb{N}$ *and for any* PPT *adversary* $\mathcal{A}$*, there exists* $\epsilon(\lambda)$ *s.t* $\Pr[\mathsf{ExpUnf}_{\mathsf{IhMA},\mathcal{A}}(\lambda) = 1] \leq \epsilon(\lambda)$*, experiments are defined in Fig 3.4, where* A_{ui} *means the (set) attribute issued by the issuer* $i\mathsf{vk}$ *to* u*.*

$\mathsf{ExpUnf}_{\mathsf{IhMA},\mathcal{A}}(\lambda)$:

- $(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$; $I \leftarrow \mathcal{A}^{<\mathcal{O}^{\mathsf{CorruptCI}},\mathcal{O}^{\mathsf{HCI}}>}(\mathsf{pp})$;
- $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{IKeyGen}(\mathsf{pp})$; $I' = I \cup \mathsf{vk}'$;
- $\mathsf{pol} \leftarrow \mathsf{GenPolicies}(I')$;
- $(D, \mathsf{avk} = (i\mathsf{vk})_{i\in[\ell]}) \leftarrow \mathcal{A}^{\langle\mathcal{O}\rangle}(pol, \mathsf{vk}')$;
- $b \leftarrow (\mathcal{A} \leftrightarrow \mathsf{CredVerify}(pol, (i\mathsf{vk})_{i\in[\ell]}, D))$
- **if** $b = 1 \wedge \exists i \in [\ell] : [i\mathsf{vk}] = [\mathsf{vk}'] \ \wedge \ (i\mathsf{vk})_{i\in[\ell]} \subset I' \ \wedge D \not\subset \bigcup_{i\in[\ell]} A_{ui}, \ \forall u \in \mathcal{CU}$

  return 1
- else **return** 0

Figure 3.4: Experiment $\mathsf{ExpUnf}_{\mathsf{IhMA},\mathcal{A}}(\lambda)$

Anonymity. Anonymity requires that a malicious verifier cannot distinguish between two users. Thus we allow the adversary to output two sets of credentials, attributes, as well as a key-policy *pol*, attribute subset D, and issuers' public keys (can be corrupted). The adversary has adaptive access to an oracle that on the input of two distinct user indexes j_0 and j_1, acts as one of the two credential owners (depending on bit b) in the verification. To make the game non-trivial, we impose restrictions that the subset D is either satisfied or not by both credentials, i.e., $D(\mathsf{A}) = 1 \Rightarrow D \subseteq \cup_{i\in[\ell]}\mathsf{A}_i$ if attributes in A satisfy D and $D(\mathsf{A}) = 0 \Rightarrow D \not\subseteq \cup_{i\in[\ell]}\mathsf{A}_i$ otherwise. The essence of the game is captured by the oracles $\mathcal{O}_b^{\mathsf{Anon}}$ in Fig 3.5.

Definition 39 (Anonymity)**.** *An* IhMA *is anonymous, if for* $\lambda \in \mathbb{N}$*, any* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *so that* $|\Pr[\mathsf{ExpAno}^0_{\mathsf{IhMA},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{ExpAno}^1_{\mathsf{IhMA},\mathcal{A}}(\lambda) = 1]| \leq \frac{1}{2} + \epsilon(\lambda)$*, experiments are defined in Fig 3.5, respectively.*

$\underline{\mathsf{ExpAno}^b_{\mathsf{IhMA},\mathcal{A}}(\lambda)}$:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
- $(j_0, j_1, pol, (i\mathsf{vk})_{i\in[\ell]}) \leftarrow \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp})$
- $b' \leftarrow \mathcal{A}^{\langle\mathcal{O}^{\mathsf{Anon}}_b,\mathcal{O}\rangle}(\mathsf{st})$
- $\mathbf{return}(b = b')$

$\underline{\mathcal{O}^{\mathsf{Anon}}_b(j_0, j_1, D, pol)}$:

- If j_0 or $j_1 > |\mathcal{L}_{\mathsf{cred}}|$, return $\perp$.
- Else, parse $\mathcal{L}_{\mathsf{cred}}[j_0]$ as $(u_0, \mathsf{A}_{0i}, \mathsf{cred}_{0i})_{i\in[\ell]}$ and $\mathcal{L}_{\mathsf{cred}}[j_1]$ as $(u_1, \mathsf{A}_{1i}, \mathsf{cred}_{1i})_{i\in[\ell]}$, such that $\forall i$, $\mathsf{cred}_{bi} \leftarrow [\mathsf{CredObtain}(u\mathsf{sk}_b, i\mathsf{vk}, \mathsf{A}_{bi}) \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}_b, A_{bi})]$
- If $D(\mathsf{A}_0) \neq D(\mathsf{A}_1) \vee (u_0, u_1) \notin \mathcal{HU}$, return $\perp$.
- Otherwise run: $\mathsf{CredShow}(u\mathsf{sk}_b, pol, \{(i\mathsf{vk}, \mathsf{A}_{bi})\}_{i\in[\ell]}, \mathsf{cred}_b, D) \leftrightarrow \mathcal{A}$, where $\mathsf{cred}_b \leftarrow \mathsf{CredAggr}\left(u\mathsf{sk}_b, \{(i\mathsf{vk}, \mathsf{A}_{bi}, \mathsf{cred}_{bi})\}_{i\in[\ell]}\right)$

Figure 3.5: Experiment $\mathsf{ExpAno}_{\mathsf{IhMA},\mathcal{A}}(\lambda)$

Issuer-hiding. Issuer-hiding refers to the property that a malicious verifier cannot distinguish if verification keys belong to the credential issuer. We let $\mathcal{A}$ output a set of issuers and a key-policy *pol*. We consider key policies $\{\sigma_i, i\mathsf{vk}\}_{i\in[n]}$. Here, σ_i is a signature on an issuer's public key $i\mathsf{vk}$ which is generated by the verifier. Consequently, users can demonstrate that credential verification is performed using the verification key as specified in the key policy. Note that this definition assumes honest issuers.

Definition 40 (Issuer-Hiding)**.** *An* IhMA *provides issuer-hiding, if for all* $\lambda \in \mathbb{N}, \ell > 1$*, for all* $D \not\subseteq \cup_{i\in[\ell]}\mathsf{A}_i$ *and* $(\mathsf{pp}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\lambda)$ *s.t:*

$$\Pr\left[\begin{array}{l} (i\mathsf{sk}, i\mathsf{vk})_{i\in[\ell]} \xleftarrow{\$} \mathsf{IKeyGen}(\mathsf{pp}); \\ (I_0, I_1, pol, D) \xleftarrow{\$} \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp}, i\mathsf{vk}_{i\in[\ell]}); \\ (u\mathsf{sk}, u\mathsf{vk}) \xleftarrow{\$} \mathsf{UKeyGen}(\mathsf{pp}); b \xleftarrow{\$} \{0,1\}; \\ \forall i\mathsf{vk} \in I_b : (\mathsf{cred}_{b,i}, st) \xleftarrow{\$} \mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow \\ \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i); \\ \mathsf{cred}_b \leftarrow \mathsf{CredAggr}\left(u\mathsf{sk}, \{(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_{b,i})\}_{i\in[\ell]}\right) \\ b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{CredShow}}}(pol, I_b, D, \mathsf{A}_{i\in[\ell]}) \end{array} : \quad b^* = b \right] \leq \epsilon(\lambda)$$

where both $|I_0| = |I_1|$ *and* $I_0, I_1 \subseteq i\mathsf{vk}_{i\in[\ell]}$ *are one or a set of selected issuer(s).*

3.4.3 Constructions

Now we are ready to describes our two constructions of IhMA, the first being based on AtoSa (Def. 21) and SPSEQ [FHS19] and the second based on ATMS (Def. 31), a set commitment scheme SC (Def. 2.4.3.3), and SPSEQ. To enhance users' privacy and prevent issuers from learning attributes issued by other issuers, we change how aux for the signatures is computed. In particular, we commit to the attributes (messages) instead of including them

in plaintext. For example, this can be achieved using a hash-based commitment scheme, where a commitment value c is generated by computing $c := H'(a, r)$ with H' being a hash function modeled as a random oracle, a being the attributing being committed to, and r a sufficiently large random value. When issuing a credential, users can reveal the relevant message (attribute) a, the opening o, and the commitment value c. The signer then verifies if the c is correct for a and o before issuing the corresponding credential. We modify $\mathsf{GenAuxTag}(S)$ and VerifyAux in AtoSa and ATMS as follows:

- $\mathsf{GenAuxTag}(S)$: Given $S = \{(m_j, \mathsf{vk}_j)_{j\in[\ell]}\}$, choose $(\rho_1, \rho_2) \xleftarrow{\$} \mathbb{Z}_p$, set $c = P^{\rho_1}||P^{\rho_2}||(c_{m_j}||\mathsf{vk}_j)_{j\in[\ell]}$, where c_{m_j} is a hash commitment to j'th message and all vk are distinct. Output $\mathsf{aux} = (c, o_j)$ and tag $\tau = ((\rho_1, \rho_2), (T_1 = h^{\rho_1}, T_2 = h^{\rho_2}))$ with $h = H(c)$.
- $\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, m_j)$ Parse aux as (c, o). Check that $\tau \in t$ (i.e., that c has the form: $P^{\rho_1}||P^{\rho_2}||...$) check that c_j exists such that $(c_j, \mathsf{vk}) \in t$ and $\mathsf{Open}(c_j, o, m_j) = 1$ where vk is a verification key related to sk (in the same equivalence class). Also check that no other vk in aux has the same equivalence class as sk.

In our IhMA schemes, tags are user identities and are used to verify the user before issuing attributes.

3.4.3.1 AtoSa based IhMA Construction in Fig. 3.6.

Here, every issuer creates a credential (signature) σ_{1i} on an attribute a_i for the user u with tag τ (and the respective aux) verified with $i$vk by the AtoSa scheme. We cannot reveal the secret part of the tag to signers (issuers) as this would violate the security of the user. To obtain a credential through the Issuing protocols, a user is required to disclose the public parts of tag as identity to the issuer and then authenticate their identity via a ZKPOK.

Interactive signing. We can adapt the signing in a way that signers (issuers) don't learn (ρ_1, ρ_2) as follows:

- u sends $(\mathsf{aux}, (h, \mathbf{T}), \pi)$, where $\mathsf{aux} = P^{\rho_1}||P^{\rho_2}||(c_{m_i}, \mathsf{vk}_i)_{i\in[n]}$ and $\pi = \mathsf{ZKPOK}\{(\rho_1, \rho_2) : T_1 = h^{\rho_1} \wedge T_2 = h^{\rho_2} \wedge u_1 = P^{\rho_1} \wedge u_2 = P^{\rho_2}\}$.
- Signer (issuer) checks if proof π is valid and if so outputs $(h' = h^{\rho_1}, s = (h^{\rho_1})^{x_j + y_{1j}\cdot m_j} \cdot (h^{\rho_2})^{y_{2j}})$

We note that this interactive signing outputs signatures that are identical to that output by Sign and this is used in Issuance. For the Show protocol, we assume that verifier(s) have signed all accepted issuer keys using an SPSEQ scheme [FHS19]. A user u can take *pol* and the set of disclosed credentials D, aggregates the respective credentials (signatures) and randomizes the aggregated signature and tag. We note that alternatively, a user could already after Issuance aggregate all credentials to a constant-size (single) credential and then in Show protocol can provide a ZKPOK of the signature and selectively disclose the

required attributes (as originally done for PS signatures in [PS16a]). This also yields constant size credentials as noted in Table 6.3.5. We stick with the former approach here as it is more efficient for showing credentials, but one can easily switch to the other option. Moreover, In $\mathsf{IhMA}_{\mathsf{AtoSa}}$, only one attribute per vk can be issued. However, if an issuer needs to issue multiple attributes, they can easily generate multiple vks.

To hide the issuer's keys, u randomizes them using a random ω and adapts the signature for these randomized keys using ConvertSig. So far, we have created a compact randomized credential (proof) for attributes in D where issuer verification keys of this signature are hidden. The next step is to show that these random verification keys correspond to those keys signed by the verifier (using SPSEQ signatures) in *pol*. In this direction, u first collects signatures in *pol* according to issuer keys that are needed in the proof. Then u runs ChangRep of SPSEQ to randomize messages (which are issuer public keys) and signatures with the same randomness ω used in convert, i.e., randomized keys. Randomized issuer keys in a credential match with the messages signed by verifier in *pol*. Finally, u uses the randomized tag as a pseudonym for communication and provides a ZKPOK of the secret part of tag (secret keys and randomness) used in the credentials.

3.4.3.2 ATMS based IhMA Construction in Fig. 3.7.

We use the framework in [FHS19] in which one can combine mercurial or SPSEQ with a set commitment such that a credential is a signature on set commitment SC. One can then open a subset of messages from this commitment while randomizing both set commitment and signature together. This provides unlinkability and selective disclosure at the same time (see [FHS19]). Unlike the previous construction, we can aggregate credentials immediately after receiving them and have a constant-size credential but still avoid zero-knowledge proof of a signature in showing protocol (because of compatibility of EQ message relation of ATMS and SC randomization).

In the Show protocol, similar to the previous construction, u first collects the signatures required to prove the attributes D from *pol*. Then, for issuer-hiding similar to AtoSa it randomizes these SPSEQ signatures using ChangRep of SPSEQ with ω. For preparing a proof for D, a user (u) randomizes issuer verification keys in credentials using ConvertVK and converts the ATMS signature using ConvertSig with ω. Subsequently, u randomizes the signature with a tag using ChangRep. Finally, u opens a subset of attributes D from the set commitments. Now a verifier can check if these attributes are in the set commitments signed by some issuers in *pol*. Same as in the first construction, since all issuer keys are randomized due to the SPSEQ signature the issuers are hidden. We run a ZKPOK to prove that u knows all secret values related to the randomized tag like before. The only point left is the signing of set commitments, which is defined in one source group in [FHS19], but we need both groups. Subsequently, we show how one can combine set commitments with a tag-based DH message space.

Set commitments for $\mathcal{M}^H_{\mathsf{TDH}}$. The main point here is that we need to convert the set commitments space to $\mathcal{M}^H_{\mathsf{TDH}}$, which can be smoothly done as follows: In addition to

- $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{pp}_{\mathsf{AtoSa}} \leftarrow \Sigma_1.\mathsf{Setup}(1^\lambda) \wedge \mathsf{pp}_{\mathsf{SPSEQ}} \leftarrow \Sigma_2.\mathsf{Setup}(1^\lambda)$, output $\mathsf{pp} = (\mathsf{pp}_{\mathsf{AtoSa}}, \mathsf{pp}_{\mathsf{SPSEQ}})$. The attribute space is $\mathbb{Z}_p$.
- $\mathsf{UKeyGen}(\mathsf{pp}, S)$: Run $(\{\mathsf{aux}_j\}, (\tau, \mathbf{T})) \leftarrow \mathsf{GenIdxTag}(\mathsf{pp}, S)$, and return $(u\mathsf{sk} = \tau, u\mathsf{vk} = \mathbf{T}, \{\mathsf{aux}_j\})$ to u.
- $\mathsf{IKeyGen}(\mathsf{pp})$: Generate $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \Sigma_1.\mathsf{KeyGen}(\mathsf{pp})$, return $(i\mathsf{sk} = \mathsf{sk}, i\mathsf{vk} = \mathsf{vk})$ to an issuer i.
- $\mathsf{Issuance}$: On input $(\mathbf{T}, \mathsf{aux}_i, a_i)$, u and each issuer i act as follows for an attribute a_i:
  - u sends $(\mathbf{T}, \mathsf{aux}_i, \pi)$, to an issuer i, where π is a zero knowledge proof that the user knows the secret part of the given tag.
  - Issuer checks π is valid and runs $\sigma_i \leftarrow \Sigma_1.\mathsf{Sign}(i\mathsf{sk}, \mathbf{T}, \mathsf{aux}_i, a_i)$ and outputs (σ_i, a_i) to u or aborts if Sign outputs $\perp$.
  - u takes $(i\mathsf{vk}, \mathsf{cred}_i = (a_i, \sigma_i))_{i\in[\ell]}$, checks $\Sigma_1.\mathsf{Verify}(i\mathsf{vk}, a_i, \mathsf{cred}_i)_{i\in[\ell]}$, and saves $\mathsf{cred} = \{\mathsf{cred}_i = (\sigma_i, \tau), \mathsf{A}\}_{i\in[\ell]}$, where $\mathsf{A} = (a_i)_{i\in[\ell]}$.
- $\mathsf{GenPolicies}$: Generate a key pair $(v\mathsf{sk}, vpk) \leftarrow \Sigma_2.\mathsf{KeyGen}(\mathsf{pp})$, run $\sigma_{2i} \leftarrow \Sigma_2.\mathsf{Sign}(v\mathsf{sk}, i\mathsf{vk})$ for $i \in I$ where $i\mathsf{vk}$ is a message vector for SPSEQ, return $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$.
- Show: On input $\mathsf{cred} = \{(\sigma_i, \tau, \mathsf{A})_{i\in[\ell]}\}$, $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$, an D (a set of attributes) from $n \subseteq I$ issuers ($|D| = n$), u prepares a proof for D as:
  - Run $\sigma \leftarrow \Sigma_1.\mathsf{AggrSign}(\mathbf{T}, (i\mathsf{vk}, a_i, \sigma_i))_{i\in[D]}$ with $\mathsf{avk} = \{i\mathsf{vk}\}_{i\in[D]}$. For $\omega \in \mathbb{Z}_p^*$, run $\mathsf{avk}' \leftarrow \Sigma_1.\mathsf{ConvertVK}(\mathsf{avk}, \omega)$, $\sigma' \leftarrow \Sigma_1.\mathsf{ConvertSig}(\mathsf{avk}, D, \mathbf{T}, \sigma, \omega)$, and randomize $(\sigma'', \mathbf{T}') \leftarrow \Sigma_1.\mathsf{RandSign}(\mathsf{vk}, \mathbf{T}, m, \sigma', \upsilon)$ for $\upsilon \in \mathbb{Z}_p^*$.
  - Run $(\sigma'_{2i}, \mathsf{avk}') \xleftarrow{\$} \Sigma_2.\mathsf{ChangRep}(\mathbf{M}_i = \mathsf{vk}_i, \sigma_{2i}, \omega)_{i\in[n]}$ where avk' is the same as $\mathsf{avk}' \leftarrow \Sigma_1.\mathsf{ConvertVK}$.
  - Prove in zero knowledge that the user knows the secret key for the tag $\mathbf{T}'$, yielding π, send $(\sigma'', \mathsf{nym} = \mathbf{T}', \sigma'_{2i}, \pi)_{i\in[n]}$ to a verifier V.
- $\mathsf{CredVerify}$: Output 1, if $\pi \wedge \Sigma_1.\mathsf{VerifyAggr}(\mathsf{avk}', \mathbf{T}', D, \sigma') \wedge \Sigma_2.\mathsf{Verify}(v\mathsf{vk}, \mathbf{M}, \sigma'_2) = 1$, where $\mathbf{M} = \mathsf{avk}'$ and $\mathbf{T}' = \mathsf{nym}$. Output 0 if this check fails.

Figure 3.6: Our IhMA scheme (Σ_1 and Σ_2 denote AtoSa and SPSEQ [FHS19], respectively)

credentials issuers, we also define a Trusted Authority $\mathsf{T}A$ who holds the trapdoor α of the set commitment scheme and can create commitments for the attributes of users who want to register in the system. WLOG, let us for simplicity assume only one attribute set $\mathsf{A} = (\mathsf{A}, \eta)$, where we have a fixed constant η which is never opened in practice and it is the same for all (it is just required for anonymity). It works as follows:

- The user sends a tag $\mathbf{T}$ and aux to $\mathsf{T}A$.
- $\mathsf{T}A$ computes a set commitment in both groups ($\mathbf{C} = (C_1, C_2), \hat{\mathbf{C}} = (\hat{C}_1, \hat{C}_2)$) (i.e., $(\mathbf{M}, \mathbf{N})$) with tag, where $(C_2, \hat{C}_2)$ are dummy commitments for a fixed constant η and the other one for the (real) attribute set A. More precisely: $\mathsf{T}A$ computes the commitment in $\mathbb{G}_1$ to base h^{ρ_i} and the one in $\mathbb{G}_2$ in base $\hat{P}$: $C_1 = (h^{f_{\mathsf{A}}(\alpha)})^{\rho_1}$, $\hat{C}_1 = \hat{P}^{f_{\mathsf{A}}(\alpha)}$, $C_2 = (h^{\eta})^{\rho_2}$

and $\hat{C}_2 = \hat{P}^{\eta}$ such that such that we have $\bigwedge_{i\in[2]} e(T_i, \hat{C}_i) = e(C_i, \hat{P})$, where $h = H(c)$, $\mathsf{aux} = (c, o)$, $c = P^{\rho_1}||P^{\rho_2}||(c_{\mathsf{A}_i}||\mathsf{vk}_j)_{j\in[2]}$, returns $(\mathbf{C}, \hat{\mathbf{C}})$. Note that $c_{\mathsf{A}} := H'(\mathsf{A}, r)$.

Note that α is a trapdoor kept by $\mathsf{T}A$, but $\mathsf{T}A$ does not need to know (ρ_1, ρ_2) (e.g., C_i be computed as $(T_1)^{f_{\mathsf{A}}(\alpha)}$). A multiparty computation protocol can also be used to hide other user details from $\mathsf{T}A$. A user can first randomize set commitment exactly like our tag-based message with (μ, υ) as $(\mathbf{C}^{\mu\upsilon}, \hat{\mathbf{C}}^{\upsilon})$ and use υ as opening information to open any subset values from $\hat{C}_1$ and still verify as follows: verifying the $\mathsf{OpenSubset}$ works $e(P, \hat{C}_1) = e(P^{f_D(\alpha)}, W)$. Consequently, we do not need any fundamental change on SC construction, and it works as stated in 2.4.3.3. In our construction, we make it explicit as:

- $\mathsf{SC.Commit}_3(\mathsf{A}, \alpha, \mathbf{T}, h) \rightarrow ((\mathbf{C}, \hat{\mathbf{C}}), O)$: On input a set $\mathsf{A} = (\mathsf{A}, \eta)$, $\mathbf{T}$ and h, compute a commitment: $C_1 = (T_1^{f_{\mathsf{A}}(\alpha)})$, $\hat{C}_1 = \hat{P}^{f_{\mathsf{A}}(\alpha)}$, $C_2 = (T_2^{\eta})$ and $\hat{C}_2 = \hat{P}^{\eta}$, output $((\mathbf{C}, \hat{\mathbf{C}}), O)$ with $O \leftarrow \perp$.

Now, we can use the same technique as AtoSa to not reveal (ρ_1, ρ_2) to issuers when signing the above commitments $(\mathbf{C}, \hat{\mathbf{C}})$ as follows:

Interactive signing. We can adapt the signing in a way that signers (issuers) don't learn (ρ_1, ρ_2) as follows:

- u sends $(\mathsf{aux}, \mathbf{T}, (\mathbf{C}, \hat{\mathbf{C}}), \pi)$, where
  $\pi = \mathsf{ZKPOK}\{(\rho_1, \rho_2) : T_1 = h^{\rho_1} \wedge T_2 = h^{\rho_2} \wedge u_1 = P^{\rho_1} \wedge u_2 = P^{\rho_2}\}$, where P^{ρ_1} and P^{ρ_2} are in aux.
- Signer (issuer) checks if proof π is valid and if so outputs
  $(h = H(c), b = \prod T_i^{z_i}, s = (h^x \cdot \prod_{i\in[2]}(C_i)^{y_i}))$.

Again we note that this interactive signing outputs signatures that are identical to that output by Sign and this is used in $\mathsf{Issuance}$.

Achieving constant-size credentials. This can be achieved by following these steps: 1) Users can obtain the (h^{α_i}) values from the TA instead of the commitments. 2) During the issuing phase, users can aggregate all the credentials received from issuers. 3) The commitments can then be recomputed using randomness and the obtained information, eliminating the need to store them. Note that in this case the size of the $|\mathbf{Show}|$ operation will become linear with respect to N instead of K.

Theorem 3.4.1. *The above* IhMA *constructions in Fig. 3.7 and in Fig. 3.6 are correct, unforgeable, anonymous, and issuer-hiding.*

Proof. We provide proof for the first and second constructions.

Lemma 3.4.2 (Unforgeability construction)**.** *Let* ZKPOK *be a simulation-sound extractable ZKPoK, and* SPSEQ *be unforgeable signature, if* AtoSa *is unforgeable, then the* IhMA *construction Fig 3.6 is unforgeable.*

Proof. Intuitively, $\mathcal{A}$ has a potential ways of breaking unforgeability: if he can forge a AtoSa signature on the challenge public key (that is, $\mathcal{A}$ does not possession proper attributes, but it can perform verification by forging credentials). We show that if an adversary $\mathcal{A}$ can win the unforgeability game (Def. 38) with non-negligible probability. We then construct an adversary (reduction) $\mathcal{B}$ that breaks the unforgeability of AtoSa (Def. 23). Note that we can extract witness from ZKPOK and assume this will only fail with negligible probability. Lets us assume that $\mathsf{A}_i = a_i$ is an attribute for simplicity, now we show this reduction as follows:

Reduction. The reduction is straightforward. $\mathcal{B}$ communicates with a challenger $\mathcal{C}$ in the unforgeability game of AtoSa and $\mathcal{B}$ simulates the IhMA-unforgeability game for $\mathcal{A}$. $\mathcal{B}$ receives from $\mathcal{C}$ values $(\hat{X} = P^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2})$ and public parameters pp of BG. Next, $\mathcal{B}$ sets $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2)$ as the challenge key and sends $(\mathsf{pp}, \mathsf{vk}')$ to $\mathcal{A}$. As in the real game, all oracles are executed normally, except for following ones which use the signing oracle in AtoSa instead of using the (challenge) signing key sk':

$\mathcal{O}^{\mathsf{User}}(u)$: On input a user identity u. If $u \in \mathcal{HU}$ or $u \in \mathcal{CU}$, return $\perp$. Else, create a fresh entry u by running $(u\mathsf{sk}, u\mathsf{vk}) \leftarrow \mathsf{U}KeyGen$ but create aux using commitments, adding u and $(u\mathsf{sk}, u\mathsf{vk}, \mathsf{aux})$ to the list $\mathcal{HU}$ and $\mathcal{L}_{uk}$, receptively. Return $u\mathsf{vk}$.

$\mathcal{O}^{\mathsf{ObtIss}}(i, u, \mathsf{A}_i)$: If $u \notin \mathcal{HU} \vee i \notin \mathcal{HCI} \cup \mathsf{vk}'$, return $\perp$. Else if $i\mathsf{vk} \neq \mathsf{vk}'$, find entries $((u\mathsf{sk} = \tau, \mathsf{aux}) \in \mathcal{L}_{uk}, i\mathsf{sk} \in \mathcal{HCI})$, and compute $\sigma_i \leftarrow \mathsf{Sign}(i\mathsf{sk}, \tau, \mathsf{aux}, \mathsf{A}_i)$ (note that with knowledge of $i\mathsf{sk}$, $\mathcal{B}$ can compute a signature on it's own). Else $i\mathsf{vk} = \mathsf{vk}'$, asks the query $\sigma_i \leftarrow \mathcal{O}^{\mathsf{Sign}}(\mathsf{A}_i, \mathsf{aux}, \tau)$ of AtoSa, which the oracle runs $\mathsf{Sign}(\mathsf{sk}', \tau, \mathsf{aux}, \mathsf{A}_i)$, adds the entry $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$, where $\mathsf{cred}_i = (\sigma_i, \tau)$.

$\mathcal{O}^{\mathsf{Issue}}(i, u, \mathsf{A}_i)$: If $u \notin \mathcal{CU} \vee i \notin \mathcal{HCI} \cup \mathsf{vk}'$, it returns $\perp$. Else, if $i\mathsf{vk} \neq \mathsf{vk}'$, compute $\sigma_i \leftarrow \mathsf{Sign}(i\mathsf{sk}, \tau, \mathsf{aux}, \mathsf{A}_i)$. Else ask the query $\sigma_i \leftarrow \mathcal{O}^{Sign}(\mathsf{A}_i, \mathsf{aux}, \tau)$ of AtoSa, add the entry $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$, where $\mathsf{cred}_i = (\sigma, \tau)$.

Obviously, $\mathcal{B}$ handles any oracle query (assuming a simulated ZKPOK for issuing and showing protocols). Thus, at the end of the game, $\mathcal{B}$ simulates all oracles perfectly for $\mathcal{A}$. To do this, $\mathcal{B}$ interacts with $\mathcal{A}$ in a showing. If $\mathcal{A}$ outputs a valid showing proof as $(\mathsf{avk}, \sigma^* = (h^*, s^*), D, \mathsf{nym}^* = \mathbf{T}^*)$ and conducting $\pi = \mathsf{ZKPOK}(\mathsf{nym}^*)$ then $\mathcal{B}$ extracts from the proof π in the Show, lets called the value (ρ_1^*, ρ_2^*) related to the nym^* (the tag tuple $(\mathbf{T}^*, (\rho_1^*, \rho_2^*))$) and stores all elements. Moreover, no credentials owned by corrupt users can be valid on this set of messages D (as $\mathcal{A}$ can win the unforgeability game). This means that, for all credentials cred_{ui} on A_{ui} and $(u\mathsf{sk})$ with $u \in \mathcal{CCU}$, we have $D \not\subseteq \cup_{i \in [\ell]} \mathsf{A}_{ui}$. From the definition we have that at least one of the key in $\mathsf{vk}_j \in \mathsf{avk}$ should be the challenge key $[\mathsf{vk}_j]_{\mathcal{R}} = [\mathsf{vk}']_{\mathcal{R}}$ such that the related attribute is in the set $\mathsf{A}_j \in D$. Finally we can find all $i\mathsf{sk} \in \mathcal{HCI} \cup \mathcal{CCI}$ corresponding to $[i\mathsf{vk}]_{\mathcal{R}} \in I'$ for all $i \in [\ell]$, and output $\mathsf{ask} = (i\mathsf{sk})_{i \in [\ell]}$. Note that even if adversarial keys are randomized, we can output initial secret keys registered in the list and reduction still works for the AtoSa scheme because of keys class $[\mathsf{vk}]_{\mathcal{R}}$. In all cases, this means that $(\mathsf{avk}, (\tau^*, \mathbf{T}^*), D, \mathsf{ask}, \sigma^*)$ is a valid forgery

against our signature scheme, $\mathcal{B}$ breaks thus unforgeability of AtoSa which concludes our proof, assuming SPSEQ is unforgeable.

Lemma 3.4.3. *Let* ZKPOK *be a simulation-sound extractable ZKPoK, if* AtoSa *is origin-hiding of ConvertSig and public key class-hiding,* SPSEQ *is perfect adaption, then the* IhMA *construction in Fig 3.6 is anonymous and issuer hiding.*

Since these properties follow almost immediately from the zero-knowledge property and the privacy notions of the underlying signature AtoSa (Def. 3.2.4).

Anonymity. From the randomization (unlinkability) of the AtoSa signature, a tuple $(\sigma = (h', s), \mathbf{T} = (T_1, T_2))$ can be hidden by randomizing them with secret randoms $\mu \in \mathbb{Z}_p^*$ as $(\sigma = (h'^{\mu}, s^{\mu}), \mathbf{T}^{\mu})$, where does not leak any information about the initial tag/signature (tag acts as pseudonym in the interaction). In addition, in the Show protocol, the proof of knowledge of the witness (tag's secret part) is zero-knowledge. This also does not leak any information about secrets either. Consequently, a credential does not leak any information about $u\mathsf{sk}$ or $u\mathsf{vk}$. Note that the Anonymity experiment guarantees that the witnesses used when computing π are valid for both $b = 0$ and $b = 1$ and also the signature/tag is randomized correctly. This means that the Anonymity experiment is indistinguishable from one where the ZKPOK simulator creates π. In the end, the view of $\mathcal{A}$ is independent of b. More formally we provide proof as follows:

Proof. The proof follows a sequence of games until a game where answers for the query to $\mathcal{O}_b^{\mathsf{Anon}}$ is independent of the bit b. Let S be the event, and for $i = 1, \ldots, n$, the construction defines an event S_i in Game_i. In Game_1 we simulate all $ZKPoKs$. In Game_2 we replace all ConvertTag and ConvertSig calls with freshly generated signatures and In Game_3 we replace the $\mathbf{T}$ with a representative element of the same class.

Game_0**:** The original game as Def. 56.

Game_1**:** As Game_0, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: All proofs $\mathsf{ZKPoK}(\mathsf{nym}_p)$ in CredShow and Obtain respectively, are simulated.

$\mathsf{Game}_0 \to \mathsf{Game}_1$: By perfect zero-knowledge of ZKPoK, we have that

$$\Pr[S_1] = \Pr[S_0]$$

Game_2**:** Consider Game_1, but with the following modifications. Let qu be the maximum number of queries to $\mathcal{O}^{\mathsf{User}}$. In the start, during Game_1, a selection is made where w is chosen randomly from the set $[qu]$ (guessing that the user with the j_b-th credential is registered at the w-th call to $\mathcal{O}^{\mathsf{User}}$). Runs $\mathcal{O}^{\mathsf{User}}$, $\mathcal{O}^{\mathsf{C}orruptU}$ and $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$ as follows:

- $\mathcal{O}^{\mathsf{User}}(u, S)$: As in Game_1, but if this is the w-th call then, setting $u* \leftarrow u$.
- $\mathcal{O}^{\mathsf{C}orruptU}(u)$: If $u \in \mathcal{CU}$ or $u \in \mathcal{O}_b^{\mathsf{Anon}}$, it returns $\perp$ (as before). If $u = u^*$ then stops and returns a random $b' \xleftarrow{\$} \{0, 1\}$. Else, it outputs $u\mathsf{sk}$ and credentials, and moves u from $\mathcal{HU}$ to $\mathcal{CU}$.

- $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: As in Game_2, except that if $u^* \neq \mathcal{L}_{\mathsf{cred}}[j_b]$, the experiment stops outputting $b' \xleftarrow{\$} \{0,1\}$.

$\mathsf{Game}_1 \to \mathsf{Game}_2$: By assumption, $\mathcal{O}_b^{\mathsf{Anon}}$ is called at least once with some input (j_0, j_1, D) such that $u_0 \leftarrow \mathcal{L}_{\mathsf{cred}}[j_0], u_1 \leftarrow \mathcal{L}_{\mathsf{cred}}[j_1] \in \mathcal{HU}$. If $u^* = u_b$ then $\mathcal{O}_b^{\mathsf{Anon}}$ and $\mathcal{O}^{\mathsf{CorruptU}}$ do not stop (not have been called on u_b before that call to $\mathcal{O}_b^{\mathsf{Anon}}$ (else $u_b \notin \mathcal{HU}$); if called afterwards, it returns $\perp$, where $u^* \in \mathcal{O}_b^{\mathsf{Anon}}$). If $u^* = [u_b]$ with probability $\frac{1}{q_u}$, the probability that the game does not stop is $\frac{1}{q_u}$, and so

$$\Pr[S_2] \geq (1 - \frac{1}{q_u})\frac{1}{2} + \frac{1}{q_u} \cdot \Pr[S_1]$$

Game_3**:** As Game_2, except $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: pick a random $\mathbf{T} \leftarrow \mathcal{T}$ and performs the showing with $D = (d_i)_{i \in [k]}$. The only difference is the choice of $\mathbf{T}$.

$\mathsf{Game}_2 \to \mathsf{Game}_3$: The difference between two games is that we use the tag class hiding Def. 25,

which indirectly implies DDH. Indeed, the reduction accepts $\mathbf{T}, \mathbf{T}^b$ from the tag class-hiding challenger and uses these for users. The oracles are simulated as in Game_2, excepting for the following ones:

- $\mathcal{O}^{\mathsf{User}}(u, S)$: Like in Game_1, but if it is the w-th call then $u^* \leftarrow u$, sets $u\mathsf{sk}[u] \leftarrow \perp$ and $u\mathsf{vk} \leftarrow \mathbf{T}$.
- $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: As in Game_2, except that for $u^* = \mathcal{L}_{\mathsf{cred}}[j_b]$, the experiment run show for $\mathbf{T}^b$ which is either $\mathbf{T}^{(0)} \xleftarrow{\$} \mathcal{T}$ or $\mathbf{T}^{(1)} \xleftarrow{\$} [\mathbf{T}]_{\mathcal{R}_\tau}$. Picks b and sends $(\mathbf{T}^b, \sigma_b, \pi)$ to $\mathcal{A}$, and receives b' form $\mathcal{A}$. Return b' as answer to the tag class-hiding game. We thus have:

$$|\Pr[S_2] - \Pr[S_3]| \leq \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2ql)\frac{1}{p}$$

Game_4**:** As Game_3, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: Like in Game_3, but for $\mu, \upsilon \in \mathbb{Z}_p^*$, all executions of RndSigTag and ConvertSig for the credential and tag $(u_b, \mathsf{A}', \sigma_b) \leftarrow \mathcal{L}_{\mathsf{cred}}[j_b]$ are replaced by freshly generated signatures (note that we can randomize signatures and output uniformly random elements in the respective spaces and we know the related secret keys). So, oracles are simulated as in Game_3.

$\mathsf{Game}_2 \to \mathsf{Game}_3$: By Origin-hiding (ConvertSig) and (ConvertTag), signatures obtained from RndSigTag and ConvertSig are identically distributed for all $(\mathsf{A}, \mathbf{T}, \mathsf{vk}, \sigma)$. We thus have

$$\Pr[S_3] = \Pr[S_4]$$

At the end, $\mathcal{O}_b^{\mathsf{Anon}}$ outputs the random signature σ and tag, and a simulated proof; the bit b is thus information-theoretically hidden from $\mathcal{A}$. probability analysis is similar to [FHS19, MSBM23].

Proof of issuer-hiding for $\mathsf{IhMA}_{\mathsf{AtoSa}}$. We refer to [MBG+23] for the proof of issuer-hiding.

Lemma 3.4.4 (Unforgeability of construction in Fig 3.7)**.** *Let* ZKPOK*,* SC*, and* SPSEQ *be a simulation-sound extractable ZKPoK and a binding commitment and unforgeable signature receptively, if* ATMS *is unforgeable, then the* IhMA *construction in Fig 3.7 is unforgeable.*

Proof. Similar to Lemma 3.4.2, we show that if an adversary $\mathcal{A}$ can win the unforgeability game (Def. 38) with non-negligible probability. We then construct an adversary (reduction) $\mathcal{B}$ that breaks the unforgeability of ATMS (Def. 33). Assume that we can extract witness from ZKPOK. It follows the same reductions as AtoSa, so we only show the differences here as follows:

Reduction. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in the unforgeability game of ATMS and $\mathcal{B}$ simulates the IhMA-unforgeability for $\mathcal{A}$.

- $\mathcal{B}$ receives from $\mathcal{C}$ values $(\hat{X} = P^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2})$ and public parameters $\mathsf{pp}_{\mathsf{ATMS}}$. Then, it sets $\mathsf{vk}' = (\hat{X} = P^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2})$ as the challenge issuer key, $\mathsf{pp}_{\mathsf{SPSEQ}} \leftarrow \Sigma_2.\mathsf{Setup}(1^\lambda)$ and pick α and create set commitment public parameters $\mathsf{pp}_{\mathsf{SC}} \leftarrow \mathsf{SC.Setup}(1^\lambda, \alpha)$ and send vk' and $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ATMS}}, \mathsf{pp}_{\mathsf{SC}}, \mathsf{pp}_{\mathsf{SPSEQ}})$ to $\mathcal{A}$.
- As shown in Lemma 3.4.2, $\mathcal{B}$ handles any oracle query and simulates all oracles perfectly for $\mathcal{A}$ similar to Lemma 3.4.2, and never aborts, we skip to mention them here.
- If $\mathcal{A}$'s winning condition is not fulfilled, $\mathcal{B}$ aborts.
- Otherwise, $\mathcal{A}$ is able, with some probability, to prove possession of a credential on attributes D^*.

So $\mathcal{B}$ interacts with $\mathcal{A}$ as verifier in a showing protocol. If $\mathcal{A}$ outputs a valid showing proof as $(\mathsf{avk}, (\mathbf{C}^*, \hat{\mathbf{C}}^*), \sigma^*, D^*, W^*, \mathsf{nym}^* = \hat{\mathbf{T}}^*)$ and conducting $\pi = \mathsf{ZKPoK}(\mathsf{nym}^*)$ then $\mathcal{B}$ extracts from the proof π, called the value $\tau^* = (\rho_1^*, \rho_2^*)$ related to the nym^* and stores all elements. Also, we know that no credentials owned by corrupt users can be valid on this set of messages D^* (as $\mathcal{A}$ can win the unforgeability game). This means that, for all credentials cred_{ui} on $(u\mathsf{sk})$ with $u \in \mathcal{CCU}$, we have $(D^* \not\subseteq \mathsf{A})$. From the definition we have that at least one of the key in $\mathsf{vk}_j \in \mathsf{avk}$ should be the challenge key $\mathsf{vk}_j = \mathsf{vk}'$ such that the related attribute is in the set $\mathsf{A}_j \in D^*$. Find all $i\mathsf{sk} \in \mathcal{HCI} \cup \mathcal{CCI}$ corresponding to $[i\mathsf{vk}]_{\mathcal{R}} \in [\ell]$. Finally, with all these cases, we conclude that $(\mathsf{avk}, \mathsf{ask}, (\tau^*, \hat{\mathbf{T}}^*), D^*, \sigma^*, (\mathbf{C}^*, \hat{\mathbf{C}}^*))$ is a valid forgery against our signature scheme, where $(\mathbf{C}^*, \hat{\mathbf{C}}^*) = (\mathbf{M}^*, \mathbf{N}^*)$. So $\mathcal{B}$ breaks

unforgeability of ATMS which concludes our proof. Note that we also assume SC is a binding and hiding commitment and SPSEQ is unforgeable, so $\mathcal{A}$ can not forge proof by breaking binding of SC or unforgeability of SPSEQ.

Lemma 3.4.5. *Let* ZKPOK *be a simulation-sound extractable ZKPoK,* ATMS *is origin-hiding of ConvertSig and ChangRep and public key class-hiding, and* SPSEQ *is perfect adaption, then the* IhMA *construction in Fig 3.6 is anonymous and issuer hiding.*

Proof. The argument follows the one in Lemma 3.4.3 except that we replace privacy notations of AtoSa by ATMS and show that a new uber assumption holds 3.4.6 for the randomization set commitments and tag. The proof follows a sequence of games until a game where answers for the query to $\mathcal{O}_b^{\mathsf{Anon}}$ is independent of the bit b. In Game_1 we replace all ChangRep and ConvertSig calls with freshly generated signatures. In Game_2 we simulate all $ZKPoK$s and In Game_3 we replace the respective commitment vectors $\mathbf{C}$ with a represantive element of the same class.

Game_0**:** The original game as given in Definition 56.

Game_1**:** As Game_0, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: Like in Game_0, but for $\mu, \upsilon \in \mathbb{Z}_p^*$, all executions of ChangRep and ConvertSig for the credential and tag $(i_b, \mathsf{A}', \sigma_b) \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_b]$ are replaced by randomized signatures (note that we can randomize signatures and output uniformly random elements in the respective spaces). So, oracles are simulated as in Game_1.

$\mathsf{Game}_1 \rightarrow \mathsf{Game}_0$: By Origin-hiding (ConvertSig), adapted privacy (ChangRep) signatures obtained from ChangRep and ConvertSig are identically distributed for all $(\mathsf{A}, \mathbf{T}, \mathsf{vk}, (\mathbf{C}, \hat{\mathbf{C}}))$. We thus have

$$\Pr[S_0] = \Pr[S_1]$$

Game_2**:** As Game_1, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: All proofs $\mathsf{ZKPoK}(\mathsf{nym}_p)$ in CredShow and ObtIss respectively, are simulated.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: By perfect zero-knowledge of ZKPoK, we have that

$$\Pr[S_1] = \Pr[S_2] \Rightarrow \Pr[S_0] = \Pr[S_1] = \Pr[S_2]$$

Game_3**:** As Game_2, except that for $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: it replaces all $(\mathbf{C}_i, \hat{\mathbf{C}}_i)$ and $\mathbf{T}$ with another vectors in the same equivalence class and performs the showing with $D = (d_i)_{i \in [k]}$ and $W_i \leftarrow f_{d_i}(a)^{-1} \cdot \hat{C}_{1i}$. The only difference is the computation of $(\mathbf{C}_i, \hat{\mathbf{C}}_i)$; while all W_i are distributed as in Game_4, in particular, they are unique elements satisfying VerifySubset.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: The difference between two games is that we use the uber assumption Def. 3.4.6 to create set commitments. Oracles are simulated as in Game_4, except for the following oracles as:

- $\mathcal{O}^{\mathsf{Obtain}}(u, i, \mathsf{A}_i)$: As in Game_2, except for the computation of following values: compute the polynomials $f_1 = F_{\mathsf{A}_i}(\alpha) \cdot \rho_1 \cdot r$, $f_2 = \rho_2 \cdot r \cdot \eta$, $\hat{f}_1 = F_{\mathsf{A}_i}(\alpha)$, and $\hat{f}_2 = \eta$, where $r \xleftarrow{\$} \mathbb{Z}_p$ is for h and come from RO and η is a constant and same for all commitments. Then for $i \in [2]$ it calls $C_i \leftarrow \mathcal{O}^{uber}(1, f_i, f_i)$ and $\hat{C}_i \leftarrow \mathcal{O}^{uber}(2, \hat{f}_i, \hat{f}_i)$ (all C_i and $\hat{C}_i$ are distributed as in the original game.)
- $\mathcal{O}^{\mathsf{CredShow}}(j, pol, D)$: As in Game_2, it computes the witness $W_i \leftarrow f_{d_i}(\alpha)^{-1} \cdot \hat{C}_{1i}$ (W_i is thus distributed as in the original game and $D = (d_i)_{i \in [k]}$.)
- $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: As in Game_4, with the following difference. For two honest users $u \leftarrow L_{\mathsf{cred}}[j_0]$ and $u' \leftarrow L_{\mathsf{cred}}[j_1]$, first parses $L_{\mathsf{cred}}[j_b]$ as $(u, \mathsf{A}_i, \mathsf{cred}_i)$ and $(u', \mathsf{A'}_i, \mathsf{cred}'_i)$.

  Compute polynomials for each set A_i as $f_{1i} = F_{\mathsf{A}_i}(\alpha) \cdot \rho_1 \cdot r$, $f_{2i} = \rho_2 \cdot r \cdot \eta$, $\hat{f}_{1i} = F_{\mathsf{A}_i}(\alpha)$, and $\hat{f}_{2i} = \eta$.

  Compute polynomials for each set A'_i as $f'_{1i} = F_{\mathsf{A'}_i}(\alpha) \cdot \rho'_1 \cdot r'$, $f'_{2i} = \rho'_2 \cdot r' \cdot \eta$, $\hat{f}'_{1i} = F_{\mathsf{A'}_i}(\alpha)$, and $\hat{f}'_{2i} = \eta$.

  Compute polynomials for the tags $\mathbf{T}$ and $\mathbf{T}'$ as $f'_{t1} = \rho'_1 \cdot r'$, $f'_{t2} = \rho'_2 \cdot r'$, $f_{t1} = \rho_1 \cdot r$, and $f_{t2} = \rho_2 \cdot r$.

  Then it calls $C_{1i} \leftarrow \mathcal{O}^{uber}(1, f_{1i}, f'_{1i})$ and $C_{2i} \leftarrow \mathcal{O}^{uber}(1, f_{2i}, f'_{2i})$, same for $\hat{C}_{1i} \leftarrow \mathcal{O}^{uber}(2, \hat{f}_{1i}, \hat{f}'_{1i})$ and $\hat{C}_{2i} \leftarrow \mathcal{O}^{uber}(2, \hat{f}_{2i}, \hat{f}'_{2i})$. Also, to compute tags, it calls

  $T_1 \leftarrow \mathcal{O}^{uber}(1, f_{t1}, f'_{t1})$ and $T_2 \leftarrow \mathcal{O}^{uber}(1, f_{t2}, f'_{t2})$.

  It sends $(\mathbf{C}_i, \hat{\mathbf{C}}_i)$ and $\mathbf{T}$ to $\mathcal{A}$, and receives b' form $\mathcal{A}$.

$$|\Pr[S_2] - \Pr[S_3]| \leq \epsilon_{uber}(\lambda)$$

  Return b' as answer to the uber assumption. We have simulated Game_2 if the uber assumption was "real" and Game_3 otherwise.

Uber Assumption To demonstrate the anonymity of $\mathsf{IhMA}_{\mathsf{AtoSa}}$, we introduce a variant of the uber assumption. For a formal definition of the uber assumption and its proofs, we refer the reader to our paper [MBG+23].

Theorem 3.4.6. *Let $\mathcal{A}$ be a adversary that solves decisional uber in a bilinear generic group $\mathcal{G}$ of prime order p of type 3, making at most (m_1, m_2, m_T, m_p) generic queries and (q_1, q_2, q_T) and (d_1, d_2, d_T) the cardinality, and the upper bound on the degrees of (R_1, R_2, R_T). Then*

$$\mathcal{A}\left(uberdecisional_{\mathcal{G}}^{\mathcal{A}}\right) \leq 2\left(\frac{d_1(q_1+m_1)^2 + d_2(q_2+m_2)^2 + \max(d_1+d_2, d_T)(q_T+m_T+m_p)^2}{p}\right).$$

Where the experiment $\mathbf{Uberinteractive}_{\mathcal{G}}^{\mathcal{A}}$ is defined in Fig 3.8.

- $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{pp}_{\mathsf{ATMS}} \leftarrow \Sigma_1.\mathsf{Setup}(1^\lambda) \wedge \mathsf{pp}_{\mathsf{SPSEQ}} \leftarrow \Sigma_2.\mathsf{Setup}(1^\lambda) \wedge \mathsf{pp}_{\mathsf{SC}} \leftarrow \mathsf{SC.Setup}$, output $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ATMS}}, \mathsf{pp}_{\mathsf{SPSEQ}}, \mathsf{pp}_{\mathsf{SC}})$.
- $\mathsf{IKeyGen}(\mathsf{pp})$: Generate $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \Sigma_1.\mathsf{KeyGen}(\mathsf{pp})$, return $(i\mathsf{sk} = \mathsf{sk}, i\mathsf{vk} = \mathsf{vk})$ to an issuer i.
- $\mathsf{UKeyGen}(\mathsf{pp}, S)$: Run $((\tau, \mathbf{T}), \mathsf{aux}) \leftarrow \mathsf{GenIdxTag}(S)$, and return $(u\mathsf{sk} = \tau, u\mathsf{vk} = \mathbf{T})$ to u. Then, TA and u interact to computes $((\hat{\mathbf{C}}_i, \mathbf{C}_i)_{i\in[\ell]}) \leftarrow \mathsf{SC.Commit}_3(\mathsf{A}_i, \alpha, \mathbf{T})$, for all attribute sets.
- Issuance: The interaction between an issuer i and a user u for one attribute-set $\mathsf{A} \in \mathbb{Z}_p$ and $(\mathbf{C}, \hat{\mathbf{C}})$ acts as follows:
  - u hands over $(\mathbf{T}, (\mathbf{C}, \hat{\mathbf{C}}), \mathsf{aux}_i, \pi)$ to an issuer i, where π is zero knowledge proof the secret parts of the tag.
  - An issuer i checks that the proof is correct, then runs $\sigma \leftarrow \Sigma_1.\mathsf{Sign}(i\mathsf{sk}, \mathbf{T}, \mathsf{aux}_i, (\mathbf{C}, \hat{\mathbf{C}}))$, and outputs $(\mathsf{A}, \mathbf{T}, \sigma) = \mathsf{cred}_i$.
  - u takes $(i\mathsf{vk}, \mathsf{cred}_i)$ for $i \in [\ell]$, checks $\Sigma_1.\mathsf{Verify}(i\mathsf{vk}, \mathbf{T}, (\mathbf{C}_i, \hat{\mathbf{C}}_i), \sigma_i)_{i\in[\ell]} = 1$, and outputs $\{\mathsf{cred} = (\sigma_i, \tau), (\mathsf{A}_i, \mathbf{C}_i, \hat{\mathbf{C}}_i)_{i\in[\ell]}\}$.
- GenPolicies: Generate a key pair $(v\mathsf{sk}, vpk) \leftarrow \Sigma_2.\mathsf{KeyGen}(\mathsf{pp})$, run $\sigma_{2i} \leftarrow \Sigma_2.\mathsf{Sign}(v\mathsf{sk}, i\mathsf{vk})$ for $i \in I$, return $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$.
- Show: On input $\mathsf{cred} = \{(\sigma_i, u\mathsf{sk}, \mathsf{A}_i)_{i\in[\ell]}\}$, $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$, and $D \subseteq \mathsf{A}$ from $n \subseteq I$ issuers, u prepares a proof for D as:
  1. Run $(\sigma'_{2i}, \mathsf{avk}') \leftarrow \Sigma_2.\mathsf{ChangRep}(\mathbf{M}_i = \mathsf{vk}_i, \sigma_{2i}, \omega)_{i\in[n]}$ for $\omega \in \mathbb{Z}_p^*$.
  2. Run $\sigma \leftarrow \Sigma_1.\mathsf{AggrSign}(\mathbf{T}, (i\mathsf{vk}, (\mathbf{C}_i, \hat{\mathbf{C}}_i), \sigma_i))_{i\in[n]}$. Convert credentials and issuer keys $\mathsf{avk}' \leftarrow \Sigma_1.\mathsf{ConvertVK}(\mathsf{avk}, \omega)$ and $\sigma' \leftarrow \Sigma_1.\mathsf{ConvertSig}(\mathsf{avk}, (\mathbf{C}, \hat{\mathbf{C}}), \sigma, \mathbf{T}, \omega)$.
  3. Run $(\sigma', \mathbf{T}') \xleftarrow{\$} \Sigma_1.\mathsf{ChangRep}(\sigma, (\mathbf{M}_i, \mathbf{N}_i)_{i\in[n]}, \mathbf{T}, (\mu, \upsilon))$ for (μ, υ), where $(\mathbf{M}_i, \mathbf{N}_i) = (\mathbf{C}_i, \hat{\mathbf{C}}_i)$, and σ' is valid for $(\mathbf{C}'_i = \mathbf{C}_i^{\mu\upsilon}, \hat{\mathbf{C}}'_i = \hat{\mathbf{C}}_i^{\upsilon})_{i\in[n]}$. Create witnesses for attributes $W_j \leftarrow \mathsf{SC.OpenSubset}(\hat{C}_{1j}, A_j, O_j, d_j)$ for $j \wedge d_j \in D$. Aggregate witness $W \leftarrow \mathsf{SC.AggregateAcross}(\{\hat{C}_{1j}, d_j, W_j\}_{j\in[\ell]})$, randomize $W' \leftarrow W^{\mu\upsilon}$.
  4. Prove in zero knowledge that the user knows the secret key for the tag $\mathbf{T}'$, yielding π, send $(\sigma', W', \mathbf{T}', \sigma'_{2i}, \pi, \mathbb{M} = \{(\mathbf{C}'_i, \hat{\mathbf{C}}'_i)\})_{i\in[n]}$ to V.
- CredVerify: Output 1, if $\pi \wedge \Sigma_1.\mathsf{VerifyAggr}(\mathsf{avk}', \mathbf{T}', \mathbb{M}, \sigma') \wedge \Sigma_2.\mathsf{Verify}(v\mathsf{vk}, \mathbf{M}, \sigma'_2) \wedge \mathsf{SC.VerifySubset}(\mathbf{C}', D, W') = 1$, where $\mathbf{M} = \mathsf{avk}'$ is verified by $v\mathsf{vk}$.

Figure 3.7: Our IhMA scheme (Σ_1 and Σ_2 denote ATMS and SPSEQ [FHS19], respectively)

3.4.4 Additional Properties

We now discuss how additional features can be obtained via slight modifications of the so far presented approach.

Blind issuing credential for AtoSa. We note that our schemes can provide a blind issuing protocol in which a user can receive credentials on blind attributes using the

$\textbf{Uberinteractive}^{\mathcal{A}}_{\mathcal{G}}$	$\mathcal{O}(t \in \{1,2,T\}, R^0, R^1)$:
• $(g_1, g_2) \leftarrow \mathsf{BG}$; $g_T \leftarrow e(g_1, g_2)$ • $b \xleftarrow{\$} \{0,1\}$ • $\mathbf{x} = (x_1, \ldots, x_m) \xleftarrow{\$} \mathbb{Z}_p^m$ • $(\mathbf{R}_1^0, \mathbf{R}_2^0, \mathbf{R}_T^0, \mathbf{R}_1^1, \mathbf{R}_2^1, \mathbf{R}_T^1) := ([1], [1], [], [1], [1], [])$ • $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(g_1, g_2)$ • If $(\mathbf{R}_1^0, \mathbf{R}_2^0, \mathbf{R}_T^0, \mathbf{R}_1^1, \mathbf{R}_2^1, \mathbf{R}_T^1)$ is τ-trivial: • Return a random bit • Else $Return\big(b = b'\big)$	• $\forall b' \in \{0,1\} : \mathbf{R}_t^{b'} := \mathbf{R}_t^{b'} :: R^{b'}$ • $Return\big(g_t^{R^b(x_1,\ldots,x_n)}\big)$

Figure 3.8: Adaptative game for the uber assumption relatively to the bilinear group $\mathcal{G}$ and adversary $\mathcal{A}$.

two-party computation technique provided in PS and Coconut [SAB+19]. It works as follows:

- A user generates an El-Gamal key-pair $(d_1, D_1 = P_1^d)$; pick a random k and compute an El-Gamal encryption of m as below: $c = \mathsf{Enc}(m) = (a = P^k, b = P^{k \cdot d} \cdot (h')^m)$. Output $(\mathsf{aux}, h, D, c, \pi)$, where π and $h' = h^{\rho_1}$ is defined by: $\pi = \mathsf{ZKPoK}\{(d, m, k) : D = P^d \wedge c = (P^k, D^k \cdot (h')^m)\}$
- A signer checks the π is correct, and generates blind signatures as follows: $\delta = (a'_1 = a^{y_1} = P^{y_1 \cdot k}, b' = (h^{\rho_2})^{y_2} \cdot (h')^x \cdot b^{y_1} = P^{y_1 \cdot k \cdot d} \cdot (h')^{y_1 \cdot m + x} \cdot h^{y_2 \cdot \rho_2})$
- The user decrypts/unblinds signature $\delta = (a', b')$ and gets $h'^{y_1 \cdot m + x} \cdot (h^{\rho_2})^{y_2}$ as follows: $(a')^d = P^{y_1 \cdot k \cdot d}$ and $(h')^{m \cdot y_1 + x} \cdot (h^{\rho_2})^{y_2} = \frac{b'}{(a')^d}$.

As we showed in the IhMA, one can also hide the tag.

Multi-Message Signatures for AtoSa. One can extend this scheme to sign a message vector $\mathbf{m}$ rather than a single m by extending a verification key $\mathsf{vk}_i = (\hat{X}_i, \hat{Y}_{i1}, \ldots, \hat{Y}_{in})$ regarding the number of messages. A signer i can sign a vector $\mathbf{m} = (m_1, \ldots, m_n)$ as $h^{x_i + \sum y_i m_i}$ for $m_i \in \mathbf{m}$ (see [PS16a] for more details).

Non-transferable Credentials. Often it is desirable to prevent users from easily sharing their credentials with others. One common approach to deter such transfers is to leverage a valuable item, such as a secret key, which would also need to be shared if a credential were to be shared [CL01]. We note that schemes involve users proving their knowledge of tag's secrets that represents their identity. We can now use the canonical representative (ρ_1/ρ_2) of the respective tag class as the valuable secret. Then note that when sharing a credential

even with a re-randomized tag $(\upsilon\rho_1, \upsilon\rho_2)$, one can extract the canonical representative and thus also shares the valuable secret. While this feature is important for ACs, however, it is not always easily achievable in all AC systems. In fact, achieving this feature can be quite challenging in some cases, especially in self-binding approaches such as SPSEQ or [CLPK22].

Proving Knowledge of AtoSa Signature. One can achieve the proving knowledge of a signature exactly similar to PS. Assume AtoSa signature is $\sigma = (h, s)$, we select random $r, t \leftarrow \mathbb{Z}_p$ and compute $\sigma' \leftarrow (h^r, (s \cdot h^t)^r)$. We send it to the verifier and carries out a zero-knowledge proof of knowledge π (a Schnorr proof) of (m, ρ_1, ρ_2) and t for the signature on a single message: $\mathsf{ZKPOK}\{(m, \rho_1, \rho_2, t) : e(h', \hat{X}) \cdot e(h', Y_1)^m \cdot e(h', Y_1)^{\rho_1} \cdot e(h', Y_2)^{\rho_2} \cdot e(h', \hat{P})^t = e(s', \hat{P})\}$. It can be extended straightforwardly for multi messages (see [PS16a] for more details).

3.5 Implementation and Evaluation

In the following we present our evaluation based on a Python library in which we implement our primitives ATMS and AtoSa as well as our IhMA protocols (Fig. 3.7 and Fig. 3.6). Our implementation is based upon the `bplib` library[1] and petlib [2] with `OpenSSL` bindings[3]. We use the popular pairing friendly curve BN256 which provides efficient type 3 bilinear groups at a security level of around 100 bits. Our measurements have been performed on an Intel Core i5-6200U CPU at 2.30 GHz, 16 GB RAM running Ubuntu 20.04.3.

Benchmark of Primitives. Table 3.2 shows the mean of the execution time of each algorithm over 500 runs such that AggrSign and VerifyAggr are computed assuming two signers ($n = 2$); the other algorithms are independent of n. ChR/Rnd stands for ChangRep and signature randomization (RandSign) for the ATMS and AtoSa, respectively. PC stands for Pre-Computation, and in ATMS it includes converting messages to the $\mathcal{M}^H_{\mathsf{TDH}}$ message space and generating tags. While in AtoSa, PC includes generating tags and aux using Pedersen commitment, but note that one could also use a hash based commitment instead. We can observe that signing is faster than verifying the signature – due to the pairing operation in the latter. Moreover, verification of ATMS is slower than AtoSa because of additional pairing operations that are needed to check if messages are in $\mathcal{M}^H_{\mathsf{TDH}}$. We

Table 3.2: Running times for ATMS and AtoSa (ms)

	PC	Sign	Verify	Convert	ChR/Rnd	AggrSign	VerifyAggr
AtoSa	6	2,5	8,4	4	2,7	0.005	9
ATMS	8.6	3	33	5,4	7,4	0.01	72

[1] https://github.com/gdanezis/bplib
[2] https://github.com/gdanezis/petlib
[3] https://github.com/dfaranha/OpenPairing

increase the number (n) of signers from 2 to 10 and show the running time in Fig. 3.9. Since aggregation is almost free (for $n = 10$ is 0.05 ms), we omit it. We should also note that the result are stated without considering VerifyAux algorithm.

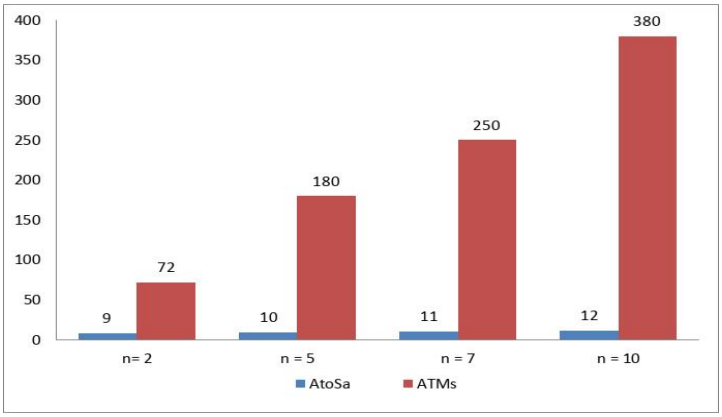


Figure 3.9: Running times of VerifyAggr in ATMS & AtoSa (ms)

IhMA **Benchmarks**. IhMA is based upon Schnorr-style discrete logarithm ZKPOK. Our library supports Damgård's technique [CDM00] for obtaining malicious-verifier interactive zero-knowledge proofs of knowledge during the showing and also ZKPoK obtained via the Fiat-Shamir heuristic. We interpret signers as issuers here and also show n as a number of issuers involved in Showing. For example, $n = 2$ means showing two credentials from 2 different issuers.

Issuing. This protocol does not depend on n, and results are as follows: 1) For IhMA based on AtoSa, including generation of signature, tag, user keys, and aux, it takes 8 ms. 2) For IhMA based on AtoSa, including generation of tag and encoding messages to $\mathcal{M}^{H}_{\mathsf{TDH}}$, with two attributes in each credential it takes 10 ms.

Showing. Fig. 3.11 shows the runtime of showing for IhMA based on AtoSa. In this experiment, we increase the number of issuers n from 2 to 10 and assume that all attributes are disclosed during verification (the worst-case scenario). Each issuer issues only one attribute, giving a total of n attributes. Fig. 3.10 shows the time for showing a credentials of IhMA based on ATMS. Here, we have a different setting; we can encode a set of attributes in a credential as we use set commitments. For our evaluation, we have the following parameters: n represents the number of the issuer, t the number of attributes in each set (each credential issued), $d < t$ is the number of disclosed attributes from each attribute set A in the respective commitment C. Here we increase n from 2 to 10, set $t = 2$, and $d = 1$. The total disclosed attributes length $|D| = d \cdot n$ and the total attribute $|A| = n \cdot t$ range from 2 to 10 and 4 to 20, respectively.

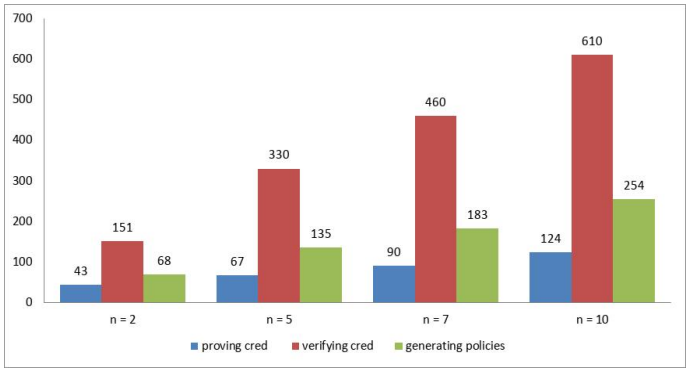


Figure 3.10: Running times of $\mathsf{IhMA}_{\mathsf{ATMS}}$

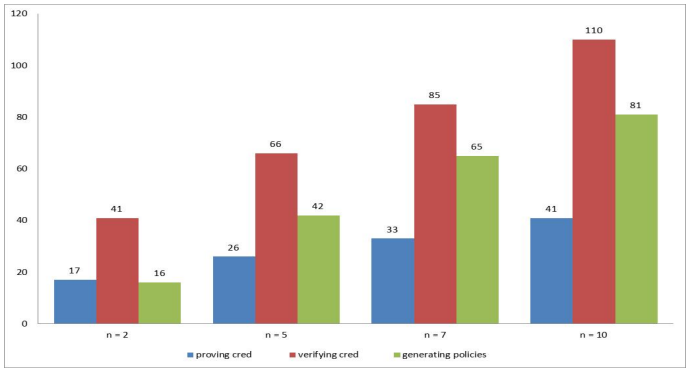


Figure 3.11: Running times of $\mathsf{IhMA}_{\mathsf{AtoSa}}$

3.5.1 Bandwidth Analysis of our IhMA Schemes

We present a concrete comparison of schemes in Table 3.3. We denote the size of zero knowledge proof of the tag as ZKPOK which as we showed in Section 5, the statements are simple and can be done efficiently with simple Schnorr proofs (the statements are presented in Section 3.4.3) .

Table 3.3: Communication complexity of our IhMA schemes (N: total issuers and K: issuers in showing).

	$\mathsf{IhMA}_{\mathsf{AtoSa}}$	$\mathsf{IhMA}_{\mathsf{ATMS}}$
$\lvert\mathbf{cred}\rvert$	$2N\mathbb{G}_1 + 2\mathbb{Z}_p$	$3N\mathbb{G}_1 + 2\mathbb{Z}_p$
$\lvert\mathbf{show}\rvert$	$4\mathbb{G}_1 + 4K\mathbb{G}_2 + 2K\mathbb{G}_1 + \mathsf{ZKPOK}$	$6K\mathbb{G}_2 + 6\mathbb{G}_1 + 2K\mathbb{G}_1 + \mathsf{ZKPOK}$

* We present the scheme in a way that supports ad-hoc attribute/issuer aggregation, but for fixed signatures, a constant size credential is achievable.

3.6 Summary

This chapter introduces the Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA). **MA** means proving possession of attributes from multiple independent credential issuers requires the presentation of independent credentials. Meanwhile, **Ih** means verifying a user's credential does not require disclosing multiple issuers' public keys.

Our proposed solution involves the development of two new signature primitives with versatile randomization features which are independent of interest: 1) Aggregate Signatures with Randomizable Tags and Public Keys (AtoSa) and 2) Aggregate Mercurial Signatures (ATMS), which extend the functionality of AtoSa to support the randomization of messages additionally.

We formalize all notations and provide rigorous security definitions for our proposed primitives. We present provably secure and efficient instantiations of the two primitives and corresponding IhMA systems. Finally, we provide benchmarks based on implementation to demonstrate the practical efficiency of our constructions.

4 Delegatable Anonymous Credentials

In this chapter we present a novel DAC scheme that supports attributes, provides anonymity for delegations, allows the delegators to restrict further delegations, and also comes with an efficient construction. Our approach builds on a new primitive that we call structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC). The high-level idea is to use a special signature scheme that can sign vectors of set commitments, where signatures can be extended by additional set commitments. Signatures additionally include a user's public key, which can be switched. This allows us to efficiently realize delegation in the DAC. Similar to conventional SPSEQ, the signatures and messages can be publicly randomized and thus allow unlinkable delegation and showings in the DAC system. We present further optimizations such as cross-set commitment aggregation that, in combination, enable efficient selective showing of attributes in the DAC without using costly zero-knowledge proofs. We present an efficient instantiation that is proven to be secure in the generic group model and finally demonstrate the practical efficiency of our DAC by presenting performance benchmarks based on an implementation.

4.1 High Level Idea of Our Approach

On a very high level, our approach to construct DAC takes inspiration from the anonymous credentials in [FHS19] as well as the approach based on DMS in [BB18]. Importantly, in contrast to the latter and similar to DACs based on mercurial signatures [CL19, CL21], it however avoids the use of NIZK for complex statements.

The idea in our DAC, omitting some details for the sake of brevity, is that in a hierarchy of delegations the root authority issues a SPSEQ-UC signature on a commitment.[1] The commitment carries the attributes for the first delegatee and the public key to which the signature is tied is the one of the delegatee. The delegatee, if provided with a corresponding update key for this signature by the delegator, can then perform further delegations. This update key allows to further extend the commitment vector (and thus add attributes) and thus delegating a credential for the next level in the delegation hierarchy. Again the public key of the delegatee, now playing the role of the delegator, is switched to the one of the next delegatee. Due to the privacy properties of the SPSEQ-UC, a signature resulting from extending the vector looks like a fresh signature (derivation-privacy) and one resulting from switching a user key also looks like a fresh signatures (conversion-privacy). This ensures

[1] Technically to guarantee anonymity we require two commitments, where the first one is a dummy commitment and not assigned any or simply some fixed attributes.

that in the DAC, delegations cannot be tracked and all credentials in a delegation chain are indistinguishable. This process keeps on going until the end of the delegation chain is reached (if no further delegations are allowed, then no update key is provided). One issue that is worth mentioning is that every delegator can control how far delegations can go by further restricting the update key and a delegator can also restrict the possibility to show attributes from a certain level in the hierarchy (which corresponds to a commitment in the commitment vector) by not providing the opening of the commitment to the delegatee.

Now showing a credential simply amounts to adapting the signature to a re-randomized signature for a re-randomized commitment vector and providing subset openings of the respective commitments. Due to the origin-hiding property of the SPSEQ-UC this results in an unlinkable showing. As we show in Section 4.5 we can realize a cross-commitment aggregation technique to make the opening of multiple commitments compact.

4.2 Practical Example Application

While DAC can be beneficial in many applications of ACs, we want to discuss a particular application of DAC as a motivation for future practical deployments.

ISO 18013-5, a recently published standard, establishes the standardization of international mobile driving licenses (mDLs). This mDL deployment serves as a widely adopted example of real-world credentials on a global scale and represents one of the initial international standards for digital credentials with built-in privacy protections. This stands in contrast to currently deployed NFC passports, which lack any provisions to address privacy-sensitive concerns.

In contrast to traditional paper/plastic card implementations, digital versions of driving licenses employ measures to prevent forging by utilizing signatures from trusted issuing authorities (IAs). These measures, known as "protection against forgery," rely on signatures over Mobile Security Objects (MSOs). These MSOs consist of hash commitments representing the credential attributes and are referred to as "issuer data authentication" in the standard. Additionally, they incorporate "protection against cloning," which is based on message authentication codes (MACs) applied during individual sessions (known as "mdoc authentication").

To ensure "protection against unauthorized access," consent dialogs are implemented on the holder's device, allowing users to choose which attributes to disclose during a presentation. Moreover, to establish unlinkability between different showings (as long as no identifying attributes are included), multiple MSOs can be provisioned [2]. Each presentation employs unique attribute hash sets and signature values, enhancing the level of privacy and security. In short, the current mDL standard [ISO] offers the following features related to Attribute Credentials (ACs): 1) Selective disclosure of attributes. 2)

[2]Note that the decision to provide multiple MSOs for a provisioned mDL lies with the issuing authority. If the IA does not provide them, holders of mDLs can be trivially linked between different showings based on the MSO.

Unlinkability against verifiers. 3) Capability to include different showings to the same verifier (assuming sufficient availability of MSOs for single-use showings). 4) Unforgeability, relying on the assumption of an unforgeable signature scheme. 5) Compactness in terms of static credential/signature data and showing sizes, along with sufficient efficiency for implementation on current smart cards and phones.

However, it currently does not address *strong anonymity* against issuers themselves or the *delegation* of capabilities to issuing authorities[3].

Delegation holds significant importance in the context of global standards and practical implementations of real-world credentials. It is unlikely that a single issuer would be universally trusted for driving licenses across the world. In many larger countries, this authority is already delegated to smaller organizational units, like US state DMVs, creating delegation chains with at least two levels. However, the current standard assumes that all verifiers possess a consolidated list of all issuer public keys. This information is then used to verify presented Mobile Security Objects (MSOs) and perform "issuer data authentication" for disclosed attributes. Consequently, the respective issuer becomes explicitly known to the verifier, even if no specific credential attribute would have otherwise disclosed this information. This poses several problems, such as potentially revealing the holder's approximate home location, citizenship, or other internal aspects of the issuing organization. For instance, if separate sub-organizations issue credentials for legal aliens, asylum seekers, or refugees, this information can be misused for discriminatory or tracking purposes.

Our proposed solution has the potential to enhance future mDLs or similar real-world credentials significantly. It stands out by explicitly providing robust anonymity protection against issuers and verifiers, as well as enabling delegation with the option for top-level issuers to determine the depth of their organizational hierarchy. Unlike some alternative proposals, our solution maintains the capability for selective attribute disclosure while ensuring efficiency and compactness in all communication and processing. Particularly in widely applicable scenarios like age verification, one of the main applications of mDLs, our solution can prevent unintentional leaks of personal data through the delegation hierarchy, leading to substantial improvements in user privacy and the prevention of potential discrimination.

4.3 Comparison with Previous Work

In Table 6.3.5, we provide a comparison of our approach with other existing efficient DAC schemes in the literature [BB18, CDD17, CL19, CL21]. We compare our DAC with these schemes in terms of the following criteria: **Attr** indicates whether credentials include attributes that can at least be selectively revealed. We use $\approx$ to indicate that [CL21] supports attributes, but as all attributes always need to be revealed it does not support

[3]The current mDL standard also does not cover efficient revocation of credentials, but instead, it recommends using short-lived MSO signatures as a mitigation. We consider revocation to be out of scope for this discussion and, therefore, do not point it out as another shortcoming.

selective disclosure. **Expr** represents the expressiveness of the supported showing policies, where R stands for arbitrary computable relations over attributes and S denotes the selective disclosure of a subset of attributes. We note that by avoiding NIZK proofs for complex statements, it seems necessary to be restricted to selective disclosure (S), which is however sufficient for most practical applications, e.g., ISO 18013-5 discussed in Section 4.2. **Rest**

Table 4.1: Comparison of practical DAC schemes (L: Delegation chain depth; n: Attributes; u: Undisclosed attributes).

Scheme	**Attr**	**Expr**	**Rest**	**SAnon**	**|Cred|**	**|Show|**
BB [BB18]	✓	S/R	≈	◐†	$O(1)$	$O(u)$
CDD [CDD17]	✓	S/R	×	◐$^{\dagger,\clubsuit}$	$O(nL)$	$O(uL)$
CL [CL21]	≈	×	×	◐*	$O(nL)$	$O(uL)$
Ours	✓	S	✓	●‡	$O(1)$	$O(L)$

† Requires a trusted setup and have a trapdoor associated to their parameters.
$^{\clubsuit}$ It does not support an anonymous delegation phase.
‡ We consider a malicious CA key and all delegators keys can be exposed.
* It also allows an adversarial CA but no delegators's keys leak.

indicates whether it is possible to apply a restriction on the delegator's power during the delegation. Here, our scheme allows such restrictions in which a (superior) delegator can decide i) how many additional levels of delegation can be made, ii) to make all attributes of selected levels "unshowable" by not providing the opening of the respective commitments, and, iii) how many attributes in each level (commitment) can be delegated by determining (potentially removing) key components in $\mathsf{uk}_{k'}$. Here, we note that BB [BB18] provides a type of restriction on the attributes such that delegators can prevent changing some attributes during delegation. However, one still can use these attributes in showings of a credential. Also, BB does not have a delegation-level concept and thus one cannot control and restrict the delegation-level number (power). (**SAnon**) refers to strong anonymity guarantees, meaning that no one can trace or learn information about the user's identity or anything beyond what they suppose to show during both the issuing/delegation and showing of credentials. Moreover, anonymity holds without relying on a trusted setup (and thus a potential trapdoor breaking anonymity) as well as under a malicious key generation by a (corrupted) root authority, and user's key can leak [4]. Here ● means that the scheme satisfies all conditions, and ◐ means that it does not provide one or more of them.

With **|Cred|** we denote the size of the credential. L indicates the length of the delegation chain. As it turns out, BB [BB18] (2 group elements) and our scheme (5 group elements) provide constant-size credentials. But as already mentioned, our scheme provides a simpler construction by *avoiding potentially costly linear-sized (in the number of attributes) zero-*

[4] We note that CL and our model require a credential cred_b of the anonymity challenge to be on a delegation path from a (corrupted) root credential where all delegations have been performed honestly. However, we additionally allow the adversary to access the user corruption oracle in which we reveal the (delegators) user's secret keys to the adversary. CL cannot support this type of corruption as then the anonymity of their construction breaks down. This makes our model stronger than the one of CL.

knowledge proofs. With $|\mathbf{Show}|$ we denote the size of the credential showing. Our showing is efficient as it needs only a constant number of group elements (5 elements) and the commitment vector with the size of delegation L. BB does not have the concept of levels and needs to send the signature (2 elements) and the elements of proving knowledge of undisclosed attributes ($|\mathsf{ZKPoK}| = u$). Note that for practical use-cases, we can typically assume $L < u$. For other schemes, this cost is much higher as their credentials grow linearly in the number of attributes and L. Note that if there are a large number of attributes to be issued, they can be split into two sets and embedded in two credentials.

Comparing efficiency with related work. We do not provide a comparison with CL as it does not support selective showing of attributes (and there is also no implementation available). CDD is the most efficient scheme and the only one from Table 6.3.5 that is fully specified. We will compare our implementation to the implementation of CDD [CDD17] that has recently been provided by [BCET21b] in Section 4.7. Moreover, we provide a more comprehensive theoretical comparison with CDD, which due to the lack of space is deferred to 4.7.1. For BB [BB18], unfortunately, only the underlying signature scheme based on Pointcheval-Sanders signatures [PS16a] is specified. But the remaining parts of their generic constructions are not detailed, making concrete performance estimates hard. However, since their credential showing must conceal the DMS signature and prove the verification relation of the DMS (resulting in a size linear in the number of undisclosed attributes), this imposes a rather complex NIZK statement.

4.4 SPSEQ on Updatable Commitments

As our primary building block, we introduce equivalence-class signatures on updatable commitments called (SPSEQ-UC). It can be viewed as a variant of SPSEQ with the following modifications: *i*) It considers the message space as vectors of randomizable set commitments, i.e., one can adapt a signature on a commitment vector to a randomized version of the signed commitments. This means that equivalence classes are defined on vectors of commitments. *ii*) SPSEQ-UC not only considers signing representatives of classes of a single projective equivalence relation $\mathcal{R}$, but a family of relations. This is, as we allow to extend signed vectors by additional commitments. Thus we consider a family of such relations $\mathbb{I}\mathcal{R}^\ell$ such that $\mathcal{R}^k \in \mathbb{I}\mathcal{R}^\ell$ for any $1 \leq k \leq \ell$. More technically, in SPSEQ-UC signing of a commitment vector of length k also produces an update key $\mathsf{uk}_{k'}$ corresponding to an integer k' with $k \leq k' \leq \ell$. Given the update key $\mathsf{uk}_{k'}$, in addition to adapting a signature on a commitment vector $\mathbf{C}$ in class $[\mathbf{C}]_{\mathcal{R}^k}$ to another representative of the given class, one also can update a commitment vector $\mathbf{C}$ (i.e., extending it) to a vector $\mathbf{C}'$ being in a class $[\mathbf{C}']_{\mathcal{R}^{k'}}$ of a new equivalence relation. Then one can adapt the signature accordingly to the updated commitment vector. Finally, *iii*) in SPSEQ-UC a signature is bound to a user public key. The signer produces a signature bound to a user public key, and this can be adapted into another valid signature for a new user public key by anyone knowing the old user secret key.

4.4.1 Formal Definitions

We recall that in an SPSEQ scheme, one can sign vectors of group elements and it is possible to jointly randomize messages and signatures in public. The messages space consists of representatives of projective equivalence classes defined on one source group of a bilinear group, i.e, $(\mathbb{G}_1^*)^\ell$ (for some fixed $\ell > 1$), and randomization of a message represents a change to another representative in the signed class. In case of SPSEQ-UC the message space consists of a vector of group elements representing set commitments (a commitment vector) from $(\mathbb{G}_1^*)^\ell$. As mentioned above since we require updating, i.e., extending, the commitment vector, in contrast to SPS-EQ, we consider a family of equivalence relations $\mathbb{R}^\ell$. Thus, for any k with $1 < k \leq \ell$, we can define the following equivalence relation $\mathcal{R}^k \in \mathbb{R}^\ell$ and the equivalence class $[\mathbf{C}]_{\mathcal{R}^k}$ of a set commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$. More concretely, for a fixed bilinear group BG and (k, ℓ), we define $\mathcal{R}^k \in \mathbb{R}^\ell$ as follows:

$$\mathcal{R}^k = \left\{ (\mathbf{C}, \mathbf{C}') \in (\mathbb{G}_1^*)^k \times (\mathbb{G}_1^*)^k \Leftrightarrow \exists \mu \in \mathbb{Z}_p^* : \mathbf{C}' = \mathbf{C}^\mu \right\}.$$

Now we are ready to present the definition.

Definition 41 (SPSEQ-UC scheme)**.** *A* SPSEQ-UC *scheme for a set commitment scheme* SC *and a parameterized family of equivalence relations* $\mathbb{R}^\ell$ *consists of the following PPT algorithms:*

$\mathsf{PPGen}(1^\lambda, 1^t, 1^\ell) \to (\mathsf{pp})$: On input the security parameter λ and an upper bound t for the cardinality of committed sets and a length parameter $\ell > 1$, this probabilistic algorithm outputs the public parameters pp. The message set space S_{SC} is well-defined from pp. pp will be an implicit input to all algorithms.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: On input the public parameters pp, this probabilistic algorithm outputs a verification and signing key pair $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{UKeyGen}(\mathsf{pp}) \to (\mathsf{sk}_u, \mathsf{pk}_u)$: On input the public parameters pp, outputs a key pair $(\mathsf{sk}_u, \mathsf{pk}_u)$ for a user u.

$\mathsf{RndmzC}(\mathbf{C}, \mathbf{O}, \mu) \to (\mathbf{C}', \mathbf{O}')$: Takes a commitment vector $\mathbf{C}$ of size $1 < k \leq \ell$, corresponding openings $\mathbf{O}$ and randomness μ. It runs $(C_i', O_i') \leftarrow \mathsf{SC.RndmzC}(C_i, O_i, \mu)$ for all $i \in [k]$ and outputs a new representative of the set commitment vector $\mathbf{C}' \in [\mathbf{C}]_{\mathcal{R}^k}$ and corresponding openings $\mathbf{O}'$.

$\mathsf{Sign}(\mathsf{sk}, \mathbf{M}, k', \mathsf{pk}_u; \rho) \to (\sigma, (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'})$: This probabilistic algorithm takes as input a signing key sk, a vector of set messages $\mathbf{M} = (M_1, \ldots, M_k)$, an index k' with $k \leq k' \leq \ell$, a user public key pk_u and a vector of randomness ρ. It computes $(C_j, O_j)_{j \in [k]} \leftarrow \mathsf{SC.Commit}(M_j; \rho_j)$ for all $j \in [k]$, sets $\mathbf{C} = (C_1, \ldots, C_k)$ and $\mathbf{O} = (O_1, \ldots, O_k)$. It outputs a signature $(\sigma, \mathbf{C})$ for pk_u, and also an update key $\mathsf{uk}_{k'}$ in case $k' \neq \ell$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) \rightarrow 0/1$: On input a verification key vk, a user public key pk_u, a commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$, the purported signature σ, and a pair (subset/witness form set commitments) $(\mathbf{T}, \mathbf{U})$, it outputs 0 if any of the following checks fail and 1 otherwise:

- Check whether σ is a valid signature for $(\mathbf{C}, \mathsf{pk}_u)$.
- For all $(T_i, U_i) \in (\mathbf{T}, \mathbf{U})$: if $U_i = W_i$ check

  $1 \stackrel{?}{=} \mathsf{SC.VerifySubset}(C_i, T_i, W_i)$. Else if $U_i = O_i$, check $1 \stackrel{?}{=} \mathsf{SC.Open}(C_i, T_i, O_i)$.

$\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) \rightarrow 0/1$: On input a verification key vk, an update key $\mathsf{uk}_{k'}$, an integer k' and a signature σ, this update key verification algorithm outputs 0 or 1.

$\mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi) \rightarrow \mathsf{pk}'_u$: On input a user public key pk_u and randomness ψ, χ, this public key randomization algorithm outputs the randomized public key pk'_u.

$\mathsf{ChangRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi) \rightarrow (\sigma', (\mathbf{C}', \mathbf{O}'), (\mathsf{uk}'_{k'} \text{ or } \perp), \mathsf{pk}'_u, \chi)$: This algorithm takes as input the user public key pk_u, a commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$ in equivalence class $[\mathbf{C}]_{\mathcal{R}^k}$ and corresponding openings $\mathbf{O}$, a signature σ for $\mathbf{C}$, randomness ψ, μ and optionally an update key $\mathsf{uk}_{k'}$. It returns an updated signature σ' for a new commitment vector and corresponding openings $(\mathbf{C}', \mathbf{O}') \leftarrow \mathsf{RndmzC}(\mathbf{C}, \mathbf{O}, \mu)$ such that $\mathbf{C}' \in [\mathbf{C}]_{\mathcal{R}^k}$ as well as a randomized user public key $\mathsf{pk}'_u \leftarrow \mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi)$ for uniform randomness χ. In case that $\mathsf{uk}_{k'} \neq \perp$, it additionally outputs a randomized update key $\mathsf{uk}'_{k'}$.

$\mathsf{ChangeRel}(M_l, \sigma, \mathbf{C}, \mathsf{uk}_{k'}, k'') \rightarrow (\sigma', (\mathbf{C}', O_l), \mathsf{uk}_{k''})$: On input a message set $M_l \subset S_{\mathsf{SC}}$ for $l = k + 1 \in [k']$, a signature σ for a vector of commitments representative $\mathbf{C} = (C_1, \ldots, C_k)$ of equivalence class $[\mathbf{C}]_{\mathcal{R}^k}$, an updatable key $\mathsf{uk}_{k'}$, and an index $k'' \leq k'$. This algorithm adapts a signature σ' for a new commitment vector $\mathbf{C}' = (\mathbf{C}, C_l)$ of equivalence class $[\mathbf{C}']_{\mathcal{R}^l}$, where C_l is a set commitment for M_l with the related opening information O_l. Also, for $k'' \in [l + 1, k']$, updates the updatable key for the range $[l + 1, k'']$ into $\mathsf{uk}_{k''}$.

$\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \rightarrow (\sigma_{\text{orph}})$: It is an algorithm run by a user who wants to delegate a signature σ. It takes as input the public verification key vk, a secret key sk_u and the signature σ. It outputs an orphan signature σ_{orph}.

$\mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'}, \sigma_{\text{orph}}) \rightarrow \sigma'$: It is an algorithm run by a user who receives a delegatable signature. It takes as input the verification key vk, a secret key $\mathsf{sk}_{u'}$, an orphan signature σ_{orph}. It outputs a new signature σ' for $\mathsf{pk}_{u'}$.

For simplicity, when we write $\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ we mean $[\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \leftrightarrow \mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'})] \rightarrow \sigma'$, where σ' is a valid signature.

We note that $\mathsf{UKVerify}$ is an algorithm that checks whether the update key is formed correctly. This is required in the DAC construction (cf. Section 4.6.2) to verify whether a delegation is valid. Moreover, it helps in the definition of the privacy properties below.

4.4.2 Security Definitions

Similar to conventional signatures, a SPSEQ-UC scheme needs to be correct and unforgeable. And similar to SPSEQ, we need additional properties covering the distribution of adapted signatures.

Correctness. We require that honest signatures verify as expected. Moreover, algorithms $\mathsf{ConvertSig}, \mathsf{ChangRep}$ and $\mathsf{ChangeRel}$ need to output valid signatures for the respective parameters. We provide a formal correctness definition as follows:

Definition 42 (Correctness)**.** *A* SPSEQ-UC *scheme for a set commitment scheme* SC *and a parameterized family of equivalence relations* $\mathbb{R}^\ell$ *for all* $\ell > 1$*, is correct if it satisfies the following conditions for all* t, λ*,* k, k' *with* $k \leq k' \leq \ell$*, for all* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*,* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$*,* $\mathsf{pk}_u \in \mathsf{UKeyGen}(\mathsf{pp})$*, all* $\mathbf{M}$*,* ρ*,* $\mathbf{T} \subseteq \mathbf{M}$*, all* $(\sigma, (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'}) \in \mathsf{Sign}(\mathsf{sk}, \mathbf{M}, k', \mathsf{pk}_u; \rho)$*, any* $\mathbf{U}$ *with* $1 = \mathsf{SC.VerifySubset}(C_j, T_j, U_j)_{j \in k}$ *(for* U_j *being a subset opening) and* $1 = \mathsf{SC.Open}(C_i, T_i, U_i)_{j \in k}$ *(for* U_j *being an opening):*

Verification*: We have that:*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1.$$

Change of set commitments representative: *For all* (μ, ψ)*,* $(\sigma', \mathsf{uk}'_{k'}, \chi) \in \mathsf{ChangRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi)$*, all* $(\mathbf{C}', \mathbf{O}') \leftarrow \mathsf{RndmzC}(\mathbf{C}, \mathbf{O}, \mu)$*,* $\mathsf{pk}'_u \leftarrow \mathsf{RndmzPK}(\mathsf{pk}, \psi, \chi)$ *and any* $\mathbf{U}'$ *s.t. either* $U'_j \leftarrow \mathsf{SC.OpenSubset}(C'_j, O'_j, T_j)$ *or* $U'_j = O'_j$ *we have:*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}'_u, \mathbf{C}', \sigma', (\mathbf{T}, \mathbf{U}')) = 1 \text{ and } \mathbf{C}' \in [\mathbf{C}]_{\mathcal{R}^k}.$$

Signature conversion*: For all* $(\mathsf{pk}_{u'}, \mathsf{sk}_{u'}) \in \mathsf{UKeyGen}(\mathsf{pp})$*,* $\sigma' \leftarrow \mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ *it holds that*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_{u'}, \mathbf{C}, \sigma', (\mathbf{T}, \mathbf{U})) = 1.$$

Change of set commitments relation*: For any iterative application of* $\mathsf{pk}_{u_l} \in \mathsf{UKeyGen}(\mathsf{pp})$*,* $(\mathsf{uk}_{k''_l}, \sigma'_l) \leftarrow \mathsf{ChangeRel}(M_l, \sigma_{l-1}, \mathbf{C}, \mathsf{uk}_{k'_{l-1}}, k'')$ *for any* M_l*, with* $\mathbf{C}' = (\mathbf{C}, C_l \in \mathsf{SC.Commit}(M_l))$ *and* $\mathbf{M}' = (\mathbf{M}, M_l)$*, any* $\mathbf{U}'$ *s.t.* $U'_j \leftarrow \mathsf{SC.OpenSubset}(C'_j, O'_j, T_j)_{j \in l} \vee U'_j = O'_j$ *with* $l < k'' \leq k'$ *and* $\mathbf{T}' \subseteq \mathbf{M}'$*, we have:*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_{u_l}, \mathbf{C}', \sigma'_l, (\mathbf{T}', \mathbf{U}')) = 1$$

whenever $l \in [k+1, k']$ *and also we have* $[\mathbf{C}']_{\mathcal{R}^l}$*.*

Unforgeability. Here, we consider an adversary that has access to signatures for message set vectors of its choice, controls randomness of commitments and is allowed to create as well as corrupt user keys. We require that it cannot come up with a signature on a commitment vector that opens to non-signed message (sub-)sets. Here we need to consider that the adversary is allowed to extend commitment vectors. In addition, the adversary

needs to specify the used user secret and public key $(\mathsf{sk}^*, \mathsf{pk}^*)$ for the forgery, which is required to make this concept useful in the application to DACs. Looking ahead, since the output of ChangeRel and ChangRep is distributed identical to Sign, we do not need to provide access to such oracles as it can be done by the adversary on its own. To detect that signatures derived with $\mathsf{uk}_{k'}$ are obtained from ChangeRel or are generated freshly in the Sign oracle, we define the following relation $\mathcal{R}_{k'}$. Consequently, signatures that can be legally derived using ConvertSig and ChangRep are not considered as forgeries.

Definition 43. *Let k, ℓ be integers. For any $\ell \geq k' > k$, we define the relation $\mathcal{R}_{k'}$ for two vectors $\mathbf{M} = (M_1, \ldots, M_k)$ and $\mathbf{M}^* = (M_1^*, \ldots, M_{k'}^*)$ as follows:*

$$(\mathbf{M}, \mathbf{M}^*) \in \mathcal{R}_{k'} \iff \forall i \leq k : M_i^* \subseteq M_i$$

$\underline{\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t)}$:

- $Q := \emptyset; \mathcal{UL} := \emptyset, \mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$
- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- $((\mathsf{sk}_u^*, \mathsf{pk}_u^*)(\mathbf{C}^*, \mathbf{T}^*, \mathbf{U}^*), \sigma^*) \leftarrow \mathcal{A}^{<\mathcal{O}>}(\mathsf{vk}, \mathsf{pp})$ **return:**

$$\begin{pmatrix} \forall(\mathsf{pk}_u, \mathsf{sk}_u) \notin \mathcal{UL}, \forall(\mathbf{M}, k', \mathsf{pk}_u) \in Q : \\ (\mathbf{M}, \mathbf{T}^*) \notin \mathcal{R}_{k'} \wedge (\mathsf{sk}_u^*, \mathsf{pk}_u^*) \in \mathsf{UKeyGen}(\mathsf{pp}) \\ \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u^*, \mathbf{C}^*, \sigma^*, (\mathbf{T}^*, \mathbf{U}^*)) = 1 \end{pmatrix}$$

$\underline{\mathcal{O}^{\mathsf{Create}}(i)}$:

- $(\mathsf{pk}_u, \mathsf{sk}_u) \leftarrow \mathsf{KeyGen}()$
- $\mathcal{UL} \leftarrow \mathcal{UL} \cup \{(i, \mathsf{pk}_u, \mathsf{sk}_u)\}$ **return** pk_u

$\underline{\mathcal{O}^{\mathsf{Sign}}(\mathbf{M}, k', \mathsf{pk}_u, \rho)}$:

- If $\ell \geq k' \geq k$:
- Then $(\sigma, (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'}) \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathbf{M}, k', \mathsf{pk}_u; \rho)$
- $Q = Q \cup \{(\mathbf{M}, k', \mathsf{pk}_u)\}$
- **return** $((\mathbf{C}, \mathbf{O}), \sigma, \mathsf{uk}_{k'})$
- Else **return** $\perp$

$\underline{\mathcal{O}^{\mathsf{Corrupt}}(i)}$:

- If $\exists i \in \mathcal{UL}$ such that $(\mathsf{pk}_u, \mathsf{sk}_u) \in \mathcal{UL}$
- Then delete the item from the list and **return** $(\mathsf{sk}_u, \mathsf{pk}_u)$
- Else **return** $\perp$

Figure 4.1: Experiment $\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t)$

Formally for unforgeability we require the following:

Definition 44 (Unforgeability)**.** *A* SPSEQ-UC *scheme is unforgeable if, for all $(\lambda, t) \in \mathbb{N}$, and $\ell > 1$, for any* PPT *adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that $\Pr[\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t) = 1] \leq \epsilon(\lambda)$, where the experiment $\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t)$ is defined in Fig 4.1 and Q is the set of queries that $\mathcal{A}$ has issued to the signing oracle.*

Note that unforgeability allows the adversary to either output a full opening $(U_i^* = O_i)$ or subset opening $(U_i^* = W_i)$ for each commitment in the commitment vector. This is also used to make it useful in the application to DAC.

Privacy notions. Subsequently, we define three privacy properties that are similar in vein to origin-hiding and signature adaption from previous works on SPSEQ and mercurial signatures [CL19,FHS19]. However, since SPSEQ-UC supports more functionality we need to introduce additional notions. Firstly we adapt *origin-hiding* from [CL19,

FHS19] to SPSEQ-UC. We want to guarantee that fresh and randomized signatures are indistinguishable, which is important to guarantee the anonymity of showings in DACs. Secondly, we newly introduce *derivation-privacy*, which guarantees that signatures obtained by extending commitment vectors are indistinguishable from fresh signatures on extended commitment vectors. Thirdly, we introduce *conversion-privacy*, which guarantees that when switching a user key in the signature, the resulting signature is indistinguishable from a fresh signature on the new user public key. We note that these properties compose (the outputs are always valid signatures and update keys) and can thus be applied an arbitrary number of times and in an arbitrary order. The notions are essential to the anonymity (of delegation) in the DAC application. Subsequently, we use $\approx$ to denote perfect indistinguishability.

Origin-hiding (also called signature adaptation [FHS15, FHS19]) formalizes the fact that signatures for well-formed commitment vectors and well-formed update keys output by $\mathsf{ChangRep}$ are distributed identical to fresh signatures on the new representative.

Definition 45 (Origin-hiding)**.** *For all* (λ, t, ℓ) *and* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*, for all* vk*,* pk_u*,* $\mathbf{C}$*,* $\mathbf{M}$*,* $\mathbf{O}$*,* $\mathbf{T}$*,* $\mathbf{U}$*,* $\mathsf{uk}_{k'}$*,* k' *and* σ*. If* $\mathsf{pk}_u \in \mathsf{UKeyGen}$ *and* $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j\in k} = 1 \ \wedge \ \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = 1$*, then for all* μ, ψ*, the algorithm* $\mathsf{ChangRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi)$ *outputs a uniformly random* $\mathbf{C}' \in [\mathbf{C}]_{\mathcal{R}^k}$ *and uniformly random* pk'_u*, and* σ' *(and if* $\mathsf{uk}_{k'} \neq \perp$ *update key* $\mathsf{uk}'_{k'}$*) in the respective spaces.*

Since we support the extension of the signed commitment vector, with derivation privacy we guarantee that signatures derived on a commitment vector $\mathbf{C}^*$ output by $\mathsf{ChangeRel}$ are indistinguishable from signatures freshly created with sk by running Sign on the extended vector.

Definition 46 (Derivation-privacy)**.** *For all* (λ, t, ℓ)*,* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*, all* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$*,* pk_u*,* $\mathbf{M}$*,* $\mathbf{O} = \rho$*,* $\mathbf{T}$*,* $\mathbf{U}$*,* $\mathsf{uk}_{k'}$*,* k'*, and* σ*. If* $\mathsf{pk}_u \in \mathsf{UKeyGen}$ *and* $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j\in k} = 1 \ \wedge \ \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = 1 \ \wedge \ \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$*, then, for all* $k'' \in [k+1, k']$*,* M_l*, we have* $(\sigma', (\mathbf{C}', O_l), \mathsf{uk}_{k''}) \leftarrow \mathsf{ChangeRel}(M_l, \sigma, \mathbf{C}, \mathsf{uk}_{k'}, k'')$ *and the following holds:*

$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_u, \mathsf{uk}_{k'}, (\sigma', (\mathbf{C}', \mathbf{O}'), \mathsf{uk}_{k''})) \approx$$
$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_u, \mathsf{uk}_{k'}, \mathsf{Sign}(\mathsf{sk}, \mathbf{M}', k'', \mathsf{pk}_u; \rho))$$

where $\mathbf{M}' = (\mathbf{M}, M_l)$ *and* $\mathbf{O}' = (\mathbf{O}, O_l)$*.*

With conversion-privacy we require that a converted signature, i.e., a signature where the public key has been switched, is identically distributed to a fresh signature using the new public key.

Definition 47 (Conversion-privacy)**.** *For all* (λ, t, ℓ)*,* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*, and for all* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$*,* $(\mathsf{sk}_u, \mathsf{pk}_u) \in \mathsf{UKeyGen}$*,* $\mathbf{C}, \mathbf{T}$*,* $\mathbf{M}$*,* $\mathbf{U}, \mathbf{O} = \rho$*,* $\mathsf{uk}_{k'}$*,* k'*, and* σ*. If*

$\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j\in k} = 1 \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = 1 \wedge \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$*, then for all* $(\mathsf{pk}_{u'}, \mathsf{sk}_{u'}) \in \mathsf{UKeyGen}$*, the following holds:*

$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_{u'}, (\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma), (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'})) \approx (\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_{u'}, \mathsf{Sign}(\mathsf{sk}, \mathbf{M}, k', \mathsf{pk}_{u'}; \rho)).$$

We can also define a class-hiding notion in the vein of [CL19,FHS19]. However, analogous to [FHS15] (Proposition 1), the origin-hiding notion together with the indistinguishability of the message space (under DDH) implies a stronger notion. We will use the above properties in combination with the DDH assumption later directly in the proof of anonymity of the DAC scheme. By class-hiding, we mean that, for all k, $1 < k \leq \ell$, given two messages vectors $\mathbf{C}_1$ and $\mathbf{C}_2$ of size k, it should be hard to tell whether or not $\mathbf{C}_1 \in [\mathbf{C}_2]_{\mathcal{R}^k}$.

Definition 48 (Class-hiding)**.** *A* SPSEQ-UC *scheme for parameterized equivalence relations* $\mathcal{R}^k$ *is class-hiding if for all* λ *and polynomial-length* $\ell(\lambda)$ *and all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, there exists a negligible function* ϵ *such that:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, 1^\ell, 1^t); \mathbf{C}_1 \leftarrow (\mathbb{G}_1^*)^k; \\ \mathbf{C}_2^0 \leftarrow (\mathbb{G}_1^*)^k; \mathbf{C}_2^1 \leftarrow [\mathbf{C}_1]_{\mathcal{R}^k}; b \leftarrow \{0,1\}; \\ b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathbf{C}_1, \mathbf{C}_2^b) : b' = b\end{array}\right] \leq \frac{1}{2} + \epsilon(\lambda)$$

4.4.3 Construction

We now present our SPSEQ-UC construction and start with an intuition behind our approach. We start from the SPSEQ scheme in [FHS15]. Inspired by their implicit use of SC in [FHS19] to construct traditional ACs, we make the message space of the scheme a vector of set commitments in a way that meets our requirements. Consequently, SPSEQ-UC encodes each message set $M_i \subset \mathbb{Z}_p^t$ into a set commitment C_i and signs a commitment vector $(\mathbb{G}_1^*)^k$ of size $k \leq \ell$. The randomization of the commitment vector is identical to a change of representative in the SPSEQ. More specifically, we use the algorithm SC.Commit to encode a message set to a set commitment in Sign and also an algorithm RndmzC to change the commitment vector representative. Note that as made explicit in the unforgeability game, we allow the adversary to control the randomness (opening information) used in commitments. So we choose this randomness externally and pass it to Sign which then passes it to SC.Commit. The most significant change compared to SPSEQ is the feature of updating a commitment vector by appending new commitments, and the support to adapt a signature to this updated commitment vector. Therefore, we can use the elements from the set commitment parameters $\{P^{a^i}\}_{0<i<t}$ and bind them to the signing key of the respective position of the vector and the randomness used in the Sign. We do this for all indices from k to k' and finally use these elements as the update key. Now, one can add a new commitment (message set) to the signed commitment vector using algorithm ChangeRel that receives a new message set, a signature and the update key. It first encodes

this set to a commitment. Then, it uses the update key to create another set commitment value for the new message set, which can be easily aggregated into the signature (element Z) and get the new signature for the updated commitment vector. For the user's public key bound to a signature, instead of signing the user's public key by including it in the vector, we define the extra element T to tie the signature to this key. This allows us to update the user's public keys by only locally updating the T element in the signature but still guaranteeing unforgeability. The $\mathsf{RndmzPK}$ then allows to randomize a public key consistently with the signature and in a way to achieve a new independent user public key for each call to $\mathsf{ChangRep}$.

$\mathsf{PPGen}(1^\lambda, 1^t, 1^\ell) \to (\mathsf{pp})$: Run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$. Pick $\alpha \leftarrow \mathbb{Z}_p^*$ and run $\mathsf{pp_{SC}} = (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i\in[t]} \leftarrow \mathsf{SC.Setup}(1^\lambda, 1^t; \alpha)$, and define $S_{\mathsf{SC}} \leftarrow \{M \subset \mathbb{Z}_p | 0 < |M| \leq t\}$. Output $\mathsf{pp} = \{\mathsf{BG}, \mathsf{pp_{SC}}, S_{\mathsf{SC}}, \ell\}$.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: For $0 \leq i \leq \ell$ pick $x_i \leftarrow (\mathbb{Z}_p^*)^\ell$, set the signing key $\mathsf{sk} = (x_0, \ldots, x_\ell)$. Compute the related verification key $\mathsf{vk} = (X_0, \hat{X}_0, \ldots, \hat{X}_\ell)$, where $X_0 = P^{x_0}$ and $\hat{X}_i = \hat{P}^{x_i}$ for $0 \leq i \leq \ell$. Output $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{UKeyGen}(\mathsf{pp}) \to (\mathsf{sk}_u, \mathsf{pk}_u)$: Pick $w_u \leftarrow \mathbb{Z}_p^*$, set $\mathsf{pk}_u \leftarrow P^{w_u}$ and $\mathsf{sk}_u = w_u$, and finally return $(\mathsf{sk}_u, \mathsf{pk}_u)$.

$\mathsf{RndmzC}(\mathbf{C}, \mathbf{O}, \mu) \to \mathbf{C}'$: On input a set commitment vector $\mathbf{C} \in [\mathbf{C}]_{\mathcal{R}^k}$ corresponding openings $\mathbf{O}$ and randomness μ, produces a new representative of the set commitment vector $\mathbf{C}' = \mathbf{C}^\mu$ and corresponding openings $\mathbf{O}' = \mu\mathbf{O}$.

$\mathsf{Sign}(\mathsf{sk}, \mathbf{M}, k', \mathsf{pk}_u; \rho) \to (\sigma, (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'})$: On input the signing key sk, a vector of message sets $\mathbf{M} = (M_1, \ldots, M_k)$, an index k', $k \leq k' \leq \ell$, a user public key pk_u and a vector of randomness ρ. For all $j \in [k]$ run $(C_j, O_j)_{j\in[k]} \leftarrow \mathsf{SC.Commit}(M_j; \rho_j)$, and get a vector of set commitments $\mathbf{C} = (C_1, \ldots, C_k)$ related to a vector of sets $\mathbf{M}$ and opening $\mathbf{O} = (O_1, \ldots, O_k)$. More precisely, $\mathsf{SC.Commit}$ computes a set commitment for each M_j in $\mathbf{M}$ as follows: define a polynomial $f_{M_j}(X) := \prod_{m\in M_j}(X - m) = \sum_{i=0}^{|M_j|} f_i \cdot X^i$ and with $\rho_j \in \rho$, compute:

$$C_j = \left(\prod_{i=0}^{|M_j|} \left(P^{\alpha^i}\right)^{f_i}\right)^{\rho_j} \text{ and } O_j = \rho_j,$$

where P^{α^i} are elements in pp. Then, compute a signature σ for $(\mathsf{pk}_u, \mathbf{C})$ as follows: Pick a random $y \leftarrow \mathbb{Z}_p^*$ and compute $\sigma =$

$$\left(Z \leftarrow \left(\prod_{j\in[k]} C_j^{x_j}\right)^{\frac{1}{y}}, Y \leftarrow P^y, \hat{Y} \leftarrow \hat{P}^y, T \leftarrow P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0}\right)$$

Also, if $k \neq \ell$, compute an update key for a range between k and k' as:

$$\mathsf{uk}_{k'} = \left(\left(\mathsf{usign}_j = \left((P^{\alpha^i})^{x_j} \right)^{y^{-1}} \right)_{j \in [k+1,k'], i \in [t]} \right).$$

Output $(\sigma, (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'})$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) \to 0/1$: On input a verification key vk, a user public key pk_u, a commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$, the purported signature σ, and a pair $(\mathbf{T}, \mathbf{U})$, it outputs 0 if any of the following checks fail and 1 otherwise:

- Check whether σ is a valid for $(\mathbf{C}, \mathsf{pk}_u)$, i.e., output 0 if one of the following checks fails:

$$\prod_{j=1}^{k} e(C_j, \hat{X}_{j+1}) = e(Z, \hat{Y}) \quad \wedge \quad e(Y, \hat{P}) = e(P, \hat{Y})$$
$$\wedge \quad e(T, \hat{P}) = e(Y, \hat{X}_1) \cdot e(\mathsf{pk}_u, \hat{X}_0).$$

- For all $(T_i, U_i) \in (\mathbf{T}, \mathbf{U})$ if $U_i = W_i$, then W_i is a witness for T_i being subset of the set committed to C_i: run $\mathsf{SC.VerifySubset}(C_i, T_i, W_i)$, i.e., output 1 if the following equation holds; else 0:

$$\prod_{i \in [|\mathbf{W}|]} e(W_i, \hat{P}^{f_{T_i}(\alpha)}) = \prod_{i \in [|\mathbf{W}|]} e(C_i, \hat{P})$$

  Else $U_i = O_i$, then $T_i = M_i$ is a message set and O_i is valid opening of C_i to T_i: run $\mathsf{SC.Open}(C_i, T_i, O_i)$, i.e., output 1 if the following holds; else 0:

$$\forall i \in [k] : O_i = \rho_i \wedge C_i = (P^{f_{M_i}(\alpha)})^{\rho_i}$$

$\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) \to 0/1$. On input a vk, $\mathsf{uk}_{k'}$, index k', a signature $\sigma = (Z, Y, \hat{Y}, T)$, parse $\mathsf{uk}_{k'} = (\mathsf{usign}_j = ((P^{\alpha^i})^{x_j})^{y^{-1}})_{j \in [k+1,k'], i \in [t]}$ output 1 if the following holds; else 0:

$$\bigwedge_{i \in [t], j \in [k+1,k']} e(P^{\alpha^i}, \hat{X}_j) = e(((P^{\alpha^i})^{x_j})^{y^{-1}}, \hat{Y})$$

$\mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi) \to \mathsf{pk}'$: On input a user public key pk_u and randomness $\psi, \chi \in \mathbb{Z}_p^*$, output the randomized public key $\mathsf{pk}'_u = (\mathsf{pk}_u \cdot P^{\chi})^{\psi}$, related to the secret key $(\chi + \mathsf{sk}_u)\psi$.

$\mathsf{ChangRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi) \to (\sigma', (\mathbf{C}', \mathbf{O}'), (\mathsf{uk}'_{k'} \text{ or } \perp), \mathsf{pk}'_u, \chi)$: On input a user public key pk_u, optionally an update key $\mathsf{uk}_{k'}$, a commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$ in equivalence class $[\mathbf{C}]_{\mathcal{R}^k}$ with corresponding openings $\mathbf{O}$, a valid signature σ for $\mathbf{C}$ and randomness $\psi, \mu \in \mathbb{Z}_p^*$.

Pick $\chi \leftarrow \mathbb{Z}_p^*$ and compute a new commitment representative $(\mathbf{C}', \mathbf{O}') \leftarrow \mathsf{RndmzC}(\mathbf{C}, \mathbf{O}, \mu)$ as well as a randomized user public key $\mathsf{pk}'_u \leftarrow \mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi)$ and update signature as $\sigma' = (Z^{\frac{\mu}{\psi}}, Y^{\psi}, \hat{Y}^{\psi}, (T \cdot X_0^{\chi})^{\psi})$. Moreover, if $\mathsf{uk}_{k'} \neq \perp$, check if $\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, randomize the update key:

$$\mathsf{uk}'_{k'} = \left((\mathsf{usign}_j^{\mu \cdot \psi^{-1}})_{j \in [k+1, k']} \right),$$

and output $(\sigma', (C', \mathbf{O}'), (\mathsf{uk}'_{k'} \text{ or } \perp), \mathsf{pk}'_u, \chi)$.

$\mathsf{ChangeRel}(M_l, \sigma, \mathbf{C}, \mathsf{uk}_{k'}, k'') \rightarrow (\sigma', (\mathbf{C}', O_l), \mathsf{uk}_{k''})$: On input a message set $M_l \subset S_{\mathsf{SC}}$ for $l = k + 1 \in [k']$, a valid signature σ for commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$ in equivalence class $[\mathbf{C}]_{\mathcal{R}^k}$, a valid update key $\mathsf{uk}_{k'}$, and an index $k'' \leq k'$.

First it creates a set commitment $(C_l, O_l = \rho_l) \leftarrow \mathsf{SC.Commit}(M_l)$. Then, it performs the following steps to update the signature for a commitment vector including C_l:

- First, compute a set commitment ϑ_l as in $\mathsf{SC.Commit}$, but using keys of the l-component of $\mathsf{uk}_{k'}$ as

$$\mathsf{usign}_l = \left((P^{\alpha^i})^{x_l} \right)^{y^{-1}} \text{ for } i \in [t] :$$

$$f_{M_l}(X) = \sum f_i X^i \Rightarrow \vartheta_l = (\prod_{i \in [t]} \mathsf{usign}_{li}^{f_i})^{\rho_l} = \prod_{i \in [t]} ((\underbrace{P^{\alpha^i \cdot x_l \cdot y^{-1}}}_{\mathsf{usign}_{li}})^{f_i})^{\rho_l}$$

- Second, update σ for a commitment vector $\mathbf{C}' = (\mathbf{C}, C_l)$ as:

$$\sigma' = \left((Z \cdot \vartheta_l), Y, \hat{Y}, T \right).$$

- Finally for $k'' \in [l+1, k']$, update the update key $\mathsf{uk}_{k'}$ for $j \in [l+1, k'']$: $\mathsf{uk}_{k''} = \left(\mathsf{usign}_j \right)_{j \in [l+1, k'']}$.

  Output $(\sigma', (\mathbf{C}', O_l), \mathsf{uk}_{k''})$.

$\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \rightarrow \sigma_{\text{orph}}$: On input $\mathsf{vk} = (X_0, \hat{X}_0, \ldots, \hat{X}_\ell)$, a user secret key sk_u for the public key pk_u, and a valid signature $\sigma = (Z, Y, \hat{Y}, T)$. Output an orphan signature $\sigma_{\text{orph}} =$

$$\left(Z, Y, \hat{Y}, T' = T \cdot (X_0^{\mathsf{sk}_u})^{-1} \right)$$

$\mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'}, \sigma_{\text{orph}}) \rightarrow \sigma'$: On input the verification key vk, a secret key $\mathsf{sk}_{u'}$, an orphan signature $\sigma_{\text{orph}} = \left(Z, Y, \hat{Y}, T' \right)$. Output a signature σ' for $\mathsf{pk}_{u'}$ as:

$$\sigma' = \left(Z, Y, \hat{Y}, T'' = T' \cdot X_0^{\mathsf{sk}_{u'}} = P^{x_1 \cdot y} \cdot \mathsf{pk}_{u'}^{x_0} \right).$$

The correctness of our SPSEQ-UC construction follows form inspection. We formally show the following:

Theorem 4.4.1 (Unforgeability)**.** *Our* SPSEQ-UC *construction is unforgeable in the generic group model for type-3 bilinear groups.*

The proof of Theorem 4.4.1 is provided in our paper [MSBM23].

Theorem 4.4.2 (Privacy)**.** *Our* SPSEQ-UC *construction is origin-hiding, conversion-privacy and derivation Privacy based on definitions 45, 46 and 47, respectively.*

We now prove that our SPSEQ-UC construction from Section 4.4.3 is Origin-hiding (Def. 45) and provides Conversion-privacy (Def. 46) and Derivation-privacy (Def. 47).

Lemma 4.4.1 (Origin-hiding)**.** *The construction described in Section 4.4.3 is Origin-hiding.*

Proof. Origin-hiding of ChangRep follows from the perfect adaptation of SPSEQ [FHS19]. The only main difference here is the additional element T in a signature as well as elements $(\mathsf{pk}_u, \mathsf{uk}_{k'})$ which we show that they are correctly randomized in both algorithms. Let $\mathbf{C} \in (\mathbb{G}_1^*)^k$, $\mathbf{T} \subseteq \mathbf{M}$, any $\mathbf{O}, \mathbf{U}$ s.t. $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1$, $\mathsf{pk}_u \in \mathbb{G}_1$, and $(x_0, x_1, \ldots, x_\ell) \leftarrow (Z_p^*)^\ell$ be such that $\mathsf{vk} = ((\hat{P}^{x_i})_{i\in[0,\ell]}, P^{x_0})$. For some $y \in Z_p^*$, a signature $(Z, Y, \hat{Y}, T) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ satisfying $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, (Z, Y, \hat{Y}, T), (\mathbf{T}, \mathbf{U})) = 1$ along with $\mathsf{uk}_{k'}$ is of the form

$$\sigma = \left(\left(\prod_{i=1}^{k} C_i^{x_i} \right)^{y^{-1}}, P^y, \hat{P}^y, P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0} \right).$$

For $\mu, \psi \in Z_p^*$, $\mathsf{ChangRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\mathbf{C}, \mathbf{O}), (Z, Y, \hat{Y}, T), \mu, \psi)$ outputs

$$\sigma' = \left(\left(\prod_{i=1}^{k} C_i^{\mu \cdot x_i} \right)^{y^{-1} \cdot \psi}, P^{y \cdot \psi}, \hat{P}^{y \cdot \psi}, P^{x_1 \cdot y \cdot \psi} \cdot X_0^{\psi(\mathsf{sk}_u + \chi)} \right),$$

which is a uniformly random element σ' in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$. That is, all elements of the signature σ' are perfectly randomized using randomness μ, ψ, χ and conditioned on $\mathsf{Verify}(\mathsf{vk}, (\mathsf{pk}_u \cdot P^\chi)^\psi, \mathbf{C}^\mu, \sigma', (\mathbf{T}, \mathbf{U})) = 1$. Now we show that $\mathsf{uk}_{k'}$ (in the case that it is requested for further delegation), and pk_u are also randomized perfectly using ψ, χ and μ. For all k', an update key $\mathsf{uk}_{k'} \in (\mathbb{G}_1^*)^{[k+1,k']}$ s.t. $\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$ and $\mathsf{pk}_u \in \mathbb{G}_1^*$ are the from

$$\mathsf{uk}_{k'} = \left(\mathsf{usign}_j = \left((P^{\alpha^i})^{x_j} \right)_{j\in[k+1,k'] \wedge i \in [t]}^{y^{-1}} \right) \text{ and } \mathsf{pk}_u = P^{\mathsf{sk}_u}.$$

For $\mu, \psi \in Z_p^*$, $\mathsf{ChangRep}$ outputs the following form, so we have

$$\mathsf{uk}'_{k'} = \left((\mathsf{usign}_j)^{\psi^{-1}\cdot\mu}\right)_{j\in[k+1,k']} \text{ and } \mathsf{pk}'_u = P^{(\mathsf{sk}_u+\chi)\cdot\psi},$$

where $\chi \leftarrow Z_p^*$ is randomness selected locally for $\mathsf{RndmzPK}$. It is not difficult to see that all elements of the $\mathsf{uk}_{k'}$ are distributed as expected and also pk_u is perfectly randomized with ψ, χ and represents a uniform element in $\mathbb{G}_1^*$. So, $\mathsf{ChangRep}$ clearly produces signatures with the same distribution as Sign:

$$(\sigma, \mathbf{C}, \mathsf{uk}_{k'}, \mathsf{pk}_u) \approx (\sigma', \mathbf{C}', \mathsf{uk}'_{k'}, \mathsf{pk}'_u)$$

Lemma 4.4.2 (Conversion-privacy)**.** *The construction described in Section 4.4.3 provides Conversion-privacy.*

First of all, let us assume that $\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ includes $[\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \leftrightarrow \mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'})] \rightarrow \sigma'$, where σ' is a valid signature.

Proof. For all $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$, $(\mathsf{sk}_u, \mathsf{pk}_u) \in \mathsf{UKeyGen}$, $\mathbf{C}, \mathbf{T}, \mathbf{M}, \mathbf{U}, \mathbf{O}$ s.t. $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1$ and σ. If $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = 1$. The signature σ in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ is the form of:

$$\sigma = \left(\left(\prod_{i=1}^{k} C_i^{x_i} \right)^{y^{-1}}, P^y, \hat{P}^y, P^{x_1\cdot y} \cdot X_0^{\mathsf{sk}_u} = P^{x_1\cdot y} \cdot \mathsf{pk}_u^{x_0} \right).$$

Then for $(\mathsf{sk}_{u'}, \mathsf{pk}_{u'}) \in \mathsf{UKeyGen}(\mathsf{pp})$, $\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ outputs a new signature with the form of:

$$\sigma' = \left(\left(\prod_{i=1}^{k} C_i^{x_i} \right)^{y^{-1}}, P^y, \hat{P}^y, P^{x_1\cdot y} \cdot X_0^{\mathsf{sk}_{u'}} = P^{x_1\cdot y} \cdot \mathsf{pk}_{u'}^{x_0} \right).$$

It is clear that this looks like a fresh signature σ' in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ for $\mathsf{pk}_{u'}$ with randomness y. So the output of $\mathsf{ConvertSig}$ is distributed the same as the output of Sign. □

Lemma 4.4.3 (Derivation-privacy)**.** *The construction described in Section 4.4.3 provides Derivation-privacy.*

Proof. For all $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$, pk_u, $\mathbf{M}$, $\mathbf{O} = \rho$, k', k'', $\mathbf{T}$, $\mathbf{U}$, $\mathsf{uk}_{k'}$, and σ. If $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1 \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = 1 \wedge \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, then for an index $l = k + 1 \in [k + 1, k']$, let M_l be a message set such that the message vector is $\mathbf{M}^* = (\mathbf{M}, M_l)$ and the related commitment vector is $\mathbf{C}^* = (\mathbf{C}, C_l)$. We intend to show that $\mathsf{ChangeRel}$ produces outputs with the same distribution as Sign for vectors $\mathbf{M}^*$ and $\mathbf{C}^*$: $\mathsf{Sign}(\mathsf{sk}, \mathbf{M}^*, k', \mathsf{pk}_u; \rho) \approx \mathsf{ChangeRel}(M_l, \sigma, \mathbf{C}, \mathsf{uk}_{k'}, k'')$. More

precisely, for some $y \in Z_p^*$, a signature $\sigma = (Z, Y, \hat{Y}, T) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ satisfying $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \mathbf{C}^*, \sigma, (\mathbf{T}, \mathbf{U})) = 1$ is of the form

$$\sigma = \left(\left(\prod_{i=1}^{k} (C_i^*)^{x_i} \right)^{\frac{1}{y}}, P^y, \hat{P}^y, P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0} \right)$$

$\mathsf{ConvertSig}$ outputs the element Z of the signature as:

$$Z = \left(\prod_{i=1}^{k} (C_i^{x_i})^{\frac{1}{y}} \cdot \left(\prod_{i \in [t] \wedge l \in [k+1, k']} P^{\alpha^i \cdot x_l \cdot y^{-1}} \right)^{f_i} \right) =$$

$$\prod \left((C_i^{x_i})^{\frac{1}{y}} \cdot C_l^{x_l \cdot \frac{1}{y}} \right) = \prod (\underbrace{C_i^{x_i} \cdot C_l^{x_l}}_{\prod_{i=1}^{l} (C_i^*)^{x_i}})^{\frac{1}{y}}$$

So for the whole signature, it outputs the signature as:

$$\sigma' = \left(\left(\prod_{i=1}^{l} (C_i^*)^{x_i} \right)^{\frac{1}{y}}, P^y, \hat{P}^y, P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0} \right)$$

which looks like a fresh signature σ in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ for $\mathbf{M}^*$ using the randomness y. This is, for vectors $\mathbf{C}^*, \mathbf{M}^*$, $\mathsf{ConvertSig}$ produces signatures with the same distribution as Sign.

4.5 Cross-Set Commitment Aggregation

We introduce an aggregatable set commitment CSCA that allows non-interactive aggregation of witnesses across multiple commitments. We use a technique inspired by [BDFG20, GRWZ20] to batch different subset opening witnesses into one and to improve the efficiency of the verification operation with batching pairing equations. Functionality-wise, we require that witnesses for multiple subsets of multiple commitments can be aggregated into a single value called proof. This allows us to use the CSCA in the DAC scheme in order to open any subset of attributes in each set commitment efficiently. CSCA adds two additional algorithms in SC (Def. 2.4.3.3) to aggregate witnesses across k-commitments and verify them:

$\mathsf{AggregateAcross}(\{C_j, T_j, W_j\}_{j \in [k]}) \to \pi$. Takes as input a collection $\{C_j, T_j\}_{j \in [k]}$ along with the corresponding subset opening witnesses $\{W_j\}_{j \in [k]}$ (computed using $\mathsf{OpenSubset}$) and outputs an aggregated proof π.

$\mathsf{VerifyAcross}(\{C_j, T_j\}_{j \in [k]}, \pi) \to b$. Takes as input a collection $(\{C_j, T_j\}_{j \in [k]})$ along with a cross-commitment-aggregated proof π, and checks: for all $j \in [k]$, C_j is a set commitment to a message set consistent with the subset T_j.

On computing commitments. Similar to SC in Section 2.4.3.3, with the knowledge of the trapdoor α, we can compute a commitment when externally provided with the randomness ρ in the group as P^{ρ}. So, we add the commitment computation to $\mathsf{CSCA.Commit}_2(S_j, \alpha, P^{\rho})$ with addition input to commit group elements.

Construction A Cross Set commitment aggregation scheme CSCA consists of the following PPT algorithms:

$\mathsf{CSCA.Setup}(1^{\lambda}, 1^{t}) \rightarrow \mathsf{pp_{CSCA}}$: On input a security parameter λ and a maximum set cardinality t, run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^{\lambda})$, choose $H : \{0,1\}^* \rightarrow Z_p$, pick $\alpha \leftarrow Z_p$, store α as a trapdoor and output $\mathsf{pp_{CSCA}} \leftarrow (\mathsf{BG}, H, (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i\in[t]})$, which defines message space $S_{\mathsf{pp_{CSCA}}} = \{S \subset Z_p | 0 < |S| \leq t\}$. $\mathsf{pp_{CSCA}}$ will be an implicit input to all algorithms.

$\mathsf{CSCA.Commit}(S_j) \rightarrow (C_j, O_j)$: On input a set $S_j \in S_{\mathsf{pp_{SC}}}$: pick $\rho_j \leftarrow Z_p$, compute $C_j \leftarrow (P^{f_{S_j}(\alpha)})^{\rho_j} \in \mathbb{G}_1^*$ and output (C_j, O_j) with $O_j \leftarrow \rho_j$.

$\mathsf{CSCA.Commit}_2(S_j, \alpha, P^{\rho_j}) \rightarrow (C_j, O_j)$: On input a set $S_j \in S_{\mathsf{pp_{SC}}}$, α, and P^{ρ_j}: compute $C_j \leftarrow (P^{\rho_j})^{f_{S_j}(\alpha)} \in \mathbb{G}_1^*$ and output (C_j, O_j) with $O_j \leftarrow \perp$.

$\mathsf{CSCA.Open}(C_j, S_j, O_j) \rightarrow 0/1$: On input a commitment C_j, a set S_j, and opening $O_j = \rho_j$: if $C_j \notin \mathbb{G}_1^*$ or $\rho_j \notin Z_p^*$ or $S_j \notin S_{\mathsf{pp_{CSCA}}}$ then return $\perp$. Otherwise if $O_j = \rho_j$ and $C_j = (P^{f_{S_j}(\alpha)})^{\rho_j}$, return 1; else return 0

$\mathsf{CSCA.OpenSubset}(C_j, S_j, O_j, T_j) \rightarrow W_j$: On input a commitment C_j, a set S_j, opening O_j and a subset T_j, if $\mathsf{CSCA.Open}(C_j, S_j, O_j)$ or $T_j \not\subseteq S_j$ or $T_j = \emptyset$ then return $\perp$. If $O_j = \rho_j$, output $W_j \leftarrow (P^{f_{S_j \setminus T_j}(\alpha)})^{\rho_j}$.

$\mathsf{CSCA.VerifySubset}(C_j, T_j, W_j) \rightarrow 0/1$: On input the commitment C_j, the subset T_j and the witness W_j: if $C_j \notin \mathbb{G}_1^*$ or $T_j \notin S_{\mathsf{pp_{SC}}}$, return 0. Else if $W_j \in \mathbb{G}_1^*$ and $e(W_j, \hat{P}^{f_{T_j}(\alpha)}) = e(C_j, \hat{P})$, return 1; else 0.

$\mathsf{CSCA.AggregateAcross}(\{C_j, T_j, W_j\}_{j\in[k]}) \rightarrow \pi$. Takes as input a collection $(\{C_j, T_j\}_{j\in[k]})$ along with the corresponding subset opening witnesses $\{W_j\}_{j\in[k]}$ and outputs an aggregated proof π as follows:

$$\pi := \prod_{j\in[k]} W_j^{t_j}, \text{ where } t_j = H(j, \{C_j, T_j\}_{j\in[k]}).$$

$\mathsf{CSCA.VerifyAcross}(\{C_j, T_j\}_{j\in[k]}, \pi) \rightarrow 0/1$. Checks that the following equation holds:

$$\prod_{j\in[k]} e(C_j, \hat{P}^{t_j \cdot Z_{S\setminus T_j}(\alpha)}) = e(\pi, \hat{P}^{Z_S(\alpha)})$$

where $S = \bigcup_j T_j$, and $Z_S(\alpha) = \prod_{i\in S}(\alpha - i)$.

We require the same correctness of opening as before, extended to cross-commitment aggregation in a natural way. To fit with the DAC, one provides one group element in $\mathbb{G}_1$ proof π of the current values of the attributes required for the showing. When multiple subsets of disclosed attribute sets are used, cross-commitment aggregation allows us to compress witnesses $(W_1, \ldots, W_k)$ into a single proof π. This can help to reduce the bandwidth overhead significantly and improves verification efficiency by saving k pairing equations. Note that without aggregation verification has the form: $\prod_{i\in[k]} e(W_i, \hat{P}^{f_{T_i}(\alpha)}) = \prod_{i\in[k]} e(C_i, \hat{P})$.

Randomization. We can randomize π and commitments C_j as $(\pi^\mu, \mathbf{C}^\mu)$ and verification works out.

Security. We can view the set commitment from Section 2.4.3.3 used for the cross-commitment aggregation as an instantiation of [BDFG20], but restricted to monic polynomials. So their analysis carries over. Note that in AggregateAcross, to be non-interactive, we aggregate witnesses using t_j using a hash function H modeled as a random oracle (as done in [GRWZ20]), meaning that their analysis carries over.

4.6 Delegatable Anonymous Credentials

We now present our definition of delegatable anonymous credentials. It is similar to [CL19, FHS19], but splits up the issuing protocol for issuing a root credential (CreateCred) and delegating a credential (IssueCred) and works as follows: A root issuer (called CA) issues a level-1 ($L = 1$) credential (a root credential) to intermediate issuers using the CreateCred protocol. The credential is assigned for a user $\mathsf{sk_u}$ (whom it knows with a pseudonym $\mathsf{nym_u}$) with an attribute vector A and the related set commitments $\mathbf{C}$ rooted at $\mathsf{pk_{CA}}$. CA also creates a delegation key $\mathsf{dk}_{L'}$ which determines how the credential can be delegated further until level L'. With this key, a user U can replace an old pseudonym with a new one and also delegate their credential further to another user, say R, by switching to R's public key and possibly adding another set of attributes A'. Showing the credential consists of the user proving possession of the secret key $\mathsf{sk_u}$ and providing a randomized signature with required attributes.

Definition 49 (Delegatable anonymous credentials)**.** DAC *includes algorithms* (Setup, KeyGen, NymGen) *and protocols* CreateCred/*ReceiveCred,* IssueCred/ReceiveCred *for issuing a credential and* CredProve/CredVerify *for showing of credentials as:*

Setup$(1^\lambda, 1^t, 1^\ell) \to (\mathsf{pp}, \mathsf{sk_{CA}}, \mathsf{pk_{CA}})$: Takes as input the security parameters λ, an upper bound t for the cardinality of committed sets and a depth (i.e., level) parameter $\ell > 1$. It generates the public parameters pp for the system as well as a signing key $\mathsf{sk_{CA}}$ and public key $\mathsf{pk_{CA}}$ for all $i \in [\ell]$ for CA. Outputs the pp and CA key pair $(\mathsf{pp}, \mathsf{sk_{CA}}, \mathsf{pk_{CA}})$. pp will be implicitly input to all algorithms.

KeyGen$(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: Generates a key pair $(\mathsf{pk}, \mathsf{sk})$, where sk refers to the user's secret key, and pk refers to the user's public key (or an initial pseudonym).

$\mathsf{NymGen}(\mathsf{pk}) \rightarrow (\mathsf{nym}, \mathsf{aux})$: Takes as input a user's public key pk, and outputs a pseudonym nym for this user and the auxiliary information aux (randomness) needed to use nym.

Issuing a root credential:

$[\mathsf{CreateCred}(L', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{u}}, \mathsf{A})] \rightarrow (\mathsf{cred}, (\mathbf{C}, \mathbf{O}), \mathsf{dk}_{L'})$: This is an interactive protocol between a user (issuer) who is known by $\mathsf{nym}_{\mathsf{u}}$ and CA. The common inputs are the pp, the CA 's public key $\mathsf{pk}_{\mathsf{CA}}$ and the attribute set A. CA creates a root credential, i.e. the (powerful) delegatable credential for a set commitment C corresponding to the attribute set A and the related opening information O as well as a delegatable key $\mathsf{dk}_{L'}$ regarding the level L' for a user $\mathsf{nym}_{\mathsf{u}}$ ($\mathsf{nym}_{\mathsf{u}}$ is sent to the CA), rooted at $\mathsf{pk}_{\mathsf{CA}}$.

Issuing/delegating a credential:

$[\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, \mathsf{sk}_{\mathsf{u}}, \mathsf{cred}_{\mathsf{u}}, A_l, L'') \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{r}}, A_l)] \rightarrow (\mathsf{cred}_{\mathsf{r}}, \mathsf{dk}'_{L''})$: It is an interactive protocol between an issuer who is known by $\mathsf{nym}_{\mathsf{u}}$ and runs the $\mathsf{IssueCred}$ algorithm, and a receiver, who is known by $\mathsf{nym}_{\mathsf{r}}$ and runs the $\mathsf{ReceiveCred}$ side. The common inputs are the pp, CA 's public key $\mathsf{pk}_{\mathsf{CA}}$, and attribute set A_l. Also, the issuer takes as input the issuer's secret key sk_{u} and his own credential $\mathsf{cred}_{\mathsf{u}}$ (a signature) together with all information associated with it (e.g. A, the delegatable key $\mathsf{dk}_{L'}$, the pseudonym $\mathsf{nym}_{\mathsf{u}}$ and its associated auxiliary information (randomness) aux_u), and (optionally) a level $L'' < L'$ (if not set $L'' := L'$). The receiver takes as input her own secret key sk_{r}, and creates a $\mathsf{nym}_{\mathsf{r}}$ (and sends to $\mathsf{nym}_{\mathsf{u}}$), and the auxiliary information $\mathsf{aux}_{\mathsf{r}}$ associated with her pseudonym $\mathsf{nym}_{\mathsf{r}}$. At the end of the protocol, the receiver side outputs her credential $\mathsf{cred}_{\mathsf{r}}$ that is issued for $\mathsf{A}' = (\mathsf{A}, A_l)$ and $\mathsf{dk}'_{L''}$ (if further delegation is allowed) or $\perp$.

Showing of a credential:

$[\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{nym}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}, \mathsf{cred}_{\mathsf{P}}, D) \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{P}}, D)] \rightarrow (0, 1)$: The protocol involves two parties: a prover, responsible for proving possession of a credential and executing the $\mathsf{CredProve}$ side of the protocol, and a verifier, executing the $\mathsf{CredVerify}$ side. The shared inputs include pp, the public key $\mathsf{pk}_{\mathsf{CA}}$ of the credential authority (CA), and the subset of attributes D that need to be disclosed.

The prover's input includes their user secret sk_{P} and their credential, along with all associated information (i.e., A, the prover's pseudonym $\mathsf{nym}_{\mathsf{P}}$, and related auxiliary data $\mathsf{aux}_{\mathsf{P}}$). On the other hand, the verifier takes the common inputs and receives $\mathsf{nym}_{\mathsf{P}}$. It outputs 1 if it accepts the proof of possession of a credential for D and 0 otherwise.

Note that the nym's can be derived from the respective secret key and algorithms (KeyGen, NymGen), we avoid passing nym's as an explicit input whenever possible.

Definition 50 (Correctness of DAC)**.** DAC *is correct if* Setup, KeyGen, NymGen, CreateCred, *and* issuing/receiving *protocols are executed correctly on honestly generated inputs, then in an honest execution of the* proving/verifying *protocol, the verifier will accept the credential* cred *with probability 1.*

4.6.1 Security of DAC

We define our security model based on the game-based framework in [FHS19], with some modifications to harmonize their definition with the one on SPSEQ-UC. The adversary $\mathcal{A}$ has access to oracles that describe the possible ways to interact with the system. We use $\langle\mathcal{O}\rangle$ to denote the collection of oracles in the games. For the anonymity game we have $\langle\mathcal{O}\rangle = (\mathcal{O}^{\mathsf{User}}, \mathcal{O}^{\mathsf{Corrupt}}, \mathcal{O}^{\mathsf{CreateRoot}}, \mathcal{O}^{\mathsf{ObtIss}}, \mathcal{O}^{\mathsf{Obtain}}, \mathcal{O}^{\mathsf{RootObt}}, \mathcal{O}^{\mathsf{CredProve}})$ and for the unforgeability game $\langle\mathcal{O}\rangle = (\mathcal{O}^{\mathsf{User}}, \mathcal{O}^{\mathsf{CreateRoot}}, \mathcal{O}^{\mathsf{Corrupt}}, \mathcal{O}^{\mathsf{ObtIss}}, \mathcal{O}^{\mathsf{RootIss}}, \mathcal{O}^{\mathsf{Issue}}, \mathcal{O}^{\mathsf{CredProve}})$. We define four global lists that are shared among oracles as $\mathcal{HU}$ a list of honest users, $\mathcal{CU}$ a list of corrupted users, $\mathcal{L}_{uk}$ a list of user's keys, and $\mathcal{L}_{\mathsf{cred}}$ a list of user-credential pairs which includes issued credentials and corresponding attributes and to which user they were issued. Note that $\mathsf{dk}_{k'}$ implicitly shows k' and subsequently $\mathcal{R}_{k'}$. Moreover, in each issuing query $(\mathcal{O}^{\mathsf{ObtIss}}, \mathcal{O}^{\mathsf{Issue}})$ only one commitment (and attributes set) is added in the commitment vector. Also, for simplicity, we assume that cred_i contains $(\sigma, (\mathbf{C}, \mathbf{O}, \mathsf{pk}_i = \mathsf{nym}_i), \mathsf{aux}_i)$.

$\mathcal{O}^{\mathsf{User}}(i)$: Takes as input a user identity i. If $i \in \mathcal{HU}$ or $i \in \mathcal{CU}$ it returns $\perp$, else it creates a fresh entry i in lists $\mathcal{HU}$ and $\mathcal{L}_{uk}$ by running $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and adding i and $(\mathsf{sk}_i, \mathsf{pk}_i)$ to the list $\mathcal{HU}$ and $\mathcal{L}_{uk}$, receptively. It returns $\mathsf{pk}_i = \mathsf{nym}_i$.

$\mathcal{O}^{\mathsf{Corrupt}}(i, \mathsf{pk}_i)$: On input a user identity i and a public key pk_i. If $i \notin \mathcal{HU}$, a new corrupt user with public key pk_i (or nym_i) is registered and add $i \in \mathcal{CU}$, else it moves the entry corresponding to i from the list of honest users $\mathcal{HU}$ and adds it to the list of corrupted users $\mathcal{CU}$. Then, it returns sk_i and all the associated credentials items $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i)$ of $\mathcal{L}_{\mathsf{cred}}[i]$. Finally, it sets the form $(\perp, \mathsf{A}, k', \perp) \in \mathcal{L}_{\mathsf{cred}}$ for all A and i in this case.

$\mathcal{O}^{\mathsf{CreateRoot}}(i, k', \mathsf{A})$: Takes as input a user identity i, an index k' and attributes A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it creates a root credential by running

$$\begin{bmatrix}\mathsf{CreateCred}(k', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \\ \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_i, \mathsf{A})\end{bmatrix} \rightarrow (\mathsf{cred}_i, \mathsf{dk}_{k'})$$

with a user i for an attribute set A and appends $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$.

$\mathcal{O}^{\mathsf{RootIss}}(k', \mathsf{A})$: Takes as input an index k' and attributes A. It creates a root credential by running the CreateCred protocol with $\mathcal{A}$: $\mathsf{CreateCred}(k', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathcal{A}$ for an attribute set A and appends $(\perp, \mathsf{A}, k', \perp)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle allows an adversary represented by nym_i to play a corrupted user to get a root credential from a CA.

$\mathcal{O}^{\mathsf{RootObt}}(i, k', \mathsf{A})$: On input a user identity i, an index k' and attributes A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it creates a root credential by running the Receive protocol with $\mathcal{A}$ who impersonates a malicious CA to issue a root credential to an honest user i by running: $\mathcal{A} \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_i, \mathsf{A})$. If $\mathsf{cred}_i = \perp$ the oracle returns $\perp$. Else it stores the resulting $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i) \in \mathcal{L}_{\mathsf{cred}}$.

$\mathcal{O}^{\mathsf{ObtIss}}(i, j, A_l, k'')$: Takes as user identities i and j, a set of attributes A_l, and (optionally) an index k''. It makes user i delegate a credential to user j. If $i, j \notin \mathcal{HU}$ or $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i) \notin \mathcal{L}_{\mathsf{cred}}$ it returns $\perp$, else finds entries $(\mathsf{sk}_i, \mathsf{cred}_i)$, sk_j, and runs the issuing protocols as:

$$\begin{bmatrix} \mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{k'}, \mathsf{sk}_i, \mathsf{cred}_i, A_l, k'') \\ \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_j, A_l) \end{bmatrix} \rightarrow (\mathsf{cred}_j, \mathsf{dk}'_{k'})$$

and adds the entry $(j, \mathsf{A}', \mathsf{dk}'_{k'}, \mathsf{cred}_j)$ to $\mathcal{L}_{\mathsf{cred}}$, where $\mathsf{A}' = (\mathsf{A}, A_l)$.

$\mathcal{O}^{\mathsf{Obtain}}(j, A_l, k'')$: On input a user identity $j \in \mathcal{HU}$ and a set of attributes A_l, and (optionally) an index k''. If $j \notin \mathcal{HU}$ it returns $\perp$. Else, the oracle runs the Receive protocol with $\mathcal{A}$: $\mathcal{A} \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_j, A_l)$. If $\mathsf{cred}_j = \perp$ the oracle returns $\perp$. Else it stores the resulting output $(\mathsf{cred}_j, \mathsf{dk}'_{k'}, \mathsf{A}')$ and it appends $(j, \mathsf{A}', \mathsf{dk}'_{k'}, \mathsf{cred}_j)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by $\mathcal{A}$, whom it knows by nym_i impersonating an issuer to issue a credential to an honest user j.

$\mathcal{O}^{\mathsf{Issue}}(i, A_l, k'')$: On input a user identity i, a set of attributes A_l, and (optionally) an index k''. If $i \notin \mathcal{HU}$ it returns $\perp$. Also it checks if $\not\exists(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i) \in \mathcal{L}_{\mathsf{cred}}$, returns $\perp$. Else, it runs: $\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{k'}, \mathsf{sk}_i, \mathsf{cred}_i, A_l, k'') \leftrightarrow \mathcal{A}$. The elements $(\perp, \mathsf{A}' = (\mathsf{A}, A_l), k', \perp)$ are then added to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by a corrupted user with adversarial nym_j to get a credential from honest issuer i.

$\mathcal{O}^{\mathsf{CredProve}}(j, D)$: On input an index of an issuance j and subsets D. This oracle first parses $\mathcal{L}_{\mathsf{cred}}[j]$ as $(i, \mathsf{A}', \mathsf{dk}'_{k'}, \mathsf{cred}_i)$. Let cred_i be the credential issued on A' for a user i during the i-th query to $\mathcal{O}^{\mathsf{ObtIss}}$ or $\mathcal{O}^{\mathsf{Obtain}}$ (or it can be outputs of $\mathcal{O}^{\mathsf{CreateRoot}}$ when directly issued by CA). If $i \notin \mathcal{HU}$ returns $\perp$. Else, it retrieves $(\mathsf{aux}_i, \mathsf{nym}_i, \mathsf{sk}_i)$ for i from lists cred_i and $\mathcal{L}_{uk}$, and runs: $\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_i, \mathsf{nym}_i, \mathsf{aux}_i, \mathsf{cred}_i, D) \leftrightarrow \mathcal{A}$, with the adversary playing the role of the verifier.

Anonymity. Anonymity requires that a malicious verifier cannot distinguish between any two users. The adversary has adaptive access to an oracle that on the input of two distinct user indexes i_0 and i_1, acts as one of the two credential owners (depending on bit b) in the verification algorithm. Note that $D(\mathsf{A}') = 1$ if attributes in A' satisfies the policy subset and $D(\mathsf{A}') = 0$ otherwise. Moreover, $\mathsf{cred}_b \nleftarrow \mathcal{O}^{\mathsf{ObtIss}}$ requires that credentials cred_0 and cred_1 are on a delegation path from a (corrupted) root credential where all delegations have been performed honestly, but the respective users might all be corrupted. We note that this requirement is similar to the anonymity model of CL in [CL19], however, we

additionally allow the adversary to access the user corruption oracle in which we reveal the user's secret keys to the adversary. CL can not support this type of corruption as then the anonymity of their construction breaks down.This makes our model stronger than the one of CL. The essence of the game is captured by the oracles $\mathcal{O}_b^{\mathsf{Anon}}$ in Fig 5.3. To make the game non-trivial, we impose restrictions that the policy is either satisfied or not by both credentials and they have commitment vectors of equal length. Note this also implies unlinkability for delegation anonymity.

Definition 51 (Anonymity)**.** *A* DAC *is anonymous, if for all* $(\lambda, \ell, t) \in \mathbb{N}$*, any* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *so that* $|\Pr[\mathsf{ExpAno}^0_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t) = 1] - \Pr[\mathsf{ExpAno}^1_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t) = 1]| \leq \frac{1}{2} + \epsilon(\lambda)$*, experiments are defined in Fig 5.3, respectively.*

Unforgeability. Unforgeability is the property that ensures no adversary can deceive a verifier into accepting a credential for a set of attributes that they do not possess. In other words, an adversary $\mathcal{A}$ wins the unforgeability experiment (cf. Fig 5.3) if $\mathcal{A}$ can convince an honest verifier that they satisfy a specific attributes subset without having the required credential.

Definition 52 (Unforgeability)**.** *A* DAC *is unforgeable if, for all* $(\lambda, \ell, t) \in \mathbb{N}$*, for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\lambda)$ *such that* $\Pr[\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t) = 1] \leq \epsilon(\lambda)$*, where the experiment* $\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)$ *is defined in Fig 5.3.*

$\underline{\mathsf{ExpAno}^b_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)}$:

- $(\mathsf{pp}, \mathsf{pk}_{\mathsf{CA}}, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, 1^\ell, 1^t)$
- $b' \leftarrow \mathcal{A}^{\langle \mathcal{O}_b^{\mathsf{Anon}}, \mathcal{O} \rangle}(\mathsf{st})$
- **return** $(b = b')$

$\underline{\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)}$:

- If j_0 or $\mathsf{j}_1 > |\mathcal{L}_{\mathsf{cred}}|$ the oracle returns $\perp$.
- Else, it parses $\mathcal{L}_{\mathsf{cred}}[\mathsf{j}_0]$ as $(i_0, \mathsf{A}'_0, \mathsf{dk}_0, \mathsf{cred}_0)$ and $\mathcal{L}_{\mathsf{cred}}[\mathsf{j}_1]$ as $(i_1, \mathsf{A}'_1, \mathsf{dk}_1, \mathsf{cred}_1)$.
- If $D(\mathsf{A}'_0) \neq D(\mathsf{A}'_1) \vee |\mathbf{C}_0| \neq |\mathbf{C}_1| \vee \mathsf{cred}_b \nleftarrow \mathcal{O}^{\mathsf{ObtIss}}$, **return** $\perp$.
- Otherwise run: $\mathcal{A} \leftrightarrow \mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_b, \mathsf{nym}_b, \mathsf{aux}_b, \mathsf{cred}_b, D)$

$\underline{\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)}$:

- $(\mathsf{pp}, (\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^t)$
- $(D^*, \mathsf{nym}^*) \leftarrow \mathcal{A}^{\langle \mathcal{O} \rangle}(\mathsf{pp}, \mathsf{pk}_{\mathsf{CA}})$
- $b \leftarrow (\mathcal{A} \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}^*, D^*))$
- $\forall(\perp, \mathsf{A}', k', \perp) \in \mathcal{L}_{\mathsf{cred}}$: If $(D^*, \mathsf{A}') \in \mathcal{R}_{k'}$, **return** 0
- Else **return** b

Figure 4.2: Experiments $\mathsf{ExpAno}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)$ and $\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)$.

4.6.2 Construction of DAC

We first give an intuition of our construction. To obtain a root credential from a CA, a user needs to send one of their pseudonyms to the CA and use some mechanism to authenticate

with the real identity. The latter concern is outside of the protocol and omitted here. The initial nym is viewed as a user public key pk in the SPSEQ-UC scheme Σ, one can always derive a new nym using NymGen, which calls RndmzCpk of Σ to randomize the nym, and the respective signature can be adapted to the randomized nym. CA then creates a root credential (signature of Σ) on a vector of two commitments and a delegation key. Regarding these first two commitments in CreateCred protocol, we can assume the first dummy commitment for a fixed set A_1 (that is used by all credentials and are never shown in practice) and the second commitment holds the initial attribute set A_2. We only require this restriction for the root credential.

Via the IssueCred protocol, users U can then delegate their credentials to another user R using the delegatable key. U needs to derive a signature on R's secret key sk_r without being given the key directly. This is achieved by removing U's pseudonym nym_u and switching to R's pseudonym nym_r using ConvertSig of Σ and adding any additional attribute set A_i using ChangeRel of Σ. To show a credential, a user U randomizes the credential σ_u and nym_u using ChangRep of Σ and providing a ZKPoK that demonstrates that the secret key and randomness $\mathsf{sk}_\mathsf{u}, \mathsf{aux}_u$ belong to the new (randomized) nym_u along with opening subsets of the attributes D using the CSCA scheme.

We provide our DAC construction in Fig 4.3. It is based on our SPSEQ-UC signature (denoted by Σ) from Section 4.4.3 and the CSCA scheme from Section 4.5. We use the following notation: Assume attribute universe $\mathcal{U} = S_{\mathsf{SC}}$, $\mathsf{A} = \mathbf{M}$, the updatable key as a delegatable key $\mathsf{uk}_{k'} = \mathsf{dk}_{L'}$ and $k' = L' + 1$ and the root authority's public key $\mathsf{pk}_{\mathsf{CA}} = \mathsf{vk}$. Then each credential is parameterized with a vector $\mathsf{A} = (A_1, \ldots, A_k)$, where $A_i \subseteq \mathcal{U}$ is an attribute set and consider the relation in Definition 43 for attributes which defines how a user can use their credentials. For the sake of compactness, we write $\mathsf{ZKPoK}(w, x)$ or $\mathsf{ZKPoK}(w, x)$ for a (non)-interactive zero-knowledge proof of knowledge of witness w for statement x and $\mathsf{ZKPoK}(w, x) = 1$ if verifier accepts (cf. Section 2.4.4 for a formal treatment). In Fig 4.3, we replace SC by CSCA (cf. Section 4.5) to improve the communication bandwidth by aggregating witnesses and also verification efficiency due to batching of pairing equations. It also allows creating a set commitment for openings ρ in the group as P^ρ. We stress that CSCA is fully compatible with SC and provides identical functionality.

Theorem 4.6.1. *The* DAC *construction in Figure 4.3 is correct, unforgeable, and anonymous.*

Proofs. The proof is presented in as follows, where we note that in our formal proof, we consider our core SPSEQ-UC which is based on SC. Unforgeability follows from the ZKPoK and unforgeability of SPSEQ-UC. While anonymity follows from the ZKPoK, privacy properties of SPSEQ-UC and the DDH assumption. Note that when using CSCA instead of SC and NIZK obtained via the Fiat-Shamir heuristic, then the proofs are in the random oracle model (ROM).

Remark 1. *We note that except for the* NIZK *in* Setup*, we require multiple-extractions in*

$\mathsf{Setup}(1^\lambda, 1^t, 1^\ell)$: Pick $\alpha \leftarrow \mathbb{Z}_p$ and run $\mathsf{pp}_\Sigma \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, 1^t, 1^{(\ell+1)}; \alpha)$. Generate a key pair $(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$ for CA as $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(\mathsf{pp}_\Sigma)$, and set $\mathsf{pk}_{\mathsf{CA}} = (\mathsf{vk}, P^\alpha)$, and $\mathsf{sk}_{\mathsf{CA}} = (\mathsf{sk}, \alpha)$, and run $\pi_{\mathsf{CA}} \leftarrow \mathsf{NIZK}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$. Output $\mathsf{pp} = (\mathsf{pp}_\Sigma, \mathsf{pk}_{\mathsf{CA}}, \pi_{\mathsf{CA}})$. The attribute space is $\mathcal{U} = S_{\mathsf{pp}_\Sigma}$ of pp_Σ.

$\mathsf{KeyGen}(\mathsf{pp})$: Pick $w_{\mathsf{u}} \leftarrow \mathbb{Z}_p^*$, set $\mathsf{pk}_{\mathsf{u}} \leftarrow P^{w_{\mathsf{u}}}$ and $\mathsf{sk}_{\mathsf{u}} = w_{\mathsf{u}}$, and return $(\mathsf{sk}_{\mathsf{u}}, \mathsf{pk}_{\mathsf{u}})$.

$\mathsf{NymGen}(\mathsf{pk}_{\mathsf{u}})$: Pick randomness $\psi, \chi \leftarrow \mathbb{Z}_p^*$, compute $\mathsf{nym}_{\mathsf{u}} = \mathsf{pk}'_{\mathsf{u}} \leftarrow \Sigma.\mathsf{RndmzPK}(\mathsf{pk}_{\mathsf{u}}, \psi, \chi)$, set $\mathsf{aux}_{\mathsf{u}} = (\chi, \psi)$, and output $(\mathsf{nym}_{\mathsf{u}}, \mathsf{aux}_{\mathsf{u}})$.

Issuing a root credential

$[\mathsf{CreateCred}(L', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{u}}, \mathsf{A})] \rightarrow (\mathsf{cred}, (\mathbf{C}, \mathbf{O}), \mathsf{dk}_{L'})$:

U picks $\rho_j \leftarrow \mathbb{Z}_p^*$, sends $((P^{\rho_j})_{j\in[2]}, \mathsf{nym}_{\mathsf{u}}, \mathsf{A} = (A_1, A_2))$, to CA and runs $\mathsf{ZKPoK}(\rho, P^\rho)$ with CA.

CA checks if proofs are correct, then runs $(\sigma, (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'}) \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{A}, L', \mathsf{pk}_{\mathsf{u}}; (P^{\rho_j})_{j\in[2]})$, where $(\mathbf{C}, \mathbf{O}) \leftarrow \mathsf{CSCA.Commit}_2(A_i, \alpha, P^{\rho_j})_{i\in[2]}$ and $\mathsf{nym}_{\mathsf{u}} = \mathsf{pk}_{\mathsf{u}}$. Sets $\mathsf{dk}_{L'} = \mathsf{uk}_{k'}$ and outputs $(\sigma, \mathsf{dk}_{L'})$ as well as $(\mathbf{C}, \mathbf{O})$.

U checks if $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{u}}, \mathbf{C}, \sigma, (\mathbf{T}, \mathbf{U})) = 1 \wedge \Sigma.\mathsf{UKVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, L', \sigma) = 1$, where $(\mathbf{T}, \mathbf{U}) = (\mathsf{A}, \mathbf{O})$, sets $\mathbf{O} = \rho$ and runs $(\sigma', (\mathbf{C}', \mathbf{O}'), \mathsf{dk}'_{L'}, \mathsf{nym}'_{\mathsf{u}}, \chi) \leftarrow \Sigma.\mathsf{ChangRep}(\mathsf{nym}_{\mathsf{u}}, \mathsf{dk}_{L'}, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi)$ for $\mu, \psi \leftarrow \mathbb{Z}_p^*$, update $\mathsf{aux}_{\mathsf{u}}$ with χ, ψ and saves a credential $\mathsf{cred}_{\mathsf{u}} = \sigma'$, as well as $(\mathsf{dk}'_{L'}, \mathbf{C}', \mathbf{O}')$.

Issuing/delegating a credential (an issuer U and a receiver R)

$[\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, \mathsf{sk}_{\mathsf{u}}, \mathsf{cred}_{\mathsf{u}}, A_l, L'') \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{r}}, A_l)] \rightarrow (\mathsf{cred}_{\mathsf{r}}, \mathsf{dk}'_{L''})$:

On input $(\mathsf{nym}_{\mathsf{r}}, A_l, \mathsf{cred}_{\mathsf{u}} = \sigma, \mathsf{dk}_{L'}, (\mathsf{A}, \mathbf{C}, \mathbf{O}))$, U prepares a delegated credential for $\mathsf{nym}_{\mathsf{r}}$ and $\mathsf{A}' = (\mathsf{A}, A_l)$:

- Run $\sigma' \leftarrow \Sigma.\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_{\mathsf{u}}, \mathsf{sk}_{\mathsf{u}'}, \sigma)$ with R, run $(\sigma'', (\mathbf{C}', O_l), \mathsf{uk}_{k''}) \leftarrow \Sigma.\mathsf{ChangeRel}(M_l, \sigma', \mathbf{C}, \mathsf{uk}_{k'}, L'')$, where L'' is a level, $\mathsf{uk}_{k'} = \mathsf{dk}_{L'}$, $\mathsf{uk}_{k''} = \mathsf{dk}_{L''}$ and $M_l = A_l$ for $\mathbf{C}' = (\mathbf{C}, C_l)$.
- Send $(\sigma'', \mathbf{C}', \mathbf{O}' = (\mathbf{O}, O_l))$ and optionally $\mathsf{dk}_{L''} = \mathsf{uk}_{k''}$ to R.

R checks $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{r}}, \mathbf{C}', \sigma'', (\mathsf{A}', \mathbf{O}')) = 1$, then for $\mu, \psi \leftarrow \mathbb{Z}_p^*$ runs $(\sigma''', (\mathbf{C}'', \mathbf{O}''), \mathsf{dk}'_{L''}, \mathsf{pk}'_{\mathsf{u}}, \chi) \leftarrow \Sigma.\mathsf{ChangRep}(\mathsf{nym}_{\mathsf{r}}, \mathsf{dk}_{L''}, (\mathbf{C}', \mathbf{O}'), \sigma'', \mu, \psi)$ and updates $\mathsf{aux}_{\mathsf{r}}$ with χ, ψ and stores $(\mathsf{cred}_{\mathsf{r}} = \sigma''', (\mathbf{C}'', \mathbf{O}''))$ and the delegatable key $\mathsf{dk}'_{L''}$.

Showing of a credential (a prover P and a verifier

$\mathsf{V})$ $\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{nym}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}, \mathsf{cred}_{\mathsf{P}}, D) \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{P}}, D) \rightarrow (0, 1)$:

On input a credential $\sigma = \mathsf{cred}_{\mathsf{r}}$ for $\mathsf{nym}_{\mathsf{r}}$ and $D = \{d_j\}_{j\in[k]}$, P prepares a proof for $\mathsf{nym}_{\mathsf{P}}$ and $\{A_j\}_{j\in[k]}$:

- Run $(\sigma', (\mathbf{C}', \mathbf{O}'), \mathsf{pk}'_{\mathsf{u}}, \chi) \leftarrow \Sigma.\mathsf{ChangRep}(\mathsf{pk}_{\mathsf{u}}, \perp, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi)$ for $\mu, \psi \leftarrow \mathbb{Z}_p^*$, where $\sigma = \mathsf{cred}_{\mathsf{r}}$ and $\mathsf{pk}_{\mathsf{u}} = \mathsf{nym}_{\mathsf{r}}$. Set $\sigma' = \mathsf{cred}_{\mathsf{P}}$ and $\mathsf{pk}'_{\mathsf{u}} = \mathsf{nym}_{\mathsf{P}}$, and update $\mathsf{aux}_{\mathsf{P}}$ with ψ, χ.
- Run $W_j \leftarrow \mathsf{CSCA.OpenSubset}(C_j, A_j, O_j, d_j)$ for $j \in [k]$. Aggregate witness $\pi \leftarrow \mathsf{CSCA.AggregateAcross}(\{C_j, d_j, W_j\}_{j\in[k]})$.
- Send $(\mathsf{cred}_{\mathsf{P}}, \mathbf{C}', \mathsf{nym}_{\mathsf{P}}, \pi)$ to V, and run $\mathsf{ZKPoK}((\mathsf{sk}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}), \mathsf{nym}_{\mathsf{P}})$ with V.

V outputs 1, if $\mathsf{ZKPoK}((\mathsf{sk}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}), \mathsf{nym}_{\mathsf{P}})$ verifies and $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{u}}, \mathbf{C}, \sigma, (T, \mathbf{U})) = 1$, where $\sigma = \mathsf{cred}_{\mathsf{P}}$, $T = D$, $\mathbf{U} = \pi$ and $\mathsf{pk}_{\mathsf{u}} = \mathsf{nym}_{\mathsf{P}}$. Else output 0.

Figure 4.3: Our DAC scheme (Σ denotes our SPSEQ-UC scheme from Section 4.4.3).

the security proofs. Thus we opted to rely on interactive ZKPoK*. We however note that when willing to pay some extra costs, one could instead use straight-line extractable* NIZK*, e.g., obtained via Fischlin's transformation [Fis05].*

The correctness of Scheme follows by inspection. Unforgeability follows from unforge-

ability of SPSEQ-UC, while anonymity follows from class and origin hiding. Proofs are based AC scheme has been predestined in [FHS19]. More precisely, we prove the following:

Lemma 4.6.1 (Unforgeability)**.** *Let* ZKPoK *be a ZKPoK and let* SPSEQ-UC *be unforgeable, then the* DAC *construction in Fig 4.3 is unforgeable.*

Proof. We show that an adversary performing an incompatible showing for a dishonest user can be used to forge an SPSEQ-UC signature similar to [FHS19]. Assume a PPT adversary $\mathcal{A}$ that wins the unforgeability game with non-negligible probability and let $(\mathbf{C}^*, \mathsf{nym}_P^*, \mathsf{A}^*, \sigma^*)$ be the message-signature pair it uses and $\mathbf{W}^*$ be the witness for an attribute set $(D^*, \mathsf{A}') \notin \mathcal{R}_{k'}$ (this implies $D^* \not\subseteq \mathsf{A}'$), for all $i = \perp$ where $i, \mathsf{A}', \mathsf{dk}_{k'} \in \mathcal{L}_{\mathsf{cred}}$; moreover, the $\mathsf{ZKPoK}(\mathsf{sk}_{\mathsf{P}}^*, \mathsf{nym}_p^*)$ verifies. We construct an adversary $\mathcal{B}$ that breaks the unforgeability of SPSEQ-UC. We note that we extract from ZKPoK and assume this will only fail with negligible probability. Then, we are ready to reduce to the unforgeability of SPSEQ-UC:

Reduction. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in the unforgeability game for SPSEQ-UC and $\mathcal{B}$ simulates the DAC-unforgeability game for $\mathcal{A}$. $\mathcal{C}$ runs $(\mathsf{pp}, \mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}}) \leftarrow \mathsf{Setup}(1^\lambda, 1^t, 1^\ell)$ and gives $(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pp})$ to $\mathcal{B}$. Then, $\mathcal{B}$ sets $\mathsf{pp} = \mathsf{pp}_{\mathsf{SPSEQ\text{-}UC}}, \mathsf{vk} = \mathsf{pk}_{\mathsf{CA}}$ and sends them to $\mathcal{A}$. It next simulates the oracles. All oracles run as in the real game, except for the oracles that use the signing oracle instead of the $\mathsf{sk}_{\mathsf{CA}}$ key.

- When the oracles $\mathcal{O}^{\mathsf{Corrupt}}$ and $\mathcal{O}^{\mathsf{User}}$ are called, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{Corrupt}}$ and $\mathcal{O}^{\mathsf{Create}}$ of the SPSEQ-UC scheme, respectively. Note that when $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{Corrupt}}$ of the SPSEQ-UC, it gets sk_i and finally returns sk_i and all the associated credentials items to A.

- $\mathcal{O}^{\mathsf{CreateRoot}}(i, k', \mathsf{A})$: On input a user identity i, an index k', and an attribute vector A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it picks ρ for an attributes vector. Then it submits $(\mathsf{nym}_i, k', \mathsf{A}, \rho)$ to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$. Receives a signature $(\sigma = (Z, Y, \hat{Y}, T), (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'})$. It sets $\sigma = \mathsf{cred}_i, \mathsf{uk}_{k'} = \mathsf{dk}_{k'}$ and appends $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$.

- $\mathcal{O}^{\mathsf{RootIss}}(k', \mathsf{A})$: On input an index k' and an attribute vector A. It extracts ρ from the proof of knowledge $\mathsf{ZKPoK}(\rho, P^\rho)$ produced by $\mathcal{A}$ for an attributes vector. Then it submits $(\mathsf{nym}_i, k', \mathsf{A}, \rho)$ to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$, where nym_i is an adversary pseudonym of a corrupted user. Receives a signature $(\sigma = (Z, Y, \hat{Y}, T), (\mathbf{C}, \mathbf{O}), \mathsf{uk}_{k'})$. It sets $(\sigma, \mathbf{C}, \mathbf{O}, \mathsf{nym}_i) = \mathsf{cred}_i, \mathsf{uk}_{k'} = \mathsf{dk}_{k'}$ and appends $(\perp, \mathsf{A}, k', \perp)$ to $\mathcal{L}_{\mathsf{cred}}$ and outputs the results.

- The oracles $(\mathcal{O}^{\mathsf{Issue}}, \mathcal{O}^{\mathsf{ObtIss}})$: In both $\mathcal{O}^{\mathsf{ObtIss}}$ and $\mathcal{O}^{\mathsf{Issue}}$, all executions of ChangeRel and ConvertSig for credentials $(i, \mathsf{dk}_{k'}, \mathsf{A}', \sigma_i) \in \mathcal{L}_{\mathsf{cred}}$ are replaced by the oracle $\mathsf{Sign}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{A}', k', \mathsf{pk}_i, \rho)$, where $\mathsf{pk}_i = \mathsf{nym}_i = 1$ for the $\mathcal{O}^{\mathsf{Issue}}$ and $\mathsf{pk}_i = \mathsf{nym}_j$ for the $\mathcal{O}^{\mathsf{ObtIss}}$.

As it is clear, $\mathcal{B}$ can handle any oracle query. So, at the end of the game, $\mathcal{B}$ simulates all oracles perfectly for $\mathcal{A}$ who is able to prove possession of a credential on A^*. To do this, $\mathcal{B}$ interacts with $\mathcal{A}$ as verifier in a showing protocol. If $\mathcal{A}$ outputs a valid showing proof as $(\mathbf{C}^*, \sigma^* = (Z^*, Y^*, \hat{Y}^*, T^*), D^*, \mathsf{nym}_p^*, \mathbf{W}^*)$ and conducting $\mathsf{ZKPoK}(\mathsf{sk}_\mathsf{P}^*, \mathsf{nym}_p^*)$ then $\mathcal{B}$ extracts from the proof ZKPoK the value sk_p^* related to the nym_P^* and stores all elements. Moreover, the credential of corrupt users can not be valid on this set of messages D^* (as $\mathcal{A}$ can win the unforgeability game). Thus, for any credential on (sk_i) with all $i = \perp$, we have $(D^*, \mathsf{A}') \notin \mathcal{R}_{k'}$. In all cases, this means that $(\mathsf{sk}_p^*, (\mathbf{C}^*, D^*, \mathbf{W}^*), \sigma^*)$ is a valid forgery against our signature scheme, $\mathcal{B}$ breaks thus unforgeability of $\mathsf{SPSEQ\text{-}UC}$ which concludes our proof.

Lemma 4.6.2 (Anonymity)**.** *Let* ZKPoK *be a ZKPoK,* NIZK *be knowledge sound, the* DDH *assumption holds and the* $\mathsf{SPSEQ\text{-}UC}$ *provides Origin-hiding, Conversion-privacy and Derivation-privacy, then the* DAC *construction in Fig 4.3 is anonymous.*

Proof. The proof is similar to [FHS19], but we adapted it with our setting. It follows a sequence of games until a game where the answers for the query to $\mathcal{O}_b^{\mathsf{Anon}}$ is independent of the bit b. In $\mathbf{Game}_1$ we use the knowledge soundness of NIZK to extract the signing key. Then, in $\mathbf{Game}_2$ we replace all $\mathsf{ChangRep}$, $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ calls with freshly generated signatures. In $\mathbf{Game}_3$ we simulate all ZKPoKs and in $\mathbf{Game}_4$ we guess a user to be asked in $\mathcal{O}_b^{\mathsf{Anon}}$. Finally, in $\mathbf{Game}_5$ we replace the respective commitment vectors with random vectors.

$\mathbf{Game}_0$: The original game as given in Definition 56.

$\mathbf{Game}_1$: Like Game_0, except when A outputs the pk_CA and corresponding $\mathsf{NIZK}(\mathsf{sk}_\mathsf{CA}, \mathsf{pk}_\mathsf{CA})$, the game runs the knowledge extractor for NIZK, which extracts a witness $((x_i)_{i\in[0,\ell]}, \alpha)$ sets them as sk_CA including the SC trapdoor. We stop in the case that the extractor fails.

$\mathsf{Game}_0 \to \mathsf{Game}_1$: The success probability in Game_1 is the same as in Game_0, unless the extractor fails, i.e., using knowledge soundness we have

$$|\Pr[S_0] - \Pr[S_1]| \leq \epsilon_{ks}(\lambda).$$

$\mathbf{Game}_2$: As Game_1, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: Like in Game_1, but for $\mu, \psi \in Z_p^*$, all executions of $\mathsf{ChangRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\mathbf{C}, \mathbf{O}), \sigma, \mu, \psi)$ for the credential $(i_b, \mathsf{dk}_{k'}, \mathsf{A}', \sigma_b) \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_b]$ are replaced by $\mathsf{Sign}(\mathsf{sk}_\mathsf{CA}, \mathsf{A}', k', \mathsf{pk}_u; \rho)$. Oracles are simulated as in Game_1 , except for the following oracles as:

- $\mathcal{O}^{\mathsf{ObtIss}ObtIss}$: all executions of $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ for credentials $(j, \mathsf{dk}_{k'}, \mathsf{A}', \sigma_j) \in \mathcal{L}_{\mathsf{cred}}$ are replaced by $\mathsf{Sign}(\mathsf{sk}_\mathsf{CA}, \mathsf{A}', k', \mathsf{pk}_j, \rho)$.

$\mathsf{Game}_2 \to \mathsf{Game}_1$: By Origin-hiding, Derivation-privacy and Conversion-privacy, replacing signatures from $\mathsf{ChangRep}$, $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ with ones from Sign are identically distributed for all $(\mathsf{A}, \mathbf{C})$. We thus have

$$\Pr[S_1] = \Pr[S_2]$$

Game_3: As Game_2, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: All proofs $\mathsf{ZKPoK}(\mathsf{sk}_p, \mathsf{nym}_p)$ and $\mathsf{ZKPoK}(\rho, P^\rho)$ in $\mathsf{CredProve}$ and $\mathsf{CreateCred}$ respectively, are simulated.

$\mathsf{Game}_2 \to \mathsf{Game}_3$: By perfect zero-knowledge of ZKPoK, we have that

$$\Pr[S_2] = \Pr[S_3] \Rightarrow \Pr[S_1] = \Pr[S_2] = \Pr[S_3]$$

Game_4: Same as Game_3, with the following changes. Let q_u be the maximum number of queries to $\mathcal{O}^{\mathsf{User}}$. At the start, during Game_4, a selection is made where ω is chosen randomly from the set $[q_u]$ (i.e., guessing that the user with the j_b credential is registered at the ω-th call to $\mathcal{O}^{\mathsf{User}}$) and runs $\mathcal{O}^{\mathsf{User}}, \mathcal{O}^{\mathsf{Corrupt}}$ and $\mathcal{O}_b^{\mathsf{Anon}}$ as follows:

- $\mathcal{O}^{\mathsf{User}}(i)$: Like Game_3, but if the call of this oracle is the ω-th call then it sets $i^* \leftarrow i$ as well.
- $\mathcal{O}^{\mathsf{Corrupt}}(i, \mathsf{pk}_u)$: Like Game_3, but it returns $\perp$, if $i \in \mathcal{CU}$ or $i \in \mathcal{O}_b^{\mathsf{Anon}}$. Also If $i = i^*$, stops and returns $b' \leftarrow \{0,1\}$. If $i \in \mathcal{HU}$, the algorithm outputs the user's secret key sk_i and credentials, and then transfers i from the $\mathcal{HU}$ to the $\mathcal{CU}$. On the other hand, if $i \notin \mathcal{HU} \cup \mathcal{CU}$, the algorithm registers a new corrupt user i, assigns them the public key pk_i, and adds i to the set $\mathcal{CU}$.
- $\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)$: As in Game_3, but if $i^* \neq i_b \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_b]$, it stops and outputs $b' \leftarrow \{0,1\}$.

$\mathsf{Game}_3 \to \mathsf{Game}_4$: By assumption, $\mathcal{O}_b^{\mathsf{Anon}}$ is called at least once with some input $(\mathsf{j}_0, \mathsf{j}_1, D)$ such that $i_0 \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_0], i_1 \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_1] \in \mathcal{HU}$. If $i^* = i_b$ then $\mathcal{O}_b^{\mathsf{Anon}}$ does not abort and neither does $\mathcal{O}^{\mathsf{Corrupt}}$. Since $i^* = [i_b]$ with probability $\frac{1}{q_u}$, so the probability of the experiment not aborting is greater than or equal to $\frac{1}{q_u}$. and thus

$$\Pr[S_4] \geq (1 - \frac{1}{q_u})\frac{1}{2} + \frac{1}{q_u} \cdot \Pr[S_3]$$

Note that Before the call to $\mathcal{O}b^{\mathsf{Anon}}$, it is ensured that i_b has not been previously called (otherwise, $i_b \notin \mathcal{HU}$). If $\mathcal{O}b^{\mathsf{Anon}}$ is called afterwards, it returns $\perp$, where $i^* \in \mathcal{O}_b^{\mathsf{Anon}}$.

Game_5: As Game_4, except that for $\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)$: it picks $\mathbf{C} \leftarrow (\mathbb{G}_1^*)^k$ and performs the showing using $\mathsf{cred}' \leftarrow (\mathbf{C}, \mathsf{Sign}(\mathsf{sk}, \mathbf{M}, \ldots))$, with $D = (d_i)_{i\in[k]}$ and $W_i \leftarrow f_{d_i}(a)^{-1} \cdot C_i$ for $i \in [k]$. The only difference is the selection of $\mathbf{C}$; while $\mathbf{W}$ is distributed as in Game_4, in particular, they are unique elements satisfying $\mathsf{VerifySubset}(C_i, D_i, W_i)_{i\in[k]}$.

$\mathsf{Game}_4 \to \mathsf{Game}_5$: Let $(\mathsf{BG}, P^x, P^y, P^z)$ be a DDH instance (not to be confused with $\mathsf{SPSEQ\text{-}UC}$ elements) for $\mathsf{BG} = \mathsf{BGGen}(1^\lambda)$ where $x, y \leftarrow Z_p$ and Z is equal to $P^{x\cdot y}$ or a random element. The extended version of DDH that we consider here is given by $(P, P^{x_1}, \ldots, P^{x_k}, P^y, Z_1, \ldots, Z_k)$ where $Z_i = P^{x_i \cdot y}$ or random for all $\in \{1, \ldots, k\}$. One can easily show that this extended version of DDH follows from DDH itself (with some polynomial security loss) as long as k is a polynomial. Oracles are simulated as in Game_4, except for the following oracles as:

- $\mathcal{O}^{\mathsf{ObtIss}}(i, \mathsf{A})$: Like Game_4, except for the following values when $i = i^*$. Since $\alpha \notin \mathsf{A}$, it computes $C_i \leftarrow f_{A_i}(a) \cdot P^{x_i}$ for $A_i \in \mathsf{A}$ (all C_i are thus distributed as in the real game.)

- $\mathcal{O}^{\mathsf{CredProve}}(j, D)$: As in Game_4, with the difference that if $i^* = i \leftarrow \mathcal{L}_{\mathsf{cred}}[j]$, it computes the witness $W_i \leftarrow f_{A_i/d_i}(a)^{\mu} \cdot P^{x_i}$. ($W_i$ is thus distributed as in the real game and $D = (d_i)_{i \in [k]}$.)

- $\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)$: As in Game_4, with the difference as follows: We use DDH self-reducibility. It selects $s, t \leftarrow Z_p^*$ and computes $Y' \leftarrow P^{t \cdot y} \cdot P^s = P^{y'}$ with $y' \leftarrow t \cdot y + s$, and $Z_i' \leftarrow P^{t \cdot z_i} \cdot P^{s \cdot x_i} = P^{(t(z_i - x_i \cdot y) + x_i \cdot y')}$. (If $z_i \neq x_i \cdot y$ then Y' and Z_i' are random; else $Z_i' = y' \cdot X_i$) Finally, it performs the showing using values: $C_i \leftarrow f_{A_i}(a) \cdot Z_i'$ and $W_i \leftarrow f_{A_i/d_i}(a)^{-1} \cdot C_i$. We assume $a \notin D$.

With a negligible probability of an error event, we successfully simulate Game_4 when the DDH instance is "real" and Game_5 otherwise. During the simulation of $\mathcal{O}b^{\mathsf{Anon}}$, if any $Y_i' = 0_{\mathbb{G}_1}$ or $Z_i' = 0_{\mathbb{G}_1}$, the distribution of values deviates from either of the two games. However, if $Y_i' \neq 0_{\mathbb{G}_1}$ and $Z_i' \neq 0_{\mathbb{G}_1}$, we implicitly set $\rho_i \leftarrow x_i$ and $\mu \leftarrow y'$ (with a fresh value y' at each call of $\mathcal{O}b^{\mathsf{Anon}}$). For a DDH instance, we have $C_i \leftarrow (P^{f_{A_i}(a)})^{\rho_i \cdot \mu}$ for all C_i; otherwise, all C_i are independently random. Assume the probability of solving DDH problem is $\epsilon_{\mathsf{DDH}}(\lambda)$, and let q_l be the number of queries to the $\mathcal{O}_b^{\mathsf{Anon}}$. Then we have:

$$|\Pr[S_4] - \Pr[S_5]| \leq \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p}$$

In Game_5 the $\mathcal{O}_b^{\mathsf{Anon}}$ returns a freshly generated signature σ on random values $\mathbf{C} \leftarrow (\mathbb{G}_1^*)^k$ and a simulated proof. The bit b is thus information-theoretically hidden from $\mathcal{A}$, so we have $\Pr[S_5] = \frac{1}{2}$. From this and the above equations we have

$$\begin{aligned}
\Pr[S_4] &\leq \Pr[S_5] + \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p} = \frac{1}{2} + \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p}, \\
\Pr[S_3] &\leq \frac{1}{2} + q_u \cdot \Pr[S_4] - \frac{1}{2} \cdot q_u \leq \frac{1}{2} + q_u \cdot (\epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p}), \\
\Pr[S_0] &\leq \Pr[S_1] + ks(\lambda) \leq \frac{1}{2} + ks(\lambda) + q_u \cdot (\epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p})
\end{aligned}$$

where $\Pr[S_1] = \Pr[S_3]$. Assuming security of ZKPoK, NIZK and DDH, this advantage is very small. Note that q_u, q_o and q_l are the number of queries to $\mathcal{O}^{\mathsf{user}}$, $\mathcal{O}^{\mathsf{Obtain}}$ and the $\mathcal{O}_b^{\mathsf{Anon}}$ oracle, respectively.

4.7 Implementation and Evaluation

We now present an evaluation of our our SPSEQ-UC and DAC schemes implement as a Python library [5]. Our implementation is based upon the `bplib`library[6] and petlib[7] with `OpenSSL` bindings[8]. It uses the BN256 curve, providing efficient type 3 pairings at around 100 bit security. We also want to point to an independent Rust implementation of our scheme[9].

We present a benchmark of SPSEQ-UC (including the cross-commitment aggregation provided by the CSCA scheme) and the DAC scheme described in Section 4.6. DAC is based upon Schnorr-style discrete-logarithm zero-knowledge proofs and using Damgard's technique [CDM00] for obtaining malicious-verifier interactive zero-knowledge proofs of knowledge during the showing and issuing/delegating of credentials. It also uses NIZK proofs obtained via the Fiat-Shamir heuristic for proofs of knowledge of $\mathsf{pk}_{\mathsf{CA}}$. Our measurements have been performed on an Intel Core i5-6200U CPU at 2.30 GHz, 16 GB RAM running Ubuntu 20.04.3. For our evaluation, we take the execution time of each algorithm for the following parameters: ℓ represents an upper bound for the length of commitment vector, t an upper bound for the cardinality of committed sets, $n_i < t$ is the number of attributes in each attribute set A_i in the respective commitment C_i of the commitment vector, which we set to be the same for every level (each commitment) for simplicity. Moreover, k represents the length of attribute set vector $\mathsf{A} = (A_1, \ldots, A_k)$ and thus commitment vector $\mathbf{C}$. The number of attribute sets which can be delegated is k'. The results are shown in Table 6.2,

Table 4.2: Running times for SPSEQ-UC and DAC (ms)

	SPSEQ-UC								DAC			
	Setup	*KeyGen*	*Sign*	*ChangRep/uk*	*ChangRep*	*ChangeRel*	*ConvertSig*	*Verify*	*CreateCred*	*DelegIssue*	*CredProve*	*CredVerify*
μ_{AV}	385	21	73	50	6	15	2	38	82	21	35	195
SD	±3	±1	±2	±1	±2	±2	±1	±1	±1	±2	±2	±3

where μ_{AV} is the mean and $\mathsf{S}D$ the standard deviation of 100 executions of each algorithm. Note that in each delegation one additional attribute set is added. More precisely, in Table 6.2, we set the above parameters as $t = 25$, $\ell = 15$, $k = 4$, $k' = 7$ and $n = 10$ to cover a broad spectrum of applications. ChangRep/uk represents the randomization of a signature along with $\mathsf{uk}_{k'}$ and ChangRep represents the randomization of signatures only. In the CredProve and CredVerify protocols, we use d_i to denote the subset of each attribute set

[5] https://github.com/mir-omid/DAC-from-EQS
[6] https://github.com/gdanezis/bplib
[7] https://github.com/gdanezis/petlib
[8] https://github.com/dfaranha/OpenPairing
[9] https://github.com/docknetwork/crypto/tree/main/delegatable_credentials.

A_i that will be disclosed and D all d_i's, i.e., $D = (d_i)_{i\in[k]}$. We thereby assume that each subset d_i contains approximately half of the attributes in A_i. Let us provide an example to clarify our notation: Assume $k = 2$, we have two commitments C_1, C_2 (for simplicity we can say each commitment is the output of one delegation level e.g., $L = 2$ is the delegation level in DAC) such that each includes 5 attributes. Then $D = (d_1, d_2)$ means that each d_1 and d_2 includes two attributes of sets of C_1 and C_2, respectively and $|D| = 4$. By that, we say the total number of attributes is $|\mathbf{M}| = |\mathsf{A}| = k \cdot n$. We use this semantics in Fig 4.4 and 4.5.

In Fig 4.4 we show the effect on the computation time of SPSEQ-UC when increasing the parameters (k, k', n). Since the Setup algorithm runs only once, we do not consider the computation time of Setup. We measure the computation time for a message (or an attribute) set of size n from 5 to 20, k from 2 to 10 (so that the total messages $|\mathbf{M}|$ is from 10 to 200), and k' from 4 to 20. Since the runtime of the algorithms KeyGen, ChangRep, and ConvertSig is independent of the parameters (k, k', n) we omit them.

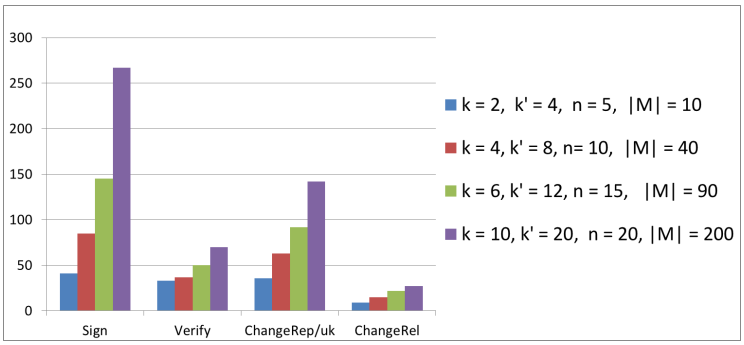
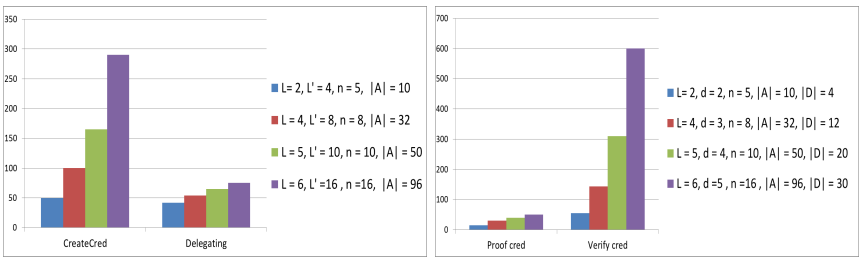


Figure 4.4: The running times of SPSEQ-UC (ms)

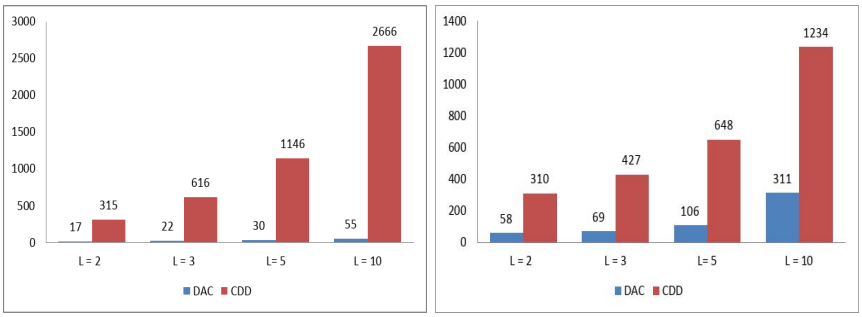
In Fig 4.5 we show the effect of increasing the parameters (k, k', d, n) on the computation time of DAC such that $k = L$ and $k' = L'$ here mean for levels. We measure the computation time of CredProve and CredVerify protocols for n, L and d (a disclosed attribute subset) varying from 5 to 16, 2 to 6 and 2 to 5, respectively. The total disclosed attributes length $|D| = d \cdot L$ and the total attributes length $|\mathsf{A}| = n \cdot L$ range from 4 to 30 and 10 to 96, respectively. The CredProve and CredVerify are independent of L'. Meanwhile, the computation time of CreateCred and IssueCred change when L' varies from 4 to 16 in addition to being dependent on n and L. These algorithms are independent of d. As can be seen in Fig 4.5, the pairing product operations in the verification produce the largest overhead. Though, in absolute terms verification is still highly efficient. For example, it is less than a second, in the maximum parameters setting with almost 100 attributes and disclosing 30 attributes. This efficiency makes our implementation also suitable for time-critical applications like public transportation or ticketing.

Comparison with CDD [CDD17, BCET21b]. In Fig. 4.6, we present an approximate comparison of running time between our DAC and CDD, which has recently been



(a) The running times of IssueCred (b) The running times of CredProve

Figure 4.5: The running times of DAC (ms)



(a) The running times of CredProve (ms) (b) The running times of CredVerify (ms)

Figure 4.6: Comparison between our DAC and CDD (n = 4)

implemented[10] by [BCET21b]. Their implementation is in Go and, similar to ours, uses the BN256 curve. In [BCET21b] running times for proving and verifying credentials from benchmarks on a c2-standard-60 GCE VM running Ubuntu 18.04 (60 vCPUs, Intel Cascade Lake 3.1 GHz, 240 GB RAM) are provided. To make the approaches comparable, we set parameters as in [BCET21b] as follows: $n = 4$ attributes per level (the maximum in [BCET21b]) and L for levels from 2 to 10. We note that the machine used to obtain the benchmarks in [BCET21b] is much more powerful than the one used to benchmark our DAC. But we still significantly outperform CDD. We note that benchmarking them on the same platform will only increase the computational advantage of our approach.

4.7.1 Theoretical Analysis and Comparison

We now analyze and compare the computational and communication complexity of our DAC compared with Camenisch et al. [CDD17] (CDD), one of the most efficient and fully specified approaches.

[10] https://github.com/IBM/dac-lib

4.7.1.1 Computational Complexity

To analyze the efficiency of our DAC, we consider the number of (multi)-exponentiations required for the IssueCred (issuing or delegating), CredProve (the showing of a credential), and CredVerify (verifying a credential). We summarize the following efficiency analysis comparing our DAC and the CDD scheme [CDD17] in Table 5.2. We use notations that are used in [CDD17], where d_i and u_i denote the amount of disclosed and undisclosed attributes at delegation level i, respectively, such that $n_i = d_i + u_i$; and $X\{\mathbb{G}_1^j\}, X\{\mathbb{G}_2^j\}$, and $X\{\mathbb{G}_t^j\}$ denote X j-multi-exponentiations in the respective group; $j = 1$ denotes a simple exponentiation. E^k denote a k-pairing product with $k = 1$ denoting a single pairing. Assume $z = |[k+1, k']|$ and $k'' = k'$ (the worst case), where k is length of message $\mathbf{M}$ and commitment $\mathbf{C}$ vectors and $n < t$ is size of a message (attributes) set M s.t. M_i includes n_i messages. Moreover, we assume that a n_i number of elements for each $j \in [z]$ can be delegated and put into $\mathsf{uk}_{k'}$. We summarize the efficiency analysis as follows, where we also count the cost of the $\mathbf{C}$ randomization in ChangRep, but ignore the cost of proving knowledge of the secret key, as a Schnorr NIZK induces an insignificant cost:

CreateCred: CA creates the first level of the credential. To do this, CA runs Sign and a user runs ChangRep/$\mathsf{uk}_{k'}$:

$$\left(\left(\sum_{i=1}^{k}(\mathbb{G}_1^{n_i} + \mathbb{G}_1)\right) + \mathbb{G}_1^2 + \mathbb{G}_1 + \mathbb{G}_1^{k+1} + \sum_{i=1}^{z} n_i\mathbb{G}_1 + \mathbb{G}_2\right) + \left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2 + \sum_{i=1}^{z} n_i\mathbb{G}_1\right)$$

IssueCred: Delegation of a credential includes running the ConvertSig, ChangeRel, and ChangRep. We have:

$$\left(2\mathbb{G}_1^2\right) + \left(2(\mathbb{G}_1^n + \mathbb{G}_1) + \mathbb{G}_1^2\right) + \left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2\right).$$

If a further delegation is needed, then the randomization of $\mathsf{uk}_{k'}$ adds $\sum_{i=1}^{z} n_i\mathbb{G}_1$.

CredProve: Proving possession of a credential by a user includes ChangRep, AggregateAcross, and OpenSubset: Let $D = (d_i)_{i\in[k]}$, we have:

$$\left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2\right) + \left(\mathbb{G}_1^{|D|}\right) + \left(\sum_{i=1}^{|D|}(\mathbb{G}_1^{u_i} + \mathbb{G}_1)\right)$$

CredVerify: The credential verification includes SPSEQ-UC.Verify (using CSCA.VerifyAcross): Let $S = \bigcup_i d_i$, where $i \in [k]$, we have

$$\left(E^k + E^2 + 4E\right) + \left(E + E^k + \mathbb{G}_2^{|S|} + \sum_{i=1}^{|D|}\left(\mathbb{G}_2^{|S-d_i|} + \mathbb{G}_2\right)\right)$$

Table 4.3: Computational complexity

	CDD [CDD17]	Our DAC
CredProve	**odd:** $\sum_{i=1,3}^{L} 1\mathbb{G}_2 + (n_i+2)\mathbb{G}_1 + (1+d_i)\mathbb{G}_t^2$ $+(1+u_i)\mathbb{G}_t^3 + (2+n_i)\mathbb{G}_1^2$	$\left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2\right) +$
	even: $\sum_{i=2,4}^{L} 1\mathbb{G}_1 + (n_i+2)\mathbb{G}_2 + (1+d_i)\mathbb{G}_t^2$ $+(1+u_i)\mathbb{G}_t^3 + (2+n_i)\mathbb{G}_2^2$	$\left(\mathbb{G}_1^{\lvert D\rvert}\right) + \left(\sum_{i=1}^{\lvert D\rvert} (\mathbb{G}_1^{u_i} + \mathbb{G}_1)\right)$
CredVerify	$(1+d_1)E + (3+u_1+d_L)E^2 + u_L E^3$ $+(4+n_1+d_L)\{\mathbb{G}_t\}+$	$\left(E^k + E^2 + 4E\right) +$
	$\sum_{i=1}^{L}((1+d_i)E^2 + (1+u_i)E^3 + (1+d_i)\{\mathbb{G}_t\})$	$\left(E + E^k + \mathbb{G}_2^{\lvert S\rvert} + \sum_{i=1}^{\lvert D\rvert}\left(\mathbb{G}_2^{\lvert S-d_i\rvert} + \mathbb{G}_2\right)\right)$

Although we do not have a concept of delegation chain length, in order to compare our scheme with [CDD17], we assume that each commitment is the output of one delegation-level, e.g., if k = 2 such that $\mathbf{C} = (C_1, C_2)$, the $L = 2$ is delegation level. In other words, at each delegation level, we add one attribute set and the related commitment. Also, unlike [CDD17] where their underlying signature construction needs switching $\mathbb{G}_1$ and $\mathbb{G}_2$ of message space throughout, we do not need it. Note that *operations in* $\mathbb{G}_1$ *are faster than* $\mathbb{G}_2$ and also *the length of elements in* $\mathbb{G}_1$ *is smaller*. We only consider the number of (multi)exponentiations required to show a credential since this will be the most frequently executed operation. The result of CDD scheme is taken from Table 1 in [CDD17].

Credential size. The size only counts the cryptographic components of the credential; the metadata and attribute values are assumed to be the same for all systems. In particular, the credential size (σ, sk_p and pseudonym nym_p) in SPSEQ-UC is independent of the delegation chain length and number of attributes. In SPSEQ-UC, credentials have constant size which is four $\mathbb{G}_1$, one $\mathbb{G}_2$ and one $\mathbb{Z}_p$ element. Let $|\mathbb{G}_1| = |Z_p| = 256$ and $|\mathbb{G}_2| = 512$ in bit, we have a size of 1792 bits. While, in CDD the credential size grows linearly with the number of attributes and delegation levels. Also, the size of the related $\mathsf{uk}_{k'}$ in our scheme is $z \cdot 256$.

4.7.1.2 Communication Complexity

We analyze the communication complexity and the size of each element exchange involved in DAC. More precisely, the IssueCred protocol depends on the number of keys in $\mathsf{uk}_{k'}$ (if delegation is requested), while the CredProveprotocol is independent of the number of attributes, delegation levels and keys. In the CredProve, we have: $((k+5)|\mathbb{G}_1| + |\mathbb{G}_2| + Z_p)$, where k is the size of $\mathbf{C}$ (that is, the delegation level L) that we send for verification and Z_p with one $\mathbb{G}_1$ element that belong to the ZKPoK. In the IssueCred, we have $((k+3)|\mathbb{G}_1| + |\mathbb{G}_2| + k|Z_p| + z|\mathbb{G}_1|)$, where $z = |[k, k']|$ is the number of keys in this range. In CDD, this communication cost grows linearly with the number of attributes and delegation levels (see Table 5.3.) Here, for the CDD scheme, we take a proof generated from an *even Level-L* credential.

Table 4.4: Communication Complexity

Schemes	CredProve
Our DAC	$((k+5)\|\mathbb{G}_1\| + \|\mathbb{G}_2\| + Z_p)$
CDD [CDD17]	$(2L + \sum_{i=1,3}^{L-1}(n_i + u_i))\|\mathbb{G}_1\| + (2L - 1 + \sum_{i=2,4}^{L}(n_i + u_i))\|\mathbb{G}_2\| + 2Z_p$

4.8 Summary

In this chapter we first present a new primitive called structure-preserving signatures on equivalence classes on updatable commitments SPSEQ-UC in which one can sign vectors of set commitments that can be extended by additional set commitments. Moreover, signatures contain a user's public key, which can be switched. Second, we present an efficient delegatable anonymous credential scheme DAC that supports attributes, provides strong privacy under a reasonable corruption model, and allows the delegators to restrict further delegations. We show the practical efficiency of our DAC by presenting performance benchmarks based on an implementation.

5 Threshold Delegatable Anonymous Credentials

In order to avoid a single point of trust and failure, decentralized AC systems have been proposed. They eliminate the need for a trusted credential issuer by replacing it with a public append-only ledger, e.g., a blockchain. Another type of decentralized system that does not rely on a public append-only ledger but reduces necessary trust in single systems can be constructed using a set of credential issuers that issue credentials in a threshold manner (e.g., t out of n). In this chapter, we present a novel AC system with such threshold issuance that additionally provides credential delegation and thus represents the first decentralized *and* delegatable AC system. We provide a rigorous formal framework for such threshold delegatable anonymous credentials (TDAC's). Our concrete approach departs from previous delegatable ACs and is inspired by the concept of predicate encryption and in particular functional credentials. More precisely, we propose a threshold delegatable subset predicate encryption (TDSPE) scheme, in which partial decryption keys are issued in a threshold way by multiple authorities and from which users then can generate full decryption keys. Finally, we use TDSPE to construct a TDAC scheme and present a comparison with previous work and performance benchmarks based on a prototype implementation.

5.1 High Level Idea of Our Approach

Our construction is inspired by the template of designing functional credentials (FCs) by Deuber et al. in [DMM+18], which departs from the common approach of designing ACs. We recall that the common and most prevalent approach blindly issues signatures on attributes and represents a zero-knowledge proof of a respective signature (or a re-randomized signature along with a suitable zero-knowledge proof) [CL03, CL04]. In FC and consequently in our approach, instead of a signature, the credential is a decryption key, and the verifier encrypts some random challenge using an attribute-policy, akin to ciphertext-policy attribute-based encryption (CP-ABE) [BSW07]. If a user is then able to decrypt the ciphertext correctly (using its credential) and to return the correct challenge, it implicitly demonstrates the possession of the required attributes such that the used policy in the ciphertext is satisfied and thus passes the authentication. To circumvent the requirement of proving the correctness of encryption using a zero-knowledge proof to deal with a malicious verifier, which can incur a significant performance penalty, Deuber et al. [DMM+18] proposed to rely on a generic approach due to Brandt et al. [BDLP98],

which we also rely on. It requires commitments to the randomness of encryption (by the verifier) and the result (by the user) and two additional moves opening these commitments. As argued by Deuber et al. [DMM+18], this additional communication is in most settings a reasonable trade-off for improved efficiency. However, in contrast to FCs which are generically built from *any* predicate encryption scheme, we focus on efficient realizations for the flexible class of subset predicates and are also interested in their practical performance validated by an implementation. Moreover, while Deuber et al. focus on the verification to be delegated to designated verifiers by computing delegation tokens for specific predicates, we realize a full featured delegation in that credentials can be delegated to other users and they can use them with all potential verifiers in an unlimited way. Finally, as already outlined above, our construction like Coconut [SAB+19] supports threshold issuing of credentials. Nevertheless, Coconut does not support delegation.

In Figure 5.1 we present a very high level schematic overview of our approach. In particular, ① there are n issuers who run a distributed key generation algorithm with a threshold $k < n$ in the setup phase. Then, ② a user engages with at least k of these issuers in an issuing protocol to obtain partial credentials for his attribute set, where the issuers do not need to interact. The issuers may also give the user the power for further delegating credentials. In this example, this is the case, and the user turns into a delegator (an intermediate issuer). Then, ③ the user combines (aggregates) their partial credentials into

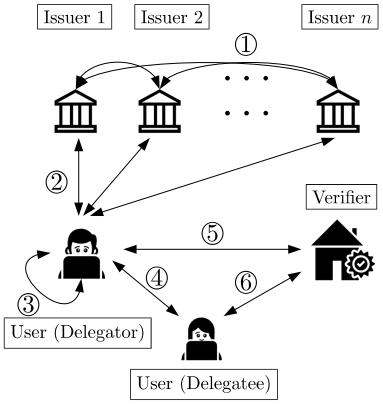


Figure 5.1: High-level overview of our approach.

a single credential. Since the user represents a delegator, ④ the credential can be further delegated to other users (the delegatees). Thereby, the delegator may decide whether the delegatees can also turn into delegators. We note that giving out delegation capabilities clearly delegates quite some power and care has to be taken when choosing delegators (e.g., there should be some trust in the delegator parties). However, we realize a fine-grained and controlled delegation mechanism (as discussed below), that allows to restrict the power of the delegator. Finally, ⑤ delegators as well as ⑥ delegatees can then engage in showing protocols with any verifier thereby demonstrating knowledge of subsets of the attributes

associated to their credentials. These selective showings do not reveal anything beyond the shown attributes and multiple showings of any user as well as all their delegatees are unlinkable.

Highly controlled, fine-grained delegation: Apart from the efficiency problem that the traditional DAC schemes [CDD17, CL19, BCC+09, Fuc11, CKLM14a] face, the main difference between the traditional DAC delegation model and our delegation model is that the traditional model does not allow issuers to restrict the use and delegation of credentials, i.e., they do not support the concept of controlled delegation. Indeed in the traditional setting the root authority gives all the power to the intermediate authorities without having any control over the further delegation chain, and it is not clear how the root can restrict this power. For example, if traditional DAC schemes allow the integration of attributes, delegation can only extend the set of attributes, but not restrict it during delegation. This fine-grained control of delegation is a feature missing in DAC schemes. While restricting the ability of the issuer in a credential scheme has been proposed in [BJS10], they consider the traditional setting of authorization and not the DAC setting with a focus on privacy guarantees. In contrast our approach allows for restricting the delegation in privacy-preserving DAC schemes.

Consequently, an inherent drawback of traditional DAC is that it is hard for root issuers to predict all the consequences of their credentials, i.e., how it could be further delegated. Thus, what is encoded in the issuer credential could have unexpected consequences, potentially leading to security breaches and, in general, policies that are more permissible than intended. For instance, Alice may want to give Bob access to her home door (when she is on vacation), but only under the condition that if this access is re-delegated, it must have been re-delegated to someone like Charlie, who is both Alice's and Bob's friend or have been re-delegated to access the home at a specific time (cf. [BJS10] for other examples).

We therefore propose a novel method for allowing credential issuers to control how their credentials can be used. To accomplish this, we embed in credentials explicitly specified constraints as delegation keys that express the authorized uses of the credentials and are enforced using proof rules.

5.2 Practical Application Scenarios

While TDACs can be used in any application of ACs, the combination of decentralization and delegation makes them particularly suitable for scenarios reflecting a hierarchical structure. We discuss the use of TDAC with two specific but widely different use cases to motivate practical impact:

Decentralized hardware root of trust. In secure systems, a root of trust is established and agreed upon by participating stakeholders. The process begins by ensuring that only trusted firmware, operating systems, drivers, and application components are loaded onto the hardware system. Each component in the boot chain verifies the next one's validity,

ensuring a secure chain of trust. The TDAC system extends TPM methods to create a complete chain of trust for the involved systems. As part of the TPM system, Remote Attestation (RA) protocols prove to online services or remote parties that a system has booted in a trustworthy state. Attestation Identity Keys (AIK) unique to TPM are signed by Certificate Authorities (CA) to establish a single root of trust.

However, the use of unique AIKs raises concerns about user traceability, which Direct Anonymous Attestation (DAA) addresses. Both RA and DAA currently support the verification of a single system and rely on a centralized trust authority. To enhance the chain of trust, delegation is introduced to extend the chain of trust to the application level in hierarchical settings, such as IoT or critical infrastructure scenarios with trust relationships between devices.

Additionally, supporting threshold issuers (e.g., t out of n issuers) enables decentralization of CAs, eliminating a single point of failure. This is crucial for IoT and infrastructure scenarios, as automatic deployment and management are required. Threshold issuing enhances resilience by requiring collaboration between multiple systems, making it harder for attackers to compromise the entire chain. The proposed TDAC system supports both decentralization and delegation of chain of trust certificate issuance, at the same time. Controlled delegation allows easier separation of (offline, long-term) root keys from (online, short-term) issuing authorities limited, e.g., to firmware for only the existing product line or geographic region and delegating (restricted) capabilities to other applications executing on such a trusted operating system.

Physical access control. We consider the specific motivating example of a mass public transport organization[1]. Several operators (e.g., tourist offices or reseller agents) manage different zones (different regions of the transport system). Each zone includes some gates for accessing directions and platforms. Groups of managers (the organization) act as independent authorities for the whole access control system, with permissions delegated to local operators for each zone.

Managers regularly issue initial credentials such as (city, zone number, expiration time). Also, using delegation keys, managers specify for which gates and tickets operators are allowed to issue credentials. For example, they create the credential (London, zone 1, "the end of year") and delegation keys as {gates (1, 2), tickets (one-way, one-day, one-week)}, and send them to operator 1. This process is similarly done for other operators with different credentials. In this example, all tickets are valid until the end of year and must be used by then, and operator 1 does not have the right to issue an annual ticket or a ticket for gate 3.

Suppose a user buys a one-way ticket for zone 1, the operator will give her a credential for (London, zone 1, "the end of year", gate 1, one-way) which means this user has access to gate 1 for a one-way direction. This credential is valid until the end of year.

[1]This use case is directly transferable to companies with different subsidiaries, office parks, apartment blocks with common rooms/areas, etc. We use transport systems as a crucial example in terms of availability and controlled delegation properties.

We argue that, in addition to the clear benefits of delegation mapping to real-world hierarchical relationships, decentralized issuers (managers) are also crucial from an availability perspective. Public mass transport has significantly higher availability requirements. Threshold-based issuers can easily tolerate the (planned or unplanned) unavailability of a subset of trusted nodes and therefore support high availability guarantees for critical infrastructure. Note that threshold-based issuers can guarantee the robustness of infrastructures both in terms of availability ($n > 1$, i.e., preventing a single point of failure) and security against compromise ($t > 1$, i.e., preventing a single point of attack). While the latter may not seem so important for transport scenarios, it enhances security as a compromised issuer key (without thresholding) will require re-setup of the system. This issue is associated with a potentially huge economic loss and administrative overhead (for re-issuing tickets that are still valid). Moreover, other physical access control scenarios may require multiple cooperating issuers for granting new access as part of the standard process (e.g., bank safe access, server housing, etc.). TDAC can technically enforce such multi-stakeholder approval.

5.3 Threshold Delegatable Subset Predicate Encryption

We introduce the notion as well as a simple and efficient construction of threshold delegatable subset predicate encryption (TDSPE), which adds support of a multi-authority setting and the delegation of decryption key capabilities to the existing concept of subset predicate encryption [KMMS17].

5.3.1 Formal Definitions

In TDSPE, a message is encrypted with respect to a set Θ' from an unbounded universe $\mathcal{U}$ and the resulting ciphertext can be decrypted by a key that is associated with a set Θ if and only if $\Theta = \Theta'$. We stress that since we are in an unbounded setting, the original interpretation of subsets in SPE in [KMMS17] is not meaningful anymore (they treat finite sets), and in our case, the subset turns into equality. Indeed the subset operator in our setting is set equality, and we achieve a notion of subsets via explicit encoding used in our TDAC application (see our DNF example 5.3.1). Moreover, the issuing of a secret key runs in a distributed way and secret key holders can obtain delegation keys at level $L = 1$ (the first level) which allow them to further delegate the keys at level $L + 1$ (e.g., the second level $L = 2$) and thereby extending the set Θ (depending on the capabilities in the delegation). We note that in contrast to HPE [OT09], which is defined with respect to predicate vectors, we are working with sets and thus find it more natural to use the term delegatable instead of hierarchical (note that when designing attributes in a way that they reflect a hierarchy one can clearly also implement a hierarchy).

Definition 53. *A* TDSPE *scheme consists of a set of authorities, encrypter(s), decrypter(s) and is modeled by the following PPT algorithms:*

$\mathsf{Setup}(n, t, \lambda) \to (\mathsf{pp}, (sk_i, pk_i)_{i\in[n]})$: Takes as input a number of authorities n, a threshold t where $1 \leq t \leq n$, and a security parameter λ. It outputs the public parameters pp and $(sk_1, \ldots, sk_n)$ is a vector of n secret key shares and corresponding public encryption key shares $(pk_1, \ldots, pk_n)$. Authority i is given the secret key share sk_i and the encryption key share pk_i. pp will be an implicit input to all algorithms.

$\mathsf{AggKey}(pk_1, \ldots, pk_t) \to \mathsf{pk}$: Takes as input a subset of t public encryption key shares. It aggregates these public encryption key shares into a single consolidated public key pk, and eventually outputs this public encryption key pk. It is run once by anyone who wants to encrypt messages (an encrypter).

$\mathsf{Enc}(\mathsf{pk}, \Theta', L, M) \to CT$: Takes as input the public key pk, a set Θ', a level L and a plaintext M. It returns a ciphertext CT for a decrypter in level L.

$\mathsf{ShareKeyGen}(sk_i, \mathsf{ST}, \Theta, \Omega) \to (dk^i_\Theta, mk^i_\Omega)$: Takes as input a secret key sk_i of authority i in a subset ST of $[n]$, where $|\mathsf{ST}| \geq t$, a set Θ and a delegation set Ω. Each authority i outputs a partial decryption key dk^i_Θ along with a delegation key mk^i_Ω for $i \in \mathsf{ST}$.

$\mathsf{KeyGenComb}(\{(dk^i_\Theta, mk^i_\Omega)\}_{i\in\mathsf{ST}}) \to (dk_\Theta, mk_\Omega)$: Takes as input the partial decryption and delegation keys $\{(dk^i_\Theta, mk^i_\Omega)\}_{i\in\mathsf{ST}}$ from authorities in the subset ST. A decrypter $(L = 1)$ aggregates keys into single consolidated decryption and delegation keys and outputs the full key dk_Θ and delegation key mk_Ω.

$\mathsf{Dec}(dk_\Theta, CT) \to \{M, \perp\}$: Takes as input the decryption key dk_Θ for the set Θ and the ciphertext CT for the set Θ'. It outputs either a message M or the symbol $\perp$.

$\mathsf{Delegate}(dk_\Theta, mk_\Omega, \hat{\Theta}) \to (dk_{\Theta\cup\hat{\Theta}}, mk_{\hat{\Omega}})$: Takes as input the decryption key dk_Θ at level L and the delegation key mk_Ω as well as a set $\hat{\Theta}$. A delegator outputs a new decryption key $dk_{\Theta\cup\hat{\Theta}}$ if $\hat{\Theta} \subseteq \Omega$ and updates the delegation key $mk_{\hat{\Omega}}$ (where $\hat{\Omega} \subseteq (\Omega \setminus \hat{\Theta})$) for a delegatee at $L + 1$ or $\perp$.

Correctness. For every $k \in \mathbb{N}$, all $\Theta, \Theta', \hat{\Theta}, \Omega, \hat{\Omega}$, a message M, every n, t with $t \leq n$ and $\mathsf{ST} \in [n]$ with $|\mathsf{ST}| \geq t$, $(\mathsf{pp}, (sk_i, pk_i)_{i\in[n]}) \leftarrow \mathsf{Setup}(n, t, \lambda)$, $\mathsf{pk} \leftarrow \mathsf{AggKey}(pk_1, \ldots, pk_t)$, $(dk^i_\Theta, mk^i_\Omega) \leftarrow \mathsf{ShareKeyGen}(sk_i, \mathsf{ST}, \Theta, \Omega)$, $(dk_\Theta, mk_\Omega) \leftarrow \mathsf{KeyGenComb}(\{(dk^i_\Theta, mk^i_\Omega)\}_{i\in\mathsf{ST}})$ we have that if $\Theta = \Theta'$ then

$$\mathsf{Dec}(dk_\Theta, \mathsf{Enc}(\mathsf{pk}, \Theta', 1, M)) = M.$$

and if $\hat{\Theta} \subseteq \Omega$ and $\Theta \cup \hat{\Theta} = \Theta'$ then

$$\mathsf{Dec}(\mathsf{Delegate}(dk_\Theta, mk_\Omega, \hat{\Theta}, \hat{\Omega}), \mathsf{Enc}(\mathsf{pk}, \Theta', L + |\hat{\Theta}|, M)) = M.$$

Thereby, $\mathsf{Delegate}$ may be iteratively applied as long as the constraints are satisfied.

Encoding of Elements in TDSPE. We are now going to discuss how we can map an

arbitrary universe of elements (we call them attributes henceforth) to the TDSPE scheme by means of an example. Let $\mathcal{U}$ be a universe of attributes and assume a decryption key associated with a set of attributes $\Theta = \mathsf{A} = \{\alpha_1, \alpha_2\}$ from the universe. We have a delegation key for the set $\Omega = \mathsf{A}' = \{\alpha_3, \alpha_4, \alpha_5\}$. It is clear that for attributes sets A, A' with corresponding Θ and Ω, we have $\mathsf{A} \cup \mathsf{A}' := \Theta \cup \Omega = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$. Also assume that we get a set as $\hat{\Theta} = \{\alpha_3, \alpha_4\}$ for the delegation, then one can simply compute a new delegation key $dk_{\Theta \cup \Theta'} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. In this case we have an updated delegation key $dk_{\hat{\Omega}}$ for $\hat{\Omega} = \alpha_4$.

In order to see the expressiveness of the approach consider a Disjunctive Normal Form (DNF) formula $f = (C_1 \vee \ldots \vee C_t)$, where each C_j represents a conjunction over some subset of the attributes, i.e., $C_j = \bigwedge_i \alpha_{i,j}$ where $\alpha_{i,j} \in \mathcal{U}$.[2]

To encrypt a message M, a sender processes each of the t clauses independently. For the j'th clause C_j, the sender encrypts the message running Enc on input $\Theta^{C_j} = \Theta'$ as:

$$\Theta^{C_j} = \{\alpha_i | \alpha_i \in C_j\}.$$

The Enc algorithm generates a concatenation of ciphertexts related to each clause. For decryption, the user identifies a clause C_j that is satisfied by their attribute set A. Note that the attribute set A satisfies the formula f if there exists a clause C_j in the set $[t]$ such that $\mathsf{A} = C_j$. Thus, the user can decrypt the ciphertext corresponding to that clause if its key is associated with a set $\Theta = \Theta^{C_j}$ (or we can view it as $\mathsf{A} \subseteq f$, A satisfies the formula f).

Moreover, a user who possesses a secret key dk_Θ in the top level, associated with attribute set $\mathsf{A} := \{\alpha_1, \alpha_2\}$, can delegate any value (say $\mathsf{A}' = \{\alpha_3, \alpha_4\}$) of the second level key $dk_{\Theta \cup \hat{\Theta}}$ with the attribute set $\hat{\Theta} \subseteq \Omega$. For instance, if $\hat{\Theta} = \mathsf{A}'' = \{\alpha_3\}$ then the decryption key would be $\mathsf{A} \cup \mathsf{A}'' = \{\alpha_1, \ldots, \alpha_3\}$ (or any other combination regarding $\hat{\Theta}$) and the updated delegation key is $\hat{\Omega} = \alpha_4$. Clearly, one can delegate keys for less powerful predicates, i.e., if the set Θ' used in encryption is kept minimal, extending the set Θ in the decryption key via delegation soon removes the power to decrypt all ciphertexts that the original key can decrypt.

5.3.2 Security Definition

Subsequently, we define the security of our scheme. It is similar to that of SPE [KMMS17], i.e., in a selective setting, except that we allow additional corruption queries (for corrupted key issuers) as well as we consider delegation. In particular, we allow an adversary $\mathcal{A}$ to take control of up to $t-1$ authorities $C_{\mathcal{A}}$ and make them behave in an arbitrary manner, while these corruptions happen in a selective way at the start of the game (static corruption). For simplicity, let us assume a KeyGen algorithm that includes both ShareKeyGen and

[2]Note that when used for selective disclosure we have only one clause instead of many clauses. If users possess several keys, they can simply decide which keys and related attributes (selectively) they want to use for decryption.

$\underline{\mathsf{Exp}^{\mathsf{CPA}-b}_{\mathsf{TDSPE},\mathcal{A}}(k,n,t)}$:

- $(\Theta^*, C_{\mathcal{A}}) \leftarrow \mathcal{A}(k)$
- $(\mathsf{pp}, (sk_i, pk_i)_{i\in[n]}) \leftarrow \mathsf{Setup}(n,t,k)$
- $b' \leftarrow \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp}, (pk_i)_{i\in[n]}, (sk_j)_{j\in C_{\mathcal{A}}})$
- **return** $(b = b')$

$\underline{\mathcal{O}^{\mathsf{KeyGen}}(\Theta,\Omega,\mathsf{ST})}$:

- **if** $\Theta \cup \Omega = \Theta^* \vee |\Theta| \neq |\Omega|$ **return** $\perp$
- **else**:
  - $(dk^i_\Theta, mk^i_\Omega) \leftarrow \mathsf{ShareKeyGen}(sk_i, \Theta, \Omega)_{i\in\mathsf{ST}}$
  - $(dk_\Theta, mk_\Omega) \leftarrow \mathsf{KeyGenComb}(dk^i_\Theta, mk^i_\Omega)_{i\in\mathsf{ST}}$
  - **return** (dk_Θ, mk_Ω)

$\underline{\mathcal{O}^{\mathsf{Challenge}}(M_0,M_1,L)}$:

- **if** $|M_0| = |M_1|$:
  - $CT \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, M_b, L, \Theta^*)$
  - Return CT
- **else return** $\perp$.

Figure 5.2: Experiment $\mathsf{Exp}^{\mathsf{CPA}-b}_{\mathsf{TDSPE},\mathcal{A}}(1^\lambda)$

KeyGenComb algorithms. $\mathcal{A}$ may adaptively ask the challenger for a number of decryption keys queries including delegations as long as these keys and what can be delegated from them do not satisfy the challenge set Θ^*. Note that we do not provide an explicit delegation oracle, as the adversary will get access to delegation keys when it queries KeyGen. Formally, the security in TDSPE is defined using a game $\mathsf{Exp}^{\mathsf{CPA}-b}_{\mathsf{TDSPE},\mathcal{A}}(1^\lambda)$ in Fig. 5.2.

Definition 54 (IND-CPA)**.** *A* TDSPE *scheme is secure against chosen-plaintext attacks* (CPA) *under static corruption of authorities, if for any adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *such that*

$$|\Pr[\mathsf{Exp}^{\mathsf{CPA}-0}_{\mathsf{TDSPE},\mathcal{A}}(k,n,t) = 1] - \Pr[\mathsf{Exp}^{\mathsf{CPA}-1}_{\mathsf{TDSPE},\mathcal{A}}(k,n,t) = 1]| \leq \epsilon(\lambda),$$

where $b \in \{0,1\}$ *is a random coin and the experiment is defined in Figure 5.2.*

5.3.3 TDSPE Construction

Intuition. Our TDSPE construction is a thresholdized version of the second construction in [KMMS17] (see Section 2.4.2.1) but extended by a novel delegation feature. We also

realize an unbounded universe, i.e., number of attributes and in particular instead of putting a Y_i element for each element in the universe in the key, we implicitly define these values via a hash function (random oracle). In particular, for any possible attribute α_i, we compute it on the fly as $Y_i = H(\alpha_i)$. With this technique, we have also a compact public parameter (fixed size) independent of the number of attributes compared to SPE, which grows linearly with the set of attributes. We note that for simplicity $\mathsf{TDSPE.Setup}$ is ran by a single entity, but in a practical realization, the shares of private key and the corresponding public key are computed without a trusted party in a distributed protocol using standard distributed key generation techniques [GJKR07]. In order to dynamically add or remove authorities (which means an adversary might corrupt more than the threshold of issuers over a longer period of time), one can rely on dynamic proactive secret sharing [BELO15] instead of a SSS.

Now in the $\mathsf{ShareKeyGen}$ algorithm, we generate partial decryption keys and delegation keys for each authority in a specified subset. The keys are derived using random values r_i, hashed attribute values $Y_i = H(\alpha_i)$, and partial secret keys x_i with the Lagrange coefficients λ_i (this yields the full key by aggregating at least a threshold number of keys). The delegation keys are also generated the same way as decryption keys by hashing the delegation attribute values.

The $\mathsf{Delegate}$ algorithm allows the delegation of decryption capabilities. Given a decryption key $dk_\Theta = (h, K)$, a delegation key $mk_{\hat{\Theta}}$, and sets of attributes for further delegation $\hat{\Omega}$, it generates a new decryption key $dk_{\Theta\cup\hat{\Theta}} = (h, K')$ and delegation key $mk_{\hat{\Omega}}$ that extend the original capabilities to encompass the additional attributes. This can be achieved by combining the keys $mk_{\hat{\Theta}}$ and dk_Θ and aggregating their attributes, where the structure of keys allows us to combine them as $dk_{\Theta\cup\hat{\Theta}} = (h, K' = K \cdot mk_{\hat{\Theta}})$, where h is the same as before and K' represents the updated decryption key resulting from the combination of K (and its attributes) and the attributes provided by $mk_{\hat{\Theta}}$. We now describe TDSPE in detail as follows:

$\mathsf{TDSPE.Setup}(n, t, \lambda)$: Run $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \leftarrow \mathsf{BGGen}(1^\lambda)$, define the function $H : \{0,1\}^* \to \mathbb{G}_1$, and set public parameters as $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, H, P, \hat{P}, g_t)$. Pick a random $x \in \mathbb{Z}_p$, and generate secret shares as $(x_1, \ldots, x_n) \leftarrow \mathsf{SSS}(n, t, p, x)$, using a polynomial of degree $(t-1)$ with coefficients in $\mathbb{Z}_p$. We denote the i'th Lagrange coefficient by λ_i. For each authority i, set secret key share $sk_i := x_i$ and public key $pk_i = \left(\hat{P}^{x_i}\right)_{i\in[n]}$, and return $(sk_i, pk_i)_{i\in[n]}$.

$\mathsf{TDSPE.AggKey}(pk_1, \ldots, pk_t)$: Given any subset of at least t values of $pk_i = \hat{P}^{x_i}$, the algorithm computes the value $\mathsf{pk} = \prod_{i=1}^{t}(pk_i)^{\lambda_i}$ using standard Lagrange interpolation in the exponent. It outputs the public key (assumed to be input to all algorithms): $\mathsf{pk} = \left(X = \hat{P}^x\right)$ The algorithm returns the ciphertext: $CT = \left\{C_0, C_1, C_2, (C_j)_{j\in[q]}\right\}$.

$\mathsf{TDSPE.ShareKeyGen}(sk_i, \mathsf{ST}, \Theta, \Omega)$: Each authority i in any subset $\mathsf{ST} \subseteq [n]$ with $|\mathsf{ST}| \geq t$, takes as input a secret secret key sk_i and sets (Θ, Ω) of size q and ℓ which encode

values for the key and the delegation set respectively. It picks a random $r'_i \leftarrow \mathbb{Z}_p$ and outputs partial decryption keys and delegation keys:

$$dk^i_\Theta = \left(h_i = \hat{P}^{r'_i}, k_i = g_1^{x_i} \left(\prod_{j \in [q]} Y_j^{\lambda_i^{-1}} \right)^{r'_i} \right),$$

where $Y_j = H(\alpha_j)$, for all $\alpha_j \in \Theta$ and $i \in \mathsf{ST}$. For all $\beta_k \in \Omega = \{\beta_1, \ldots, \beta_\ell\}$, we have

$$mk^{i,k}_\Omega = \left(P^{x_i} \cdot Y_k^{r'_i} \right)_{k \in [\ell], i \in \mathsf{ST}},$$

where $Y_k = H(\beta_k)$, where β_k is the k'th element from Ω.

$\mathsf{TDSPE.KeyGenComb}(\{(dk^i_\Theta, mk^i_\Omega)\}_{i \in \mathsf{ST}})$: A user takes as input the partial functional decryption and delegation keys in the set ST, and eventually outputs the functional decryption key dk_Θ and the delegation key as:

$$dk_\Theta = \begin{pmatrix} h = \prod_{i \in \mathsf{ST}} h_i = g_2^{\sum r'_i}, K = \prod_{i \in \mathsf{ST}} k_i^{\lambda_i} = \\ g_1^{(\lambda_i \cdot x_i)} \cdot Y_j^{\sum_{i \in \mathsf{ST}} r'_i \left(\sum_{i=1}^{t} \lambda_i \cdot \lambda_i^{-1} \right)} \\ = \left(g_1^x \cdot \prod_{j \in [q]} Y_j^{\sum_{i \in \mathsf{ST}} r'_i} \right) \end{pmatrix}$$

and

$$mk^k_\Omega = \left(\prod_{i \in \mathsf{ST}, k \in [\ell]} (mk^{i,k}_\Omega)^{\lambda_i} \right) = \left(P^x \cdot Y_k^{\sum_{i \in \mathsf{ST}} r'_i} \right)_{k \in [\ell]}$$

$\mathsf{TDSPE.Enc}(\mathsf{pk}, \Theta', L, M)$: On input the public key pk and a set $\Theta' = (\alpha'_1, \ldots, \alpha'_I)$, a level L ($L = 1$ for the dk without delegation keys), and a message $M \in \mathbb{G}_t$, it chooses a random $r \in \mathbb{Z}_p$ and computes the ciphertext CT as follows:

$$\left(C_0 = M \cdot e(P, \prod_{L=1} X)^r, C_1 = g_2^r, (C_j = Y_j^r)_{j \in [I]} \right)$$

where $Y_j = H(\alpha'_j)$, for all $\alpha'_j \in \Theta'$.

$\mathsf{TDSPE.Dec}(dk_\Theta, CT)$: On input the decryption key $dk_\Theta = (h, K)$ for the set Θ and ciphertext CT, if $\Theta = \Theta'$ and the same level L, returns 1 if the following equation hols and $\perp$ otherwise:

$$\frac{C_0 \cdot e(\prod_{j \in q}(C_j), h)}{e(K, C_1)} = M,$$

where $j \in q$ means for an attribute in q, as the size of an attribute set Θ.

$\mathsf{TDSPE.Delegate}(dk_\Theta, mk_\Omega, \hat{\Theta}, \hat{\Omega})$: On input the decryption key $dk_\Theta = (K, h)$, for level L, the delegation key $mk_\Omega = (mk_\Omega^k)_{k\in[\ell]}$, and set $\hat{\Theta}$. If $\hat{\Theta} \not\subseteq \Omega$, returns $\perp$, else computes key components for $\hat{\Theta}$ from mk_Ω: Let $mk_{\hat{\Theta}} = \prod_{k\in\hat{\Theta}} mk_\Omega^k$ and $\sum_{i\in\mathsf{ST}} r'_i = r$, we can return a decryption key for $\Theta \cup \hat{\Theta}$ composed as:

$$dk_{\Theta\cup\hat{\Theta}} = (K \cdot mk_{\hat{\Theta}}) = \left(g_1^{\sum_{i=1}^{u} x} \cdot \prod_{j\in[|\Theta\cup\hat{\Theta}|]} (Y_j)^r, h \right)$$

which decrypts ciphertexts with $L + |\hat{\Theta}| = u$, note that $|\hat{\Theta}|$ can be 1 (the common case), if $\hat{\Theta}$ is only one element in the set Ω. A delegation key for $\hat{\Omega} \subseteq \Omega \setminus \hat{\Theta}$ (for further delegation) as:

$$mk_{\hat{\Omega}} = mk_\Omega^{k'} = \left(g_1^x \cdot \prod_{j\in\hat{\Omega}} Y_j^{\sum_{i\in\mathsf{ST}} r'_i} \right).$$

Remark 2. *We remark that one could use certain CP-ABE schemes instead of our TDSPE scheme which would yield (depending on the CPA-ABE) more expressive policies. The key advantages of using TDSPE is its inherent support for thresholdizing and delegation, which provides significant benefits in terms of flexibility and functionality. Nevertheless, we want to mention that a CP-ABE scheme that efficiently supports key delegation and thresholdizing the key generation could serve as a potential candidate for our construction. While the BSW CP-ABE [BSW07] scheme supports efficient delegation, the need to compute g^β and $g^{1/\beta}$ makes the distributed key generation very costly. We consider it as an interesting future work to identify or even modify CP-ABE schemes, in a way that both additional requirements can be met efficiently.*

For our construction we can show the following:

Theorem 5.3.1. *Assuming that the Decisional Bilinear Diffie-Hellman assumption holds, the above* TDSPE *construction is CPA secure in the random oracle model according to Definition 54.*

Proof. Correctness follows from inspection. The main difference compared to [KMMS17] is that we now do not fix the Y_i elements in the setup, but derive them via a random oracle (RO). For the proof strategy this makes up to collisions (which happens with negligible probability when using the RO) only a conceptual difference, i.e., for elements inside the challenge set we program the random oracle just to random values and for the elements outside the challenge set (which is in the selective setting known from the beginning) the RO is programmed exactly as the Y_i elements are generated in [KMMS17]. The only addition is to handle the delegation. For the delegation feature, we can easily compute consistent mk elements when using the proof strategy of [KMMS17] as follows: The case

where $\Omega = \emptyset$, i.e., no delegation keys are requested, skip to compute the mk key. Otherwise, we can observe that if $\Theta \cup \Omega \not\subseteq \Theta^*$, directly follows from [KMMS17] i.e, compute mk_Ω similar the decryption key for Θ and then since $\Theta \neq \emptyset$, i.e., at least one index must be non-zero, this in particular means that $\Theta \not\subseteq \Theta^*$. This implies that this also needs to be set in Θ^* in order to be able to decrypt and this does not represent a valid key query. Consequently, whenever Θ not already represents a subset of Θ^*, then Ω will not help to achieve this either. Therefore, we can exactly follow the simulation of keys in [KMMS17] which implicitly sets the randomness to $r' = r - \frac{b}{\sum_{j \in I} w_j}$ (where the w is from the challenge Θ^*). More precisely, we can perfectly simulate the respective delegation key elements by implicitly computing $Y^{r'}_{k \in I'}$ as $Y^{r'}_k = P^r \cdot ((B_1)^{-1})^{\frac{1}{w}}$. For the threshold issuing, we can exactly follow the strategy by Boneh et al [BBH06] in their threshold Identity based encryption. We describe the reduction as follows:

Basis of DBDH reduction. If there exists an adversary $\mathcal{A}$ that could break the security property of our proposed scheme (Def. IND-Security 54), then we could built a simulation $\mathcal{B}$ to break the DBDH assumption. Our reduction follows an approach described in [KMMS17], and behaves as follows: The simulation $\mathcal{B}$ takes as input a decision DBDH tuple $(P, A_1 = P^a, B_1 = P^b, C_1 = P^c, \hat{P}, A_2 = \hat{P}^a, B_2 = \hat{P}^b, C_2 = \hat{P}^c, Z)$, where Z is either $e(P, \hat{P})^{abc}$ or a random element $\mathbb{G}_T$. $\mathcal{B}$ outputs 1 if $Z = e(P, \hat{P})^{abc}$ and 0 otherwise. Without losing generality, the simulation receives the targeted predicates sets Θ^* and a list of $t-1$ corrupted authorities $C_\mathcal{A}$ (and their secret keys) from the adversary. The reduction $\mathcal{B}$ gets as input a DBDH tuple as above and uses $\mathcal{A}$ to interact with a TDSPE challenger:

- Initialization. The selective game begins with $\mathcal{A}$ first outputting predicates set $\Theta^* = \{\alpha^*_1, \ldots, \alpha^*_q\}$, that it intends to attack. Moreover, $\mathcal{A}$ chooses a set $C_\mathcal{A}$ of $t-1$ decryption servers that it wants to corrupt. Let $C_\mathcal{A} = \{s_1, \ldots, s_{t-1}\} \subset \{1, \ldots, n\}$.

- Setup. To generate the system parameters, the algorithm $\mathcal{B}$ proceeds as follows:
  1. First, $\mathcal{B}$ gives $\mathcal{A}$ the public parameters as $\mathsf{pp} = (P, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$.
  2. Next, $\mathcal{B}$ generates the secret key shares for the $t-1$ corrupt servers in $C_\mathcal{A}$. To do so, $\mathcal{B}$ first picks $t-1$ random integers $x_1, \ldots, x_{t-1} \in \mathbb{Z}_p$. Let $f \in \mathbb{Z}_p[X]$ be the degree $t-1$ polynomial implicitly defined to satisfy $f(0) = x$ and $f(s_i) = x_i$ for $i = 1, \ldots, t-1$; note that $\mathcal{B}$ does not know f since it does not know x. $\mathcal{B}$ gives $\mathcal{A}$ the $t-1$ secret key shares $\mathsf{sk}_{s_i} = x_i$. These keys are consistent with this polynomial f since $\mathsf{sk}_{s_i} = f(s_i)$ for $i = 1, \ldots, t-1$.
  3. Finally, $\mathcal{B}$ constructs the public encryption key, which is a n-vector $(pk_1, \ldots, pk_n)$ such that $pk_i = \hat{P}^{f(i)}$ for the polynomial f defined above, as follows:
     - For $i \in C_\mathcal{A}$, computing pk_i is easy since $f(i)$ is equal to one of the $x_1, \ldots, x_{t-1}$, which are known to $\mathcal{B}$. Thus, $pk_{s_1}, \ldots, pk_{s_k} \in \mathbb{G}_2$ are easy for $\mathcal{B}$ to compute.

– For $i \notin C_{\mathcal{A}}$, algorithm $\mathcal{B}$ needs to compute the Lagrange coefficients $\lambda_0, \lambda_1, \ldots, \lambda_{t-1} \in \mathbb{Z}_p$ such that $f(i) = \lambda_0 f(0) + \sum_{j=1}^{t-1} \lambda_j f(s_j)$; these Lagrange coefficients are easily calculated since they do not depend on f. $\mathcal{B}$ then sets $pk_i = g_2^{\lambda_0} pk_{s_1}^{\lambda_1}, \ldots pk_{s_{t-1}}^{\lambda_{t-1}}$, which entails that $pk_i = g_2^{f(i)}$ as required.

Once it has computed all the pk_i 's, $\mathcal{B}$ gives to $\mathcal{A}$ the public encryption key $\mathsf{pk} = (pk_1, \ldots, pk_n)$.

- Query. $\mathcal{A}$ may adaptively make a polynomial number of key generation queries to $\mathcal{B}$.

  Random Oracle Calls:

  – It proceeds by uniformly sampling for all $\alpha_i \in \Theta^*$ an elements $y_i \in \mathbb{Z}_p^*$ and setting $Y_i = P^{y_i}$. It stores all values y_i.

  – For $j = 1, 2, \ldots, \varphi$, $\mathcal{B}$ responds to hash query number j for a fresh input $\Theta^j \not\subseteq \Theta^*$ by choosing $(y_i, w) \in (\mathbb{Z}_p^*)^{q+1}$ for all $\alpha_i \in \Theta^j$ (or $\beta_i \in \Omega^j$, when the delegation key is required) and sets $Y_i = A^w \cdot P^{y_i} = g^{a \cdot w + y_i}$, where $q = |\Theta^j|$.

  Decryption Key Query: $\mathcal{A}$ may adaptively make a polynomial number of key generation queries for some sets $\Theta^j = \{\alpha_i\}_{i \in [q]}, \Omega^j = \{\beta_i\}_{i \in [q]}$ under the constraint that for all $j \in \{1, \ldots, \varphi - 1\}$, it holds that $\Theta^j \not\subseteq \Theta^*$ and $|\Omega^j| = |\Theta^j| = q$. The $\mathcal{B}$ calls RO, picks r and responds to each query j and partial decryption keys as follows:

$$dk_{\Theta^j} = \left(B_1^{\frac{-\sum_{i \in \Theta^j} y_i}{qw}} \cdot \left(\prod_{i \in \Theta^j} Y_i' \right)^r, B_2^{\frac{-1}{qw}} \cdot g_2^r \right).$$

We observe that:

$$dk_{\Theta^j} = \left(g_1^{b \cdot \frac{-\sum_{i \in \Theta^j} y_i}{q \cdot w} + r\left(\sum_{i \in \Theta^j} a \cdot w + y_i\right)}, g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab + b \cdot \left(\frac{-\sum_{i \in \Theta^j} y_i}{qw} - a \right) + r\left(\sum_{i \in \Theta^j} a \cdot w + y_i\right)}, g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab - \frac{b}{qw} \cdot \left(\sum_{i \in \Theta^j} y_i + \sum_{i \in \Theta^j} aw\right) + r\left(\sum_{i \in \Theta^j} (a \cdot w + y_i)\right)}, g_2^{r - \frac{b}{qw}} \right)$$

$$= \left(g_1^{ab + (r - \frac{b}{qw})\left(\sum_{i \in \Theta^j} (a \cdot w + y_i)\right)}, g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab} \cdot \left(\prod_{i \in \Theta^j} Y_i \right)^{\left(r - \frac{b}{qw}\right)}, g_2^{r - \frac{b}{qw}} \right)$$

To see that dk_{Θ^j} is a valid secret key for Θ^j, set $r' := r - b/(\sum w_i) \in \mathbb{Z}_p$ and $x = ab$. This provides a valid and properly distributed key, as r is uniformly distributed over $\mathbb{Z}_p$ as $g^x \cdot \prod_{j \in \Theta^j} (Y_j)^{r'}$. Moreover, if $\Omega \neq \emptyset$, we can simulate the delegation key mk_{Ω^j} in the same way of dk_{Θ^j} as

$$mk_{\Omega^j} = \left(B_1^{\frac{-\sum_{i \in \Theta^j} y_i}{qw}} \cdot \left(\prod_{i \in \Omega^j} Y_i \right)^r , g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{\frac{-\sum_{i \in \Theta^j} b \cdot y_i}{qw} \cdot r\left(\sum_{i \in \Omega^j} aw + y_i\right)} , g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab + b \cdot \left(\frac{-\sum_{i \in \Omega^j} y_i}{qw} - a \right) + r\left(\sum_{i \in \Theta^j} a \cdot w + y_i\right)} , g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab - \frac{b}{qw} \cdot \left(\sum_{i \in \Omega^j} y_i - \sum_{i \in \Omega^j} aw\right) + r\left(\sum_{i \in \Omega^j} a \cdot w + y_i\right)} , g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab + (r - \frac{b}{qw})\left(\sum_{i \in \Omega^j} (a \cdot w + y_i)\right)} , g_2^{r - \frac{b}{qw}} \right) =$$

$$\left(g_1^{ab} \cdot \left(\prod_{i \in \Omega^j} Y_i \right)^{\left(r - \frac{b}{qw}\right)} , g_2^{r - \frac{b}{qw}} \right)$$

which one can produce a key $dk_{\Theta^j \cup \Omega^j}$ for $\Theta^j \cup \Omega^j \not\subseteq \Theta^*$, this in particular means that delegation key does not help $\mathcal{A}$ to win the game.

$$dk_{(\Theta \cup \Omega)^j} =$$
$$\left(g_1^{ab} \cdot \left(\prod_{i \in \Theta^j} Y_i \right)^{\left(r - \frac{b}{qw}\right)} \right) \cdot \left(g_1^{ab} \cdot \left(\prod_{i \in \Omega^j} Y_i \right)^{\left(r - \frac{b}{qw}\right)} \right)$$
$$= \left(g_1^{ab} \cdot g_1^{ab} \cdot \left(\prod_{i \in (\Theta \cup \Omega)^j} Y_i \right)^{\left(r - \frac{b}{qw}\right)} \right)$$

We mention that the procedure encounters a failure whenever $w = 0$ (due to division by zero in the exponent). This situation occurs only if the queried set Θ^j is a subset of the challenge set Θ^*, but with a negligible probability.

- Challenge: The adversary submits two messages $(M_0, M_1) \in \mathbb{G}_T$. $\mathcal{B}$ randomly selects a value $b \in \{0,1\}$ and provides the attacker with the challenge ciphertext $CT^* = (C_0 = M_b \cdot Z, C_1, \forall j \in \Theta^* : C_1^{y_i})$ We should mention that if $Z = e(P,\hat{P})^{abc}$ then CT^* is a valid ciphertext as

$$\begin{Bmatrix} C_1 = M_b \cdot Z = M_b \cdot e(P,\hat{P})^{abc} = M_b \cdot e(A_1, B_2)^c, \\ C_2 = \hat{P}^c, \forall j \in \Theta^* : C_1^{y_j} = (g_1^{y_j})^c = Y_j^c \end{Bmatrix}$$

  In contrast, the message M_b remains hidden from the adversary's view in an information-theoretic sense as Z is uniformly distributed in $\mathbb{G}_T$,

Clearly, when the ciphertext tuple contains $Z = (P,\hat{P})^{abc}$, $\mathcal{A}$'s view perfectly resembles the expected inputs in the standard security experiment. This implies that $\mathcal{A}$ has an advantage greater than some non-negligible $\epsilon(\lambda)$, as assumed. Alternatively, if the input tuple includes a uniformly distributed element Z in $\mathbb{G}_T$, $\mathcal{A}$ gains no information about the secret bit b. Consequently, in this scenario, $\mathcal{A}$ cannot outperform random guessing, leading to a contradiction with the DBDH assumption, and thus concluding our proof.

5.4 Threshold Delegatable Anonymous Credentials

We start by introducing a formal model for a threshold delegatable anonymous credential (TDAC) system. We note that we model the algorithm TDAC.IssueCred as a non-interactive algorithm as there is no input from the user required and assume that users obtain their partial credentials via a private authenticated channel (which is outside the model).

5.4.1 Formal Definition

Definition 55. *A Threshold Delegatable Anonymous Credential* TDAC *system consists of a set of authorities, users/provers, and verifier(s) and also the following algorithms:*

TDAC.Setup$(\lambda, n, t) \to (\mathsf{pp}, (sk_i, pk_i)_{i\in[n]})$: Setup takes as input the security parameter λ, two additional parameters (n,t), and generates the public parameters pp which include the combined public key pk (assumed to be input to all algorithms), a vector of n secret key shares $(sk_1, \ldots, sk_n)$ with threshold t, and corresponding public keys $(pk_1, \ldots, pk_n)$ for each authority. It is run by authorities.

TDAC.IssueCred$(sk_i, \mathsf{pk}, \mathsf{A}, \mathsf{A}') \to (\sigma^i, mk^i)$: This algorithm is run by authority i in a subset ST of $[n]$, where $|\mathsf{ST}| \geq t$, by which the user obtains partial credentials $(\sigma^i)_{i\in\mathsf{ST}}$ for attributes $\mathsf{A} = \{\alpha_1 \ldots, \alpha_q\}$, and also partial delegation keys $(mk^i)_{i\in\mathsf{ST}}$ for attributes A'.

TDAC.AggCred$((\sigma^i, mk^i)_{i\in\mathsf{ST}}) \to (\sigma, mk)$: The user runs AggCred to aggregate any subset of $|\mathsf{ST}| \geq t$ partial credentials σ^i and delegation keys into a single consolidated

$\underline{\mathsf{ExpAno}^{b}_{\mathsf{TDAC},\mathcal{A}}(k,n,t)}$:

- $f^* \leftarrow \mathcal{A}(k)$
- $(\mathsf{pp}, (sk_i, pk_i)_{i\in[n]}) \leftarrow \mathsf{Setup}(k,n,t)$
- $b' \leftarrow \mathcal{A}^{\langle \mathcal{O}_b^{\mathsf{Anon}}, \mathcal{O}\rangle}(\mathsf{pp}, (pk_i)_{i\in[n]}))$
- **return** $(b = b')$

$\underline{\mathcal{O}_b^{\mathsf{Anon}}(i_0, i_1, f^*)}$:

- If i_0 or $i_1 > |Q|$ the oracle returns $\perp$.
- Else, it parses $Q[i_0]$ as $(\mathsf{Id}_0, \mathsf{A}_0, \mathsf{A}'_0, mk_0, \sigma_0)$ and $Q[i_1]$ as $(\mathsf{Id}_1, \mathsf{A}_1, \mathsf{A}'_1, mk_1, \sigma_1)$.
- If $f^*(\mathsf{A}_0) \neq f^*(\mathsf{A}_1)$, **return** $\perp$.
- Otherwise, **return**: $\mathsf{ProveCred}(\sigma_b, \mathsf{A}_b, f^*) \leftrightarrow \mathcal{A}$

$\underline{\mathsf{ExpUnf}_{\mathsf{TDAC},\mathcal{A}}(k,n,t)}$:

- $(f^*, C_{\mathcal{A}}) \leftarrow \mathcal{A}(k)$
- $(\mathsf{pp}, (sk_i, pk_i)_{i\in[n]}) \leftarrow \mathsf{Setup}(k,n,t)$
- $\top \leftarrow \mathcal{A}^{\langle \mathcal{O}\rangle}(\mathsf{pp}, (pk_i)_{i\in[n]}), (sk_j)_{j\in C_{\mathcal{A}}})$
- If $\forall(\mathsf{Id}, \mathsf{A}, .) \in Q$ s.t. $\mathsf{Id} \in C_{\mathcal{U}}$: $f^*(\mathsf{A}) = 0$, **return**

$$\mathcal{A} \leftrightarrow \mathsf{VerifyCred}(\mathsf{pk}, f^*)$$

Figure 5.3: Experiments $\mathsf{ExpAno}_{\mathsf{TDAC},\mathcal{A}}(k,n,t)$ and $\mathsf{ExpUnf}_{\mathsf{TDAC},\mathcal{A}}(k,n,t)$.

credential σ and delegation key, and obtains a full credential on the attribute set A and also gets a full delegation key mk.

$\mathsf{TDAC.ProveCred}(\sigma, \mathsf{A}, f) \leftrightarrow \mathsf{TDAC.VerifyCred}(\mathsf{pk}, f) \rightarrow (1/0)$: This interactive protocol is run by a user and a verifier to prove possession of a credential certifying that attributes satisfy the policy (predicate) f under the public key pk of issuers. $\mathsf{ProveCred}$ takes as input the credential σ, attributes A and the policy f, while $\mathsf{VerifyCred}$ takes as input pk and the policy f. After execution, the verifier outputs 1 if the credential satisfies the policy or 0 otherwise.

$\mathsf{TDAC.DelegIssue}(\sigma, mk, \mathsf{A}, \mathsf{A}', \mathsf{A}'') \rightarrow \sigma'$. The delegation algorithm gets as input the credential σ for attributes A, the delegation key mk for A' and a new attribute vector $\mathsf{A}'' \subseteq \mathsf{A}'$. A delegator outputs a derived credential σ' for attributes $\mathsf{A} \cup \mathsf{A}''$ and an updated delegation key mk regarding A'' to a delegatee. It is run by the delegatee and delegator.

5.4.2 Security Definition

We define our security model for TDAC based on the game-based framework of Deuber et al. [DMM+18]. The adversary has access to oracles below that describe the possible ways to interact with the system. We use $\langle\mathcal{O}\rangle$ to denote the collection of all oracles defined in the games. Moreover, we assume four global lists that are shared among the oracles as: $\mathcal{HU}$ a list of honest users, $\mathcal{CU}$ a list of corrupted users, Q a list of user-credential pairs, and $C_{\mathcal{A}}$ a list of corrupted authorities. Note that we do not provide an explicit delegation oracle, as the adversary will get access to all delegation keys when corrupting users. Oracles are defined as follows:

- $\mathcal{O}^{\mathsf{User}}(\mathsf{Id})$: This oracle takes a user identity Id as input. If Id is found in the list of honest users $\mathcal{HU}$ or the list of corrupted users $\mathcal{CU}$, it returns $\perp$, indicating an error. Otherwise, it creates a new entry for Id in the honest users list $\mathcal{HU}$.
- $\mathcal{O}^{\mathsf{Corrupt}}(\mathsf{Id})$: This oracle takes a user identity Id as input. If Id is not found in the honest users list $\mathcal{HU}$, it returns $\perp$. Otherwise, it moves the entry associated with Id from the honest users list $\mathcal{HU}$ to the corrupted users list $\mathcal{CU}$. It also returns all the items in the form of $(\mathsf{Id}, \mathsf{A}, \mathsf{A}', mk, \sigma) \in Q$, as well as σ.
- $\mathcal{O}^{\mathsf{Issue}}(\mathsf{Id}, \mathsf{A}, \mathsf{A}')$: This oracle takes a user identity Id, an attributes set A, and a delegatable attributes set A' as input. If Id is not found in either the honest users list $\mathcal{HU}$ or the corrupted users list $\mathcal{CU}$, it returns $\perp$, indicating an error. Otherwise, it generates a signature σ using the $\mathsf{IssueCred}$ function with the user's secret key sk_i, A, and A'. It then adds the entry $(\mathsf{Id}, \mathsf{A}, \mathsf{A}', mk, \sigma)$ to the list Q.
- $\mathcal{O}^{\mathsf{ProveCred}}((i, f))$: This oracle takes an index i and a policy set Θ as input. If i is greater than the size of the list Q, it returns $\perp$, indicating an error. Otherwise, it retrieves the entry at index i from Q and parses it as $(\mathsf{Id}, \mathsf{A}, \mathsf{A}', mk, \sigma)$. It then runs the $\mathsf{ProveCred}(\sigma, \mathsf{A})$ function with the adversary (acting as a verifier in the verification protocol).

Anonymity. Anonymity requires that a malicious verifier cannot distinguish between any two users (cf. Figure 5.3). The adversary has adaptive access to an oracle that on input two distinct user indexes i_0 and i_1, acts as one of the two credential owners (depending on bit b) in the verification algorithm on some policy predicate f^* chosen by the adversary in a selective way. Note that $f^*(\mathsf{A}) = 1$ if attributes in A satisfy the policy and $f^*(\mathsf{A}) = 0$ otherwise. To make the game non-trivial, we impose restrictions that the policy is either satisfied or not satisfied by both credentials. Note that this implies unlinkability.

Definition 56 (Anonymity)**.** *A* TDAC *is anonymous, if for all* $k \in \mathbb{N}$*,* n, t *with* $t \leq n$*, any* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *such that*

$$\begin{aligned} |\Pr[\mathsf{ExpAno}^0_{\mathsf{TDAC},\mathcal{A}}(k, n, t) = 1] - \\ \Pr[\mathsf{ExpAno}^1_{\mathsf{TDAC},\mathcal{A}}(k, n, t) = 1]| \leq \epsilon(\lambda), \end{aligned}$$

where $b \in \{0, 1\}$ *is a flipped coin and the experiment is defined in Fig 5.3.*

Unforgeability. Unforgeability demands that no adversary can persuade a verifier into accepting a credential for a policy that the adversary does not genuinely satisfy. In simple terms, an adversary succeeds in the unforgeability experiment (refer to Figure 5.3) if they manage to convince an honest verifier that they satisfy a particular policy, even though they lack the appropriate credential. The adversary has the freedom to select any policy that no corrupted user can satisfy and can corrupt any set of authorities below the threshold.

Definition 57 (Unforgeability)**.** *A* TDAC *scheme is unforgeable if, for all* $k \in \mathbb{N}$, n, t *with* $t \leq n$, *for any* PPT *adversary* $\mathcal{A}$, *there exists a negligible function* $\epsilon(\lambda)$ *such that* $\Pr[\mathsf{ExpUnf}_{\mathsf{TDAC},\mathcal{A}}(k,n,t) = 1] \leq \epsilon(\lambda)$, *where* $\mathsf{ExpUnf}_{\mathsf{TDAC},\mathcal{A}}(k,n,t)$, *where the experiment is defined in Figure 5.3.*

Remark 3. *We note that in contrast to Deuber et al. [DMM+18], who aim to provide a generic framework for functional credentials, our goal is very efficient instantiations. Consequently, due to the lack of efficient adaptively secure underlying primitives we opted to provide our definitions in a selective sense. Nevertheless, our definitions can easily be adapted to the adaptive setting if efficient primitives are available.*

5.4.3 Construction

Our construction TDAC is based on TDSPE and an equivocable and extractable commitment scheme EQTDC, which closely follows the interactive protocol presented by Deuber et al. [DMM+18]. The idea behind the protocol is that the attribute policy (predicate) is associated with the ciphertext. This is an important property to prevent using zero knowledge proofs that might be inefficient and complex. Consequently, the user proves the possession of a credential by decrypting a ciphertext that encodes the policy set, where the decryption key represents the credential. To ensure that the ciphertext was honestly computed by the verifier, the verifier initially commits to randomness r, the user sends randomness r' and the verifier determines the challenge S and encryption randomness t as $S||t = r \oplus r'$. After decrypting the challenge ciphertext CT, the user first commits to the result M of the decryption (which in case the verifier behaves honest is equal to S). It then lets the verifier disclose the randomness r, which was used to compute the initial ciphertext. Now having r and r' the user can compute $S||t$ and by re-encryption can check if the ciphertext CT was indeed honestly generated. If so, the user sends the opening information to the challenge $M = S$ and otherwise if $M \neq S$ it aborts. Finally, the verifier accepts if the opening yields the plaintext $M = S$ corresponding to the original ciphertext. The full protocol is depicted in Figure 5.4. Note that we use the respective syntax of TDSPE and thus the encoding of attribute sets A, A', A'' to the respective sets $\Theta, \Omega, \hat{\Theta}$ is done in the obvious way as already discussed in Section 5.3.1.

By setting $\mathsf{A}' = \mathsf{A}$ in the TDAC.IssueCred, we obtain the standard functionality of Delegatable AC. Therefore, maintaining A' equal to A ensures the same level of functionality as previous DAC definitions. However, the advantage in our approach lies in making A' smaller, which enables fine-grained controlled delegation for enhanced flexibility and granularity in the credential delegation.

Also, note that for a DNF f, we use the abbreviation $\mathsf{TDSPE.Enc}(PK, f, M)$ to actually mean running the encryption algorithm of the respective sets as also discussed in Section 5.3.1 and also f includes the level L. Finally, we make the simplification and write $\mathsf{TDSPE.Dec}(CT, \sigma)$ whereas we mean that the prover determines the set Θ that satisfies f with attribute A and then decrypts the respective ciphertext. We can prove the following:

- $\mathsf{TDAC.Setup}(n, t, \lambda)$: Run $(\mathsf{pp}, (sk_i, pk_i)_{i\in[n]}) \leftarrow \mathsf{TDSPE.Setup}(k, n, t)$. Compute $\mathsf{pk} \leftarrow \mathsf{TDSPE.AggKey}(pk_1, \ldots, pk_t)$, $\mathsf{crs} \leftarrow \mathsf{EQTDC.Setup}(1^\lambda)$ and output $\big((\mathsf{pp}, \mathsf{pk}, \mathsf{crs}), (sk_i, pk_i)_{i\in[n]}\big)$.
- $\mathsf{TDAC.IssueCred}(sk_i, \mathsf{pk}, \mathsf{A}, \mathsf{A}')$: Each authority runs $(dk^i_\Theta, mk^i_\Omega) \leftarrow \mathsf{TDSPE.ShareKeyGen}(sk_i, \mathsf{ST}, \Theta, \Omega)$. When a user obtains them it sets a partial credential as $\sigma^i = dk^i_\Theta$ for attributes A and also a partial delegation key as mk^i_Ω for attributes A'.
- $\mathsf{TDAC.AggCred}(\{\sigma^i, mk^i\}_{i\in\mathsf{ST}})$: The user runs $(dk_\Theta, mk_\Omega) \leftarrow \mathsf{TDSPE.KeyGenComb}(\{dk^i_\Theta, mk^i_\Omega\}_{i\in\mathsf{ST}})$ to aggregate a subset of t partial credentials σ^i into a single credential and sets $\sigma = dk_\Theta$ as a full credential and $mk = mk_\Omega$ as a full delegation key.
- $\mathsf{TDAC.ProveCred}(\sigma, \mathsf{A}, f) \leftrightarrow \mathsf{TDAC.VerifyCred}(\mathsf{pk}, f)$: The user ($\mathsf{U}$) and the verifier ($\mathsf{V}$) interact as follows:
  - V chooses a random r and computes $(\mathsf{com}_0, \mathsf{decom}_0) \leftarrow \mathsf{EQTDC.com}(\mathsf{crs}, r)$ and sends com_0 to U.
  - U chooses a random r' and sends it to V.
  - V computes $S\|t = r \oplus r'$, $CT \leftarrow \mathsf{TDSPE.Enc}(\mathsf{pk}, f, L, S; t)$ and sends CT to U.
  - U computes $M \leftarrow \mathsf{TDSPE.Dec}(CT, \sigma)$ and $(\mathsf{com}_1, \mathsf{decom}_1) \leftarrow \mathsf{EQTDC.com}(\mathsf{crs}, M)$, sends com_1 to V.
  - V sends the opening information decom_0 of com_0 to U.
  - U obtains $S'\|t' = \mathsf{EQTDC.VerCom}(\mathsf{crs}, \mathsf{com}_0, \mathsf{decom}_0) \oplus r'$ and checks if $CT = \mathsf{TDSPE.Enc}(\mathsf{pk}, f, L, S'; t')$. If this check holds it sends the opening information decom_1 to V.
  - V accepts if $S = \mathsf{EQTDC.VerCom}(\mathsf{crs}, \mathsf{com}_1, \mathsf{decom}_1)$ and returns $\perp$ otherwise.
- $\mathsf{TDAC.DelegIssue}(\sigma, mk, \mathsf{A}, \mathsf{A}', \mathsf{A}'')$: If a user (delegator) on level-L wants to issue a credential to a user (delgatee) on level-$L+1$, the delegator performs $(dk_{\Theta\cup\hat{\Theta}}, mk_{\hat{\Omega}}) \leftarrow \mathsf{Delegate}(dk_\Theta, mk_\Omega, \hat{\Theta}, \hat{\Omega})$ on inputs the delegation key with attributes A' and a new attribute vector $\mathsf{A}'' \subseteq \mathsf{A}'$. It outputs a derived credential $\sigma' = dk_{\Theta\cup\hat{\Theta}} = dk_{\mathsf{A}\cup\mathsf{A}''}$ and a updated delegation key $mk_{\hat{\Omega}}$ to a delegatee.

Figure 5.4: Our threshold delegatable anonymous credentials scheme.

Theorem 5.4.1. *If* TDSPE *and* EQTDC *are secure, then* TDAC *protocol in Figure 5.4 is unforgeable and anonymous.*

Our proof is inspired by the paper of Deuber et al. [DMM+18]. We note that Deuber et al. [DMM+18] use a very different delegation mechanism, which is not present in our construction, but can easily be omitted from their construction (one then does no longer require a EUF-CMA secure signature scheme). Moreover, in contrast to Deuber et al. we have two additional features: we provide a threshold issuing as well as a credential delegation in a way that a delegation key mk allows to compute a delegated credential for an extended attribute vector. The threshold issuing however, is encapsulated by the underlying TDSPE scheme and thus does not require any change to the proof (i.e., $C_\mathcal{A}$ the

list of corrupted authorities just calls the corrupt authorities in TDSPE). The same holds for the delegation, which exactly matches to the KeyGen query of TDSPE.

Lemma 5.4.2 (Unforgeability)**.** *Let* TDSPE *be a secure threshold delegatable subset predicate encryption scheme and* EQTDC *be an equivocable and extractable commitment scheme, then* TDAC *is unforgeable.*

Proof. We employ a sequence of games to modify the unforgeability experiment and demonstrate that the probability of success for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ is negligibly.

Game_0. Original game in Definition 57.

Game_1. The $\mathsf{Setup}(1^\lambda)$ is changed by having crs from $\xi(1^\lambda)$ (along with a which is stored locally).

- $(\mathsf{pp}, (sk_i, pk_i)_{i\in[n]}) \leftarrow \mathsf{TDAC.Setup}(n, t, \lambda)$
- $\boxed{\mathsf{crs}, a} \leftarrow \mathsf{EQTDC}.\xi(1^\lambda)$

Game_2. In this game, we modify the algorithm $\mathsf{ProveCred}(\sigma, \mathsf{A}, f)$ as follows:

- Receives com_0 and chooses $r' \leftarrow \mathbb{Z}_p^*$
- $\boxed{m' \leftarrow \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_0)}$
- $m \leftarrow \mathsf{TDSPE.Dec}(CT, \sigma)$ and $\mathsf{com}_1 \leftarrow \mathsf{EQTDC.Commit}(\mathsf{crs}, m)$.
- Computes $\boxed{s'\|t' = m' \oplus r'}$. It sends decom_1 to Verifier.

Game_3. In this transition, we change the algorithm $\mathsf{ProveCred}(\sigma, \mathsf{A}, f)$ as follows:

- Receives com_0 and chooses $r' \leftarrow \mathbb{Z}_p^*$
- $m' \leftarrow \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_0)$
- $m \leftarrow \mathsf{TDSPE.Dec}(CT, \sigma)$ and $\boxed{\mathsf{com}_1 \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a)}$.
- Computes $s'\|t' = m' \oplus r'$ and $\boxed{\mathsf{decom}_1 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_1, m)}$.

Game_4. In the next step, we modify the algorithm $\mathsf{ProveCred}(\sigma, \mathsf{A}, f)$ as follows: The decryption is no need to evaluate and this algo computes $\mathsf{decom}_1 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_1, s')$ if the attributes associated to a credential σ satisfy the policy f, otherwise return $\perp$.

- Receives com_0 and chooses $r' \leftarrow \mathbb{Z}_p^*$
- $m' \leftarrow \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_0)$
- $\mathsf{com}_1 \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a)$
- Computes $s'\|t' = m' \oplus r'$ and $\boxed{\mathsf{decom}_1 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_1, s')}$

Game_5. The algorithm $\mathsf{VerifyCred}(\mathsf{pk}, f)$ is changed as follows:

- Choose $r \leftarrow \mathbb{Z}_p^*$, $\boxed{\mathsf{com}_0 \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a)}$.
- Receives r', $s \| t = r \oplus r'$, and $CT \leftarrow \mathsf{TDSPE.Enc}(\mathsf{pk}, f, L, s; t)$.
- Receives com_1 and $\boxed{\mathsf{decom}_0 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_0, r)}$.
- Receives decom_1. Accept if $s = \mathsf{EQTDC.VerCom}(\mathsf{crs}, \mathsf{com}_1, \mathsf{decom}_1)$ and return $\perp$ otherwise.

Game_6. The algorithm $\mathsf{VerifyCred}(\mathsf{pk}, f)$ is further changed as follows:

- Choose $r \leftarrow \mathbb{Z}_p^*$, $\mathsf{com}_0 \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a)$.
- Receives r', $s \| t = r \oplus r'$, and $CT \leftarrow \mathsf{TDSPE.Enc}(\mathsf{pk}, f, L, s; t)$.
- Receives com_1, $\mathsf{decom}_0 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_0, r)$, $\boxed{m' \leftarrow \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_1)}$.
- Receives decom_1. If $\boxed{s = m'}$, return 1 and $\perp$ otherwise.

Game Transitions The game transition section is mostly taken verbatim from [DMM+18], but adapted for our construction as follows:

$\underline{\mathsf{Game}_0 \approx \mathsf{Game}_1}$: These two games are identical except that the common reference string crs is sampled uniformly at random. It works exactly as in FC and due to lack of space it is omitted.

$\underline{\mathsf{Game}_1 \approx \mathsf{Game}_2}$: The two games are indistinguishable according to the fact that the probability of ξ_{Ext} to extract the wrong message out of a commitment is negligible. It works exactly as in FC and due to lack of space it is omitted.

$\underline{\mathsf{Game}_2 \approx \mathsf{Game}_3}$: The indistinguishability of these games follows from the equivocability and extractability of the commitment scheme. Assume towards contradiction that there exists an adversary $\mathcal{A}$ for some non-negligible function such that

$$\Pr\left|[1 \leftarrow \mathsf{Game}_2^{\mathcal{A}}] - \Pr[1 \leftarrow \mathsf{Game}_3^{\mathcal{A}}]\right| \geq \epsilon(\lambda)$$

Then we can build the distinguisher against the equivocability and extractability property of EQTDC as follows:

$\underline{\text{Reduction } \mathcal{B}(\mathsf{crs})}$: $\mathcal{B}$ simulates the inputs of Game_2 plugging crs in the public parameters. In the simulation of the oracle $\mathcal{O}^{\mathsf{ProveCred}}$ it modifies $\mathsf{ProveCred}$ as follows: it evaluates the decryption $M \leftarrow \mathsf{Dec}(\sigma, CT)$ and sends M to the challenger. In response it receives the commitment com_1, then it proceeds with the execution until it recomputes CT. If the check succeeds, then $\mathcal{B}$ sends M to the challenger and it receives decom_1, which is forwarded to $\mathcal{A}$. $\mathcal{B}$ continues with the simulation and outputs 1 if the adversary succeeds and 0 otherwise. It is not hard to check that the reduction is efficient. We note that whenever

com_1 and decom_1 are honestly computed, the reduction perfectly simulates the inputs that the adversary is expecting in Game_2, while when ξ^0_{Eq} and ξ^1_{Eq} are executed by the challenger, $\mathcal{B}$ faithfully simulates Game_3. It follows that

$$\Pr[1 \leftarrow \mathsf{Game}_2^{\mathcal{A}}] = \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}| \\ (\mathsf{com}_1, \mathsf{decom}_1) \leftarrow \mathsf{Commit}(\mathsf{crs}, m) \end{bmatrix}$$

and

$$\Pr[1 \leftarrow \mathsf{Game}_3^{\mathcal{A}}] = \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}|\ \mathsf{com}_1 \leftarrow E^0_{\mathsf{Eq}}(\mathsf{crs}, a); \\ \mathsf{decom}_1 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_1, m) \end{bmatrix}.$$

Therefore, by the initial assumption, we can estimate the success probability of the distinguisher as

$$\left| \begin{matrix} \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}| \\ (\mathsf{com}_1, \mathsf{decom}_1) \leftarrow \mathsf{Commit}(\mathsf{crs}, m) \end{bmatrix} - \\ \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}|\mathsf{com}_1 \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a), \\ \mathsf{decom}_1 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_1, m) \end{bmatrix} \end{matrix} \right| \geq \epsilon(\lambda)$$

Since $\mathsf{crs} \leftarrow \{0,1\}^{\mathsf{poly}(k)} \approx \mathsf{crs} \leftarrow \xi(1^\lambda)$, we have

$$\left| \begin{matrix} \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}|\mathsf{crs} \leftarrow \{0,1\}^k, \\ (\mathsf{com}_1, \mathsf{decom}_1) \leftarrow \mathsf{Commit}(\mathsf{crs}, m) \end{bmatrix} - \\ \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}|\mathsf{crs} \leftarrow \xi(1^\lambda), \\ (\mathsf{com}_1) \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a), \\ \mathsf{decom}_1 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}, m) \end{bmatrix} \end{matrix} \right| \geq \epsilon(\lambda)$$

which is a contradiction to the EQTDC. So, it proves our claim.

$\underline{\mathsf{Game}_3 \approx \mathsf{Game}_4}$: The indistinguishability of the two games comes from the correctness of TDSPE encryption scheme and builds upon the following observations:

- In the simulation of $\mathsf{ProveCred}$, the policy f associated to the ciphertext CT is always known to the challenger (in the case it is given as in input).
- The ciphertext CT is deterministically recomputed so it must be the case that it is generated by $\mathsf{Enc}(\mathsf{pk}, f, .)$.
- The ciphertext CT is uniformly distributed on the ciphertext space: Note that we point out that the string $s'||t'$ is computed as $\xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_0) \oplus r'$ and since r' is uniformly distributed and $\xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_0)$ is independent from r', then $s'||t'$ is distributed uniformly in the set $\{0,1\}^{2^k}$.

As a result, CT is always a ciphertext of the form $\mathsf{Enc}(\mathsf{pk}, f, .)$, for some well-defined policy f, uniformly distributed in its domain. Thus, by the correctness of TDSPE, and for all the set of attributes A associated with σ and for all M, we have that:

- $\mathsf{Dec}(\sigma, \mathsf{Enc}(\mathsf{pk}, f, L, M)) = M$ if $f(\mathsf{A}) = 1$.
- $\mathsf{Dec}(\sigma, \mathsf{Enc}(\mathsf{pk}, f, L, M)) = \perp$, if $f(\mathsf{A}) = 0$.

This implies that Game_4 simulates Game_3 with very intense probability.

$\underline{\mathsf{Game}_4 \approx \mathsf{Game}_5}$: The two games are identical following from the equivocability and extractability of the commitment scheme. More precisely, assume towards contradiction that there exists an adversary $\mathcal{A}$ for some non-negligible function such that

$$\left|\Pr\left[1 \leftarrow \mathsf{Game}_4^{\mathcal{A}}\right] - \Pr\left[1 \leftarrow \mathsf{Game}_5^{\mathcal{A}}\right]\right| \geq \epsilon(\lambda)$$

Then we can build the distinguisher against the equivocability and extractability property of EQTDC. The reduction looks as follows:
$\underline{\text{Reduction } \mathcal{B}(\mathsf{crs})}$: $\mathcal{B}$ simulates the inputs of Game_4 plugging crs in the public parameters. In the simulation of $\mathsf{VerifyCred}$, it modifies the algorithm as follows: The algorithm sends r to the challenger and receive com_0 in response. They continue then with the normal execution until they send r to the challenger and receive decom_0, which is forwarded to $\mathcal{A}$. $\mathcal{B}$ continues with the simulation and outputs 1 if the adversary succeeds and 0 otherwise.

We note that whenever com_0 and decom_0 are honestly computed, $\mathcal{B}$ perfectly simulates the inputs that the adversary is expecting Game_4, while when ξ^0_{Eq} and ξ^1_{Eq} are executed by the challenger, $\mathcal{B}$ faithfully simulates Game_5. It follows that

$$\Pr\left[1 \leftarrow \mathsf{Game}_4^{\mathcal{A}}\right] = \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}| \\ (\mathsf{com}_0, \mathsf{decom}_0) \leftarrow \mathsf{Commit}(\mathsf{crs}, r) \end{bmatrix}$$

and

$$\Pr[1 \leftarrow \mathsf{Game}_5^{\mathcal{A}}] = \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}|\ \mathsf{com}_0 \leftarrow E^0_{\mathsf{Eq}}(\mathsf{crs}, a); \\ \mathsf{decom}_0 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_0, r) \end{bmatrix}$$

So, with the primary assumption, we can evaluate the success probability of the distinguisher:

$$\left|\begin{matrix} \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}| \\ (\mathsf{com}_0, \mathsf{decom}_0) \leftarrow \mathsf{Commit}(\mathsf{crs}, r) \end{bmatrix} - \\ \Pr\begin{bmatrix} 1 \leftarrow \mathcal{B}|\mathsf{com}_0 \leftarrow \xi^0_{\mathsf{Eq}}(\mathsf{crs}, a), \\ \mathsf{decom}_0 \leftarrow \xi^1_{\mathsf{Eq}}(\mathsf{crs}, a, \mathsf{com}_0, r) \end{bmatrix} \end{matrix}\right| \geq \epsilon(\lambda)$$

which is a contradiction to the equivocability and extractability property of EQTDC. This proves our claim.

$\underline{\mathsf{Game}_5 \approx \mathsf{Game}_6}$: The indistinguishability of these two games comes from the overwhelming probability of ξ_{Ext} to extract the correct message out of a commitment computed by the adversary. Clearly, Game_5 and Game_6 only differ in the case that the success of the adversary is determined by the check $s = \mathsf{VerCom}(\mathsf{crs}, \mathsf{com}_1, \mathsf{decom}_1)$ in the former and by $s = \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_1)$ in the latter. Thus, the two games differ only whenever $\mathsf{VerCom}(\mathsf{crs}, \mathsf{com}_1, \mathsf{decom}_1) \neq \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_1)$. By the equivocability and extractability property of EQTDC we have that

$$\Pr[\mathsf{VerCom}(\mathsf{crs}, \mathsf{com}_1, \mathsf{decom}_1) \neq \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_1)] \leq \epsilon(\lambda) \ ,$$

therefore

$$\Pr[1 \leftarrow \mathsf{Game}_5^{\mathcal{A}}] - \Pr[1 \leftarrow \mathsf{Game}_6^{\mathcal{A}}] \leq \epsilon(\lambda).$$

($\mathsf{Game}_0 \approx \ldots \approx \mathsf{Game}_6$). As mentioned above, the adversary's success probability is negligibly different between each pair of neighboring experiments. Since a polynomially-bounded sum of negligible functions is still a negligible function, we have a negligible difference between the success probability of the adversary in Game_0 and in Game_6. Thus, to prove our lemma, we only need to show that the adversary's advantage in Game_6 is bounded by a negligible function in the security parameter.
Intuitively, if the adversary is able to win the game with more than negligible probability, then it would imply that the adversary can break the CPA property of TDSPE encryption scheme with the same probability assumed to be infeasible. More precisely, assuming towards contradiction that there exists an adversary $\mathcal{A}$ such that

$$\Pr[1 \leftarrow \mathsf{Game}_6] \geq \epsilon(\lambda)$$

Then, we can construct the reduction against the CPA of TDSPE encryption scheme as follows.

$\underline{\text{Reduction } \mathcal{B}(\mathsf{pk})}$: $\mathcal{B}$ plugs pk into the public parameters and declares two random messages (r_0, r_1) along with the f^*. $\mathcal{B}$ simulates Game_6 by keeping the same lookup lists except that the oracle $\mathcal{O}^{\mathsf{Issue}}(\mathsf{Id}, \mathsf{A})$, when queried on a user $\mathsf{Id} \in \mathcal{HU}$ only records the attributes vector A without generating the corresponding credential. $\mathcal{B}$ instead generates credentials for the user Id only upon a query Id from $\mathcal{A}$ to the oracle $\mathcal{O}^{\mathsf{Corrupt}}(\mathsf{Id})$. Credentials are generated by querying the challenger for the related set of attributes A and setting σ to be the answer dk_{Θ}. The rest of Oracles remain consistent. In the simulation of $\mathsf{ProveCred}$ ($\mathsf{DelegIssue}$, respectively), $\mathcal{B}$ initially samples two random messages (r_0, r_1) along with the predicates (f^*). In the simulation of $\mathsf{ProveCred}$, the challenger returns CT^* and $\mathcal{B}$ sets $CT = CT^*$. When $\mathcal{A}$ returns com_1 in the next step, $\mathcal{B}$ computes $m^* \leftarrow \xi_{\mathsf{Ext}}(\mathsf{crs}, a, \mathsf{com}_1)$ and returns 1 if $m^* = r_1$, and 0 if $m^* = r_0$ otherwise it flips a random coin.

We now show that $\mathcal{B}$ perfectly simulated the Game_6 to $\mathcal{A}$. First, we argue that in the simulation of the oracles there is no significant difference: The decryption algorithm is in fact no longer evaluated in $\mathcal{O}^{\mathsf{ProveCred}}$ and its execution relies on the attributes vector A associated with the input credential σ. So, it is sufficient to generate the credentials only when the adversary corrupts a certain user. We now point out that the ciphertext CT^* is correctly distributed from the adversary's point of view. Note that the uniform randomness r of the ciphertext CT which is revealed to $\mathcal{A}$ in the last phase of the simulation is not executed due to the early interruption of $\mathcal{A}$. On the other hand, com_0 is also computed independently from r. Therefore, the randomness of CT^* is also sampled uniformly at random, it follows that the ciphertext CT^* is also correctly distributed according to the view of the adversary. Furthermore, it is clear that the tuple (r_0, r_1) is a valid input for the challenger since the experiment requires that all the corrupted users (thus all credentials requested to the challenger) do not satisfy the target policy f^*. Now we provide a bound for the success probability of the reduction in the CPA game of TDSPE encryption scheme. By assumption we have that $|\Pr[1 \leftarrow \mathsf{Game}_6]| \geq \epsilon(\lambda)$, which means that with the same probability $CT^* = \mathsf{Enc}(\mathsf{pk}, f^*, L, s) = \mathsf{Enc}(\mathsf{pk}, f^*, L, m^*)$. We define this event by guess. Note that the probability of the event guess to happen is independent of the random coins of the challenger (both r_0 and r_1 are correctly distributed), so we can rewrite the success probability of the reduction of CPA as:

$$\mathcal{A}^{\mathcal{B}}(k) = \frac{1}{2} - \begin{pmatrix} \Pr[1 \leftarrow \mathcal{B} | b = 1 \textbf{ and guess}] \Pr[b = 1] \\ + \Pr[0 \leftarrow \mathcal{B} | b = 0 \textbf{ and guess}] \Pr[b = 0] \end{pmatrix} \cdot \text{guess}$$

$$+ \begin{pmatrix} \Pr[1 \leftarrow \mathcal{B} | b = 1 \text{ and } \overline{\textbf{guess}}] \Pr[b = 1] \\ + \Pr[0 \leftarrow \mathcal{B} | b = 0 \text{ and } \overline{\textbf{guess}}] \Pr[b = 0] \end{pmatrix} \cdot \text{guess}$$

Whenever the event guess does not happen we can upper bound the success probability of the reduction by $\frac{1}{2}$, so, by our initial assumption, we have:

$$\mathcal{A}^{\mathcal{B}}(k) \geq \frac{1}{2} - \begin{pmatrix} \Pr[1 \leftarrow \mathcal{B} | b = 1 \textbf{ and guess}] \Pr[b = 1] + \\ \Pr[0 \leftarrow \mathcal{B} | b = 0 \textbf{ and guess}] \Pr[b = 0] \end{pmatrix} \epsilon(\lambda)$$

$$+ \frac{1}{2}(1 - \epsilon(\lambda))$$

On the other hand whenever guess happens, the reduction successfully guesses the bit of the challenger with probability 1, therefore the advantage of $\mathcal{B}$ is:

$$\mathcal{A}^{\mathcal{B}}(k) \geq \frac{1}{2} - \left(1 \cdot \epsilon(\lambda) + \frac{1}{2} \cdot (1 - \epsilon(\lambda))\right) \geq$$
$$\frac{1}{2} - \left(\frac{1}{2} + \frac{1}{2} \cdot \epsilon(\lambda)\right) \geq \frac{1}{2}\epsilon(\lambda)$$

which is a non-negligible function in the security parameter. This is a contradiction to the CPA property of TDSPE and it concludes our proof.

Lemma 5.4.3 (Anonymity)**.** *Let* TDSPE *be a secure threshold delegatable subset predicate encryption scheme and* EQTDC *be an equivocable and extractable commitment scheme, then* TDAC *is anonymous.*

Proof. The argumentation follows exactly the one in [DMM+18]. Therefore, we will not recall the full proof.

Verifiable partial keys. We note that the well-formedness of the partial secret keys issued by the single authorities can be realized by standard means, i.e., by adding a Non-Interactive Zero-Knowledge proof (NIZK) demonstrating the well-formedness. In our concrete approach, this would require standard Schnorr-type proofs about discrete-logarithm relations that can be efficiently instantiated.

5.4.4 Potential Extensions

Subsequently, we discuss two potential extensions of our approach, whose detailed study we consider as future work.

Blind issuing of attributes. Issuing credentials for attributes without the issuer learning the attributes is a feature that is useful when encoding a user's secret key into a credential, when attributes should be kept private for other reasons or when users need to transfer their attributes among credentials blindly. While the encoding of the user's key is required in Coconut [SAB+19] (as a private attribute, otherwise signatures are forgeable) and thus this feature is an inherent part of the scheme we do not include user-specific secrets into the credential. Using blind issuing for some other reason than in Coconut heavily depends on the applications. However, we do not consider this feature as an integral feature of the applications discussed in our work. Nevertheless, this feature can be generically added by employing a standard two-party computation protocol. Consequently, one can achieve this feature on top of TDAC using a similar technique as used by Coconut [SAB+19] if needed. Informally, a user could compute a blind version of attributes using ElGamal encryption as follows: Generate an ElGamal key-pair $(d, \omega = g_1^d)$, pick a random k and compute $m = \{B = g_1^k, c = \omega^k \cdot H(\alpha)\}$ and send m to the issuing authorities (along with a NIZK proof of well-formedness). Each authority can respond with a partial key as $D = B^{r'_i}, h_i = g_2^{r'_i}, K_i = g_1^{x_i} \cdot (c^{\lambda_i^{-1}})^{r'_i}$, where r' is randomly chosen by each authority. Now the user can unblind the keys and compute a full key as $dk = (h = \prod_{i \in \mathsf{ST}} h_i, \frac{\prod_{i \in \mathsf{ST}} K_i^{\lambda_i}}{\prod D^d})$.

Revocation. In many scenarios, the revocation of credentials represents an important property. A straightforward solution would be to apply time discretization and, at the beginning of each time epoch, set up the parameters anew, thus invalidating all old credentials. This is, for instance, a common approach in group signatures [BCC+16]. Alternatively, one could also aim for accumulator-based deny-lists, i.e., one needs to

demonstrate that a credential is not one of the revoked ones in the current list of revoked credentials. However, this direction is less straightforward when constructing TDAC based on TDSPE schemes, as we do not incorporate user identifiers or user specific secrets into the credentials. Consequently, an efficient revocation mechanism for our approach that does not rely on re-issuing is an interesting question for future research.

5.5 Performance Evaluation

This section shows our evaluation results corresponding to the proposed scheme in terms of computation and communication overhead.

5.5.1 Experimental Results

To experimentally analyze the performance we have implemented our TDAC system and the underlying TDSPE. As EQTDC we use the well known construction $\mathsf{com} := H(m, r)$ with H being a random oracle and $r \leftarrow \{0,1\}^k$. The benchmarks are performed on a PC with an Intel Core i5-6200U CPU at 2.30 GHz, 8 GB RAM running Ubuntu 16.04.1. Our implementation is written in Java based upon the `upb.crypto` library[3] with `mcl` bindings[4]. We use the pairing friendly curve BN256 which provides a security level of around 100 bit.

Selective showings. In a selective disclosure credentials application, a prover can prove possession of a subset of credentials (attributes). That is, a prover who has obtained a collection of credentials that include attributes, can selectively reveal his attributes. For our evaluation, we take the execution time of each algorithm for a number of attributes of 5 and 10 over 10000 iterations. We also assume a threshold $t = 2$ with a total number of issuers $n = 5$. We set our threshold as honest majority ($n/2 < t$) to prevent malicious authorities from issuing credentials arbitrarily. For simplicity, let us assume a KeyGen algorithm that includes both ShareKeyGen and KeyGenComb algorithms. Moreover, this KeyGen demonstrates the issuing credentials cost in TDAC (note that credentials refer to decryption keys in the underlying scheme). The results are shown in Table 6.2, where delegation considers that one more attribute is added to the key. Verifying is faster than proving credentials—due to the double-check of encryption operation in the latter. The total time of about 5 ms necessary for the whole verification phase makes our implementation suitable for time-critical applications like public transportation, ticketing, etc.

Moreover, the trend of increasing the effective parameters (t, q) and their effect on computation time is shown in Fig 5.6. Since the Setup algorithm runs only once, we do not consider the computation time of Setup. The measured computation time of KeyGen, which includes both the ShareKeyGen and KeyGenComb algorithms, demonstrates the costs associated with issuing credentials in TDAC. It considers a threshold t range from 2 to

[3] https://github.com/cryptimeleon/math
[4] https://github.com/cryptimeleon/mclwrap

Table 5.1: Execution times for TDSPE and TDAC protocols in milliseconds.

	TDSPE					TDAC	
	Setup	KeyGen	Enc	Delegate	Dec	ProveCred	VerifyCred
$q = 10$	4.5	1.65	1.70	1.2	2.45	3.13	1.80
$q = 5$	3	1.12	1.34	1	1.50	2.74	1.44

10 and an attribute set size range from 5 to 20 (refer to Fig 5.6(a)). In contrast, the computation time of other algorithms remains independent of the number of authorities (refer to Fig 5.6(b)). Observe that we evaluate our scheme using large parameters like $q = 100$. Even in that case the running time of ProveCred and VerifyCred is below 10 ms, and the total time of these algorithms is below 20 ms, which demonstrates the scalability of our system. Because the standard deviation in our measurements is low, we chose to omit them.

DNF formula: We recall the encoding for DNFs from Section 5.3.1 and use η to show the number of disjunctive clauses in a DNF expression such that each clause represents a conjunction over some subset of the attributes as $C_j = \bigwedge_i \alpha_{i,j}$. Let us present a DNF expression via an example: Assume, we have two clauses C_1 and C_2 such that the former requires 3 attributes and the later requires 5 attributes. We denote this as $\eta_2 = (3, 5)$:

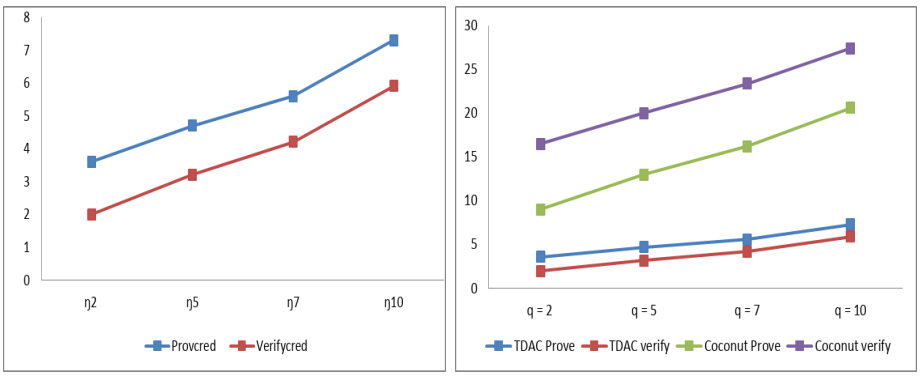
$$\eta_2 = (3, 5) = \left(C_1 = \bigwedge_{i=3} \alpha_{i,1}, C_2 = \bigwedge_{i=5} \alpha_{i,2} \right).$$

Suppose a user has a credential that contains five attributes $q = 5$. Now we show the trend of increasing the parameter η from 2 to 10 and its effect on computation time of ProveCred and VerifyCred algorithms (see Fig 5.5(a)). More precisely, the parameter η sets as follows: $\eta_2 = (3, 5), \eta_5 = (2, 3, 5, 6, 7), \eta_7 = (2, 3, 5, 6, 7, 8, 9)$, and $\eta_{10} = (2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$.

Comparison with Coconut. Finally, we present an approximate comparison between TDAC and Coconut library[5] (the most relevant work) in Fig. 5.5(b). They implement the primitives in Python with native OpenSSL as backend.[6] The pairing is defined over the BN256 curve. We take running times of ProveCred and VerifyCred protocols for the parameter q from 2 to 10, while assuming the DNF policy as η_q (like above) for TDAC (e.g., for $q = 2$ is $\eta_2 = (3, 5)$) and only the required private attributes for Coconut (public

[5] https://github.com/asonnino/coconut

[6] https://github.com/dfaranha/OpenPairing)



(a) The running times of TDAC in DNF version (ms)

(b) Comparison between TDAC and Coconut (ms)

Figure 5.5: The running times of TDAC in DNF version and Coconut

attributes are revealed in plain). We mention that this is only an approximate comparison, but is useful as an order of magnitude analysis on specific parameters.

5.5.2 Theoretical Analysis and Comparison

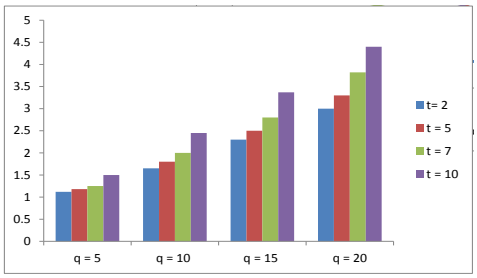
Subsequently, we analyze the computational and communication complexity of our approach in order to compare it to existing approaches.

5.5.2.1 Computational Complexity

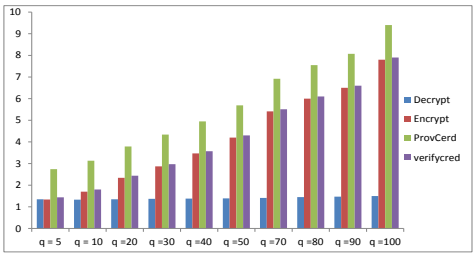
To analyze the efficiency of our TDAC scheme, we consider the number of exponentiations required for the showing of a credential, since this will be the most frequently executed operation. We summarize the efficiency analysis in comparison with related works in Table 5.2. Here, q refers to the number of attributes to be certified. We denote by $E_{\mathbb{G}_1}$ the cost of an exponentiation in $\mathbb{G}_1$ (resp. $E_{\mathbb{G}_2}$, $E_{\mathbb{G}_T}$) and P represents the cost of a pairing computation. $\mathsf{POK}\{E_{\mathbb{G}_2}[q+1]\}$ (resp. $\mathsf{POK}\{P[q+1]\}$) denotes the cost of proving knowledge of q secret scalars involved in a $\mathbb{G}_2$ (resp. P) multi-exponentiation, and $\mathsf{Ver}(\mathsf{POK})$ the cost of verifying this proof. To be consistent with related works, our computation is intended for selective disclosure of attributes. This computation would be $\eta(qE_{\mathbb{G}_1} + E_{\mathbb{G}_2} + E_{\mathbb{G}_T})$ for a DNF expression, where η is the number of clauses in the DNF.

5.5.2.2 Communication Complexity

We analyze the communication complexity and the size of element exchanged (bandwidth consumed). The communication complexity is expressed as functions of the threshold number of authorities t and q being an upper bound on the number of attributes. More precisely, IssueCred depends on the number of authorities, while ProveCred and VerifyCred



(a) KeyGen based on (t, q)



(b) Other algorithms based on q

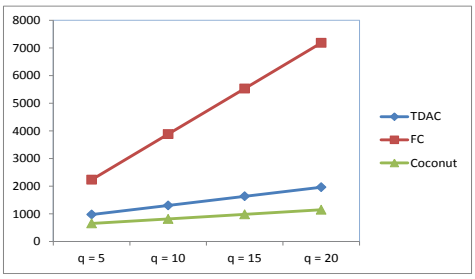
Figure 5.6: The running times of KeyGen (i.e., issuing credentials in TDAC) and other algorithms (ms)

Table 5.2: Computational complexity

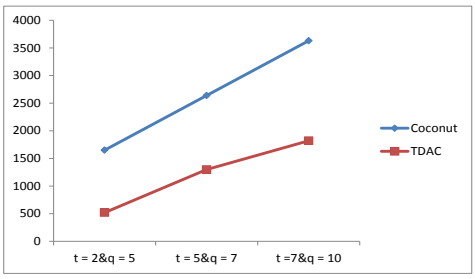
	Presentation proof cost	
	ProveCred	VerifyCred
FC [DMM+18]	$11P + 5(q-1)E_{\mathbb{G}_1} + (5q+1)E_{\mathbb{G}_1} + E_{\mathbb{G}_T}$	$(5q+1)E_{\mathbb{G}_1} + E_{\mathbb{G}_T}$
Coconut [SAB+19]	$3E_{\mathbb{G}_1} + \mathsf{POK}\{E_{\mathbb{G}_2}[q+1]\}$	$\mathsf{Ver}(\mathsf{POK}) + 2P$
TDAC	$2P + qE_{\mathbb{G}_1} + E_{\mathbb{G}_2} + E_{\mathbb{G}_T}$	$qE_{\mathbb{G}_1} + E_{\mathbb{G}_2} + E_{\mathbb{G}_T}$

algorithms are only dependent on the number of attributes. As an example, we calculate the size (in bytes) of each message exchanged in the credentials scheme for five attributes $q = 5$ and the threshold $t = 2$ in Table 5.3. We use SHA-2 as hash function (for commitments) which has output size of 32 bytes. Moreover, the sizes in bytes of elements of the bilinear groups are $|\mathbb{G}_1| = 64$, $|\mathbb{G}_2| = 128$, and $|\mathbb{G}_T| = 384$.

We evaluate the trend of increasing the effective parameters (t, q) and their effect on communication complexity of showing and issuing credentials in our construction with related works in relation to q (from $q = 5$ to $q = 20$) in Fig 5.7(a). Compared to FC, our showing is at least 50% smaller. Compared to Coconut, TDAC is comparable. Regarding issuing credentials, we show this evaluation only between TDAC and Coconut in Fig 5.7(b). Note that both TDAC and Coconut linearly depend on the number t threshold authorities. However, in TDAC it does not depend on q and we have a communication size of about 40% compared to Coconut, even in the case we assume $q = 1$ (which is the best case for Coconut, as it linearly grows with q).



(a) Showing (indep. of t)



(b) Issuing

Figure 5.7: The transaction size of the verification and the issuing algorithm (bytes)

Table 5.3: Communication complexity in bytes ($t = 2$ and $q = 5$)

Schemes.Transaction	**Complexity**	**Size(bytes)**
TDAC.IssueCred	$(2\|\mathbb{G}_1\|, \|\mathbb{G}_2\|)t$	520
TDAC.ProveCred	$(q)\|\mathbb{G}_1\| + \|\mathbb{G}_T\| + \|\mathbb{G}_2\|$ $+2\|Z_p\| + 2\|\mathsf{Hash}\|$	974
FC.IssueCred	$11\|\mathbb{G}_1\|$	726
FC.ProveCred	$(5q+1)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$ $+2\|Z_p\| + 2\|\mathsf{Hash}\|$	2232
Coconut.IssueCred	$t(3\|\mathbb{G}_1\|) + (q+2)\|\mathbb{G}_1\|+$ $\mathsf{POK}\{(2q+2)\|Z_p\| + (q+1)\|\mathbb{G}_1\|\}$	1650
Coconut.ProveCred	$3\|\mathbb{G}_1\| + \|\mathbb{G}_2\|+$ $\mathsf{POK}\{(q+1)\|Z_p\| + 1\|\mathbb{G}_2\|\}$	652

Communication rounds. For issuing credentials, both schemes Coconut and TDAC are similar as they need a single round communication between a user and $t + 1$ authorities. Moreover, they are non-interactive in terms of interaction between authorities, which means authorities do not need to interact with each other. For showing credentials, TDAC requires a 3-round interactive protocol. In contrast, Coconut can be a non-interactive protocol (assuming the Fiat-Shamir heuristic). Communication rounds of showing credentials of FC scheme is similar to TDAC.

The attribute and parameters domain. TDAC construction has compact parameters (constant size) independent of the number of attributes. The attribute domain is unbounded

as we can compute all Y_i in our TDSPE scheme by means of hashing as $Y_i = H(\alpha_i)$ or $Y_i = H(\alpha_i||v_i)$ for some attribute name α_i and value v_i to map these values to group elements in $\mathbb{G}_1$. Clearly, attributes (and values) here can be an arbitrary bit string. In Coconut, they also use hash functions to map an arbitrary bit string to $\mathbb{G}$ and $\mathbb{Z}_p$. However, Coconut does not provide constant size parameters as public keys and parameters grow linearly with to the number of attributes (this is also the case for FC).

5.5.3 Comparison

We provide a qualitative comparison of TDAC with some of the most prominent AC schemes in Table 6.3.5 by means of selected criteria. In particular, by Threshold Issuance (TI), we denote whether multiple authorities are involved in the issuing credentials protocol and a credential can be aggregated using partial credentials issued by a subset of authorities. Note, the authorities do not need to communicate with each other in this phase (a non-interactive way). Here, × denotes that they are not designed for use in a multi-authority setting.

Table 5.4: Comparison of some popular credential schemes (q is the number of attributes).

Scheme	**TI**	**Express**	**Delegate**	**Security**	**|Cred|**
FC [DMM+18]	×	CNF/DNF	×	✓	11
U-Prove [PZ11]	×	S	×	×	$O(q)$
CL [CL04]	×	R	×	✓	$O(q)$
Idemix [BCHB+09]	×	R	×	✓	$O(q)$
GGM [GGM14a]	×	S	×	✓	2
CDD [CDD17]	×	S	✓	✓	$O(q)$
Coconut [SAB+19]	✓	R	×	×	2
TDAC	✓	DNF/S	✓	✓	2

Moreover, we compare the expressiveness (*Express*) of the supported policies, where R denotes for arbitrary relations over attributes, S stands the selective disclosure of attributes, CNF as well as DNF represents conjunctive and disjunctive normal form respectively. With $|Cred|$ we represent the size of the credential. By *Security*, we show whether the proposed schemes provide a formal model and rigorous security proofs.

Briefly summarizing Table 6.3.5, we see that TDAC improves on the computational overhead by presenting the most efficient constructions in terms of credential size and verification among the most well-known ACs [CL04], [PZ11], [GGM14a] and idemix [BCHB+09]. Also, other schemes have larger credentials, where often their size grows linearly with the number of attributes. Moreover, they are not delegatable.

Coconut [SAB+19] extends these related schemes and presents a short, aggregatable, and randomizable credential scheme, allowing threshold issuing. However, compared to TDAC, Coconut suffers from a lack of formal security definitions and proofs. Moreover,

TDAC is the only scheme that supports threshold issuing and delegatable credentials and still offers compact credentials and reasonable performance.

Finally, for a fair comparison, we should mention that although various AC schemes without our features are more expressive and in particular allow to prove arbitrary relations over attributes, this is not a huge limitation. Firstly, this allows us to avoid potentially costly zero-knowledge proofs and secondly TDAC still supports fairly complex functionalities (i.e., DNF formulas). Moreover, we believe that this functionality is sufficient for a host of applications, and in particular access control, where it is typically not required to prove relations over attributes, but the proof of possession of attributes (aka selective showing) is sufficient.

6 Privacy-Preserving, Single Sign-On

As mentioned in the introduction, in current single sign-on authentication schemes on the web, users are required to interact with identity providers securely to set up authentication data during a registration phase and receive a token (credential) for future access to services and applications. This type of interaction can make authentication schemes challenging in terms of security and availability. From a security perspective, a main threat is theft of authentication reference data stored with identity providers. An adversary could easily abuse such data to mount an offline dictionary attack for obtaining the underlying password or biometric. From a privacy perspective, identity providers are able to track user activity and control sensitive user data. In terms of availability, users rely on trusted third-party servers that need to be available during authentication.

We propose a novel decentralized privacy-preserving single sign-on scheme through the Decentralized Anonymous Multi-Factor Authentication (DAMFA), a new authentication scheme where identity providers no longer require sensitive user data and can no longer track individual user activity. Moreover, our protocol eliminates dependence on an always-on identity provider during user authentication, allowing service providers to authenticate users at any time without interacting with the identity provider. It also offers enhanced security by ensuring that users do not need to store sensitive information on their personal devices. This feature becomes particularly beneficial when a user's device gets compromised.

Our approach builds on threshold oblivious pseudorandom functions (TOPRF) to improve resistance against offline attacks and uses a distributed transaction ledger to improve availability. We prove the security of DAMFA in the universally composable (UC) model by defining a UC definition (ideal functionality) for DAMFA and formally proving the security of our scheme via ideal-real simulation. Finally, we demonstrate the practicability of our proposed scheme through a prototype implementation.

6.1 Building blocks

We begin by introducing the building blocks necessary to build a privacy-preserving SSO.

6.1.1 Oblivious Pseudo-random Function (OPRF)

This primitive is taken from [JL09, JKKX17] as follows:

> An oblivious pseudo-random function (OPRF), cf. [JL09] is a protocol involving two parties: a sender and a receiver. It securely computes $F_k(x)$ where both x

Key and server initialization. A random key $k \leftarrow Z_p$ is secret shared using Shamir's scheme with parameters n, t; each server $S_i, i \in [n]$, receives a share k_i. Also, they use a hash function as $H_g : M \rightarrow \mathbb{G}$.
Threshold oblivious computation of $F_k(x)$:

- On input x, user U picks $r \leftarrow Z_p$ and computes $A := H_g(x)^r$; it chooses a subset SR of $[n]$ of size $t+1$ and sends to each server $S_i, i \in \mathsf{SR}$, the value A and the subset SR.
- Upon receiving the message A from U, server S_i verifies that $A \in \mathbb{G}$ and, upon successfuly verification, responds with $b_i := A^{\lambda_i \cdot k_i}$ where λ_i is a Lagrange interpolation coefficient for index i and subset SR.
- When U receives messages b_i from each server $S_i, i \in \mathsf{SR}$, U outputs the value

$$H\left(x, \left(\prod_{i \in \mathsf{SR}} b_i\right)^{1/r}\right)$$

as the result of $F_k(x)$.

Figure 6.1: (n, t)-threshold computation in a TOPRF protocol [JKKX17]

and k are inputs of the sender and receiver, respectively. The protocol ensures that no party learns anything other than the input holder, who learns $F_k(x)$.

A threshold OPRF (TOPRF, cf. [JKKX17]) is an extension of the OPRF that enables a group of servers to secret-share a key k for a pseudo-random function (PRF) F. The servers use a shared PRF evaluation protocol to allow the user to compute $F_k(x)$ on an input x. In this setting, both x and k remain secret if no more than t out of n servers are corrupted (see Fig. 6.1).

A formal definition of the TOPRF protocol as a realization of the TOPRF functionality is given in Fig. 6.2. Note that we just duplicate these functionalities so that readers can easily follow our ideal functionality and construction (for more details see [JKKX17]).

6.1.2 Public Append-Only Ledger

A ledger allows us to keep a list of public information and maintains the integrity of the dataset. It guarantees a consistent view of the ledger for every party. Every user can insert information into the ledger and, once some data is uploaded, nobody can delete or modify it. Moreover, the ledger assures the correctness of pseudonyms and guarantees that no one can impersonate another participant to release information. Furthermore, it distributes up-to-date data to all participants. In this chapter, we construct our system using a public append-only ledger (blockchain). There are already some works constructing advanced applications based on this assumption, such as [SCG+14, GGM14b, FVY14]. Yang et al. [YAXY19] formally define a public append-only ledger which we use for constructing our DAMFA system (see Fig. 6.3).

Assume $tx(p, S)$ and $T(p, x)$ are undefined for all p, x, S.

Initialization.

- On message $(Init, \mathsf{sid}, \mathsf{SI})$ from S, ignore if $|\mathsf{SI}| \neq n$ or S is active. Otherwise, mark S as "active" and if no record $\langle \mathsf{sid}, [\ldots] \rangle$ exists, let $t^* = \mathsf{Corrupted}$ be the subset of SI that is corrupted. If $t^* \leq t$ then picks any previously unused label p and records $\langle \mathsf{sid}, \mathsf{SI}, p \rangle$. Sends $(Init, \mathsf{sid}, S, \mathsf{SI}, p)$ to $\mathcal{A}^*$.
- On message $(Init, \mathsf{sid}, \mathcal{A}^*, p)$ from $\mathcal{A}^*$, check that p is a label that has not been used before, record $\langle \mathcal{A}^*, p \rangle$ and return $(Init, \mathsf{sid}, \mathcal{A}^*, p)$ to $\mathcal{A}^*$.
- On message $(InitComplete, \mathsf{sid}, S)$ from $\mathcal{A}^*$, retrieve tuple $\langle \mathsf{sid}, \mathsf{SI}, p \rangle$. Ignore the message if there is no such tuple, $S \notin \mathsf{SI}$, or not all servers in SI are active. Otherwise, send $(InitComplete, \mathsf{sid})$ to S and mark S as "initialized".

Evaluation.

- On message $(\mathsf{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{SE}, x)$ from $P \in \{U, \mathcal{A}^*\}$, retrieve $\langle \mathsf{sid}, \mathsf{SI}, p \rangle$ if $P = U$ or $\langle A^*, p \rangle$ if $P = \mathcal{A}^*$. Ignore the message if there is no such tuple or if $|\mathsf{SE}| \neq t + 1$. Otherwise, record $\langle \mathsf{ssid}, P, p, \mathsf{SE}, x \rangle$ and send $(\mathsf{Eval}, \mathsf{sid}, \mathsf{ssid}, P, \mathsf{SE})$ to $\mathcal{A}^*$.
- On message $(SndrComplete, \mathsf{sid}, \mathsf{ssid}, S)$ from $\mathcal{A}^*$, retrieve tuple $\langle \mathsf{sid}, \mathsf{SI}, p \rangle$. Ignore the message if there is no such tuple, if $S \notin \mathsf{SI}$, or if S is not initialized. Otherwise, set $tx\,(p, S)$ ++ (or set it to 1 if $tx\,(p, S)$ is undefined) and send $(SndrComplete, \mathsf{sid}, \mathsf{ssid})$ to S.
- On message $(RcvComplete, \mathsf{sid}, \mathsf{ssid}, P = \{U, \mathcal{A}^*\}, p^*)$ from $\mathcal{A}^*$, retrieve $\langle \mathsf{ssid}, P, p, \mathsf{SE}, x \rangle$. Ignore the message if there is no such tuple or if any of the following conditions fail: (i) if $p^* = p$ then $|\{S \in \mathsf{SI} \mid tx\,(p, S) > 0\}| > t$, (ii) if all servers in SE are honest then $p^* = p$. Otherwise, if $p^* = p$ then set $tx\,(p, S)$ -- for any $t + 1$ distinct $S \in \mathsf{SI}$ s.t. $tx\,(p, S) > 0$. Then, if $T\,(p^*, x)$ is defined, send $(\mathsf{Eval}, \mathsf{sid}, T\,(p^*, x))$ to P. Otherwise, pick $\rho \leftarrow \{0,1\}^l$ and set $T\,(p^*, x) := \rho$. Finally, send $(\mathsf{Eval}, \mathsf{sid}, \mathsf{ssid}, \rho)$ to P.

Figure 6.2: Functionality F_{TOPRF} [JKKX17]

6.1.3 Dynamic Accumulators

A dynamic accumulator is a fundamental concept that enables the accumulation of a substantial set of values into a single quantity known as the accumulator. Each value within the accumulator has a corresponding witness, serving as evidence that certifies the presence of that value in the accumulator. The proof of showing that a value is part of an accumulator can be zero-knowledge proof, which reveals neither the value nor the witness to the verifier. Camenisch et al. [CKS09] define a concrete construction of dynamic accumulators with the five algorithms AccSetup, AccAdd, AccUpdate, AccWitUpdate, and AccVerify:

- AccSetup: This is the algorithm to output the public parameters. Select bilinear groups $\mathsf{pp}_{BM} = (q, \mathbb{G}, \mathbb{G}_T, e, g)$ with a prime order p and a bilinear map e. Select $g \in \mathbb{G}$. Select $\gamma \in Z_p$. Generate a key pair msk and pk for a secure signature scheme. Compute and publish $\{p, \mathbb{G}, T, e, g, g_1 = g^{\gamma_1}, \ldots, g_n = g^{\gamma_n}, g_{n+2} = g^{\gamma^{n+2}}, \ldots, g_{2n} = g^{\gamma^{2n}}\}$ and $z = e(g, g)^{\gamma_{n+1}}$ as the public parameters.
- $\mathsf{AccAdd}(sk_A, i, \mathsf{acc}_V, \mathsf{state}_U)$. Compute $\omega = \prod_{j \in V}^{j \neq i} g_{n+1-j+i}$ and a signature σ_i on $g_i \parallel$

F_{B} executes the following steps with parties $\{\mathsf{PA}_{\mathsf{i}1}, \ldots, \mathsf{PA}_{\mathsf{i}n}\}$ and an ideal adversary $\mathcal{S}$ as follows:

- Initialize. Initialize creates an empty list L_p in the beginning.
- Store. On input $(\mathsf{Store}, \mathsf{PA}_{\mathsf{i}i}, \mathsf{nym}_u^o, M)$, checks that nym_u^o is a valid pseudonym for $\mathsf{PA}_{\mathsf{i}i}$, then stores the tuple (nym_u^o, M) to L_p and declares to $\mathcal{S}$ that a new item was appended to the list L_p.
- Retrieve. On input $(\mathsf{Retrieve}, \mathsf{PA}_{\mathsf{i}i})$, returns the list L_p to $\mathsf{PA}_{\mathsf{i}i}$.

Figure 6.3: Functionality F_B [YAXY19]

i under signing key sk. The algorithm outputs $wit_i = (\omega, \sigma_i, g_i)$, an updated accumulator value $\mathsf{acc}_{V \cup i} = \mathsf{acc}_V \cdot g_{n+1-i}$, and $\mathsf{state}_{U \cup i} = (U \cup \{i\}, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n})$.

- AccUpdate: This is the algorithm to compute the accumulator using the public parameters. The accumulator acc_V of V is computed as

$$\mathsf{acc}_V = \prod_{i \in V} g_{n+1-i}$$

- AccWitUpdate: This is the algorithm to compute the witness that values are included in an accumulator, using the public parameters. Given V and the accumulator acc_V, the witness of values $i_1, \ldots, i_k$ in U is computed as

$$\omega' = \omega \cdot \frac{\prod_{j \in V/V_\omega} g_{n+1-j+i}}{\prod_{j \in V_w/V} g_{n+1-j+i}}$$

- AccVerify: This is the algorithm to verify that values in U are included in an accumulator, using the witness and the public parameters. Given acc_V, state_U, and ω, accept if

$$\frac{e(g_i, \mathsf{acc}_V)}{e(g, \omega)} = z$$

As Camenisch et al. [CKS09] point out, the purpose of an accumulator is to have accumulator and witnesses of size independent of the number of accumulated elements.

6.2 Decentralized Anonymous Multi-Factor Authentication (DAMFA)

We build a new practical Decentralized Anonymous Multi-Factor Authentication scheme (DAMFA), where the process of user authentication no longer depends on a single trusted third party. The scheme also permits services where authenticating users remain anonymous within a group of users. Subsequently, our scheme does not require the IdP to be online during the verification. To protect the private key of their user, we use personal identity agents as auxiliary devices that participate in a threshold secret sharing scheme to store the distributed private key of the user.

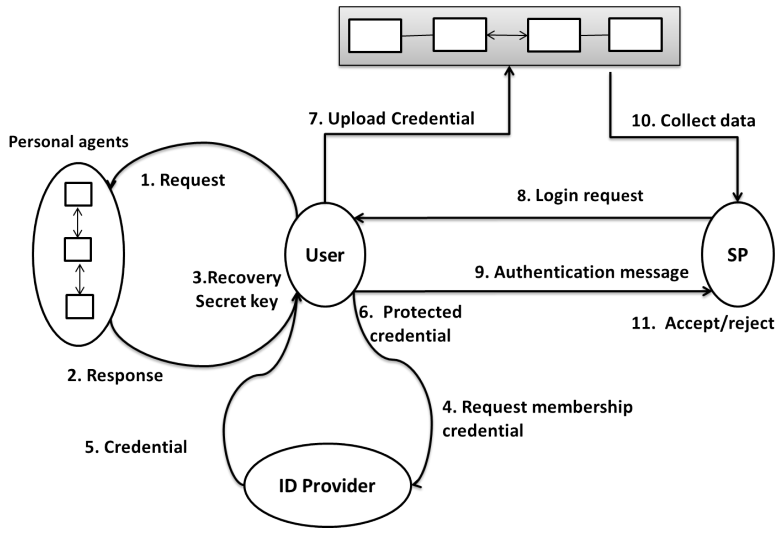


Figure 6.4: A system model of the DAMFA scheme

6.2.1 System Model

The overall system model of DAMFA is shown in Fig. 6.4. The protocol is executed between four participants:

- User U: A user who wants to access various services offered by different service providers. During the registration phase (which runs only once), U obtains a biometric template Bio from a sensor and chooses a password pw. In the authentication phase, users U interact with a set of personal identity agents to authenticate themselves in an anonymous manner.

- Personal identity agent $\mathsf{PA_i}$: We associate each user with a set of personal agents which are auxiliary devices that assist a user in creating a credential for authentication. These personal agents remain under the administrative control of their associated users, who can freely choose where to run them. For example, they could be run on a smart home controller, at a cloud provider, or even on a mobile phone. U generates a private key and executes threshold secret sharing on the private key to generate secret shares of that private key. The user stores the secret shares among their personal agents such that each $\mathsf{PA_i}$ has one share of the overall secret key.

- Service provider (verifier) SP: These are the service providers (untrusted and distributed servers) that require authentication from a user U. After verifying a user's credentials, they provide access to the corresponding service.

- Identity provider IdP: The identity provider is an entity that issues credentials to users. These credentials grant permission to use specific services by proving membership of

a specific permission group (e.g., clients, employees, department members, account holders, subscribed users, etc.).

In addition, users act as nodes in the blockchain network: They collaboratively maintain a list of credentials in a public ledger (blockchain) and enforce a specific credential issuing policy when adding to that list. For more details on how these steps work we refer to subsection High-Level description 6.2.3.

6.2.2 Threat Model

In order to demonstrate the security of the proposed protocol, we determine the capabilities and possible actions of an attacker. We consider a PPT attacker who has perfect control of the communication channels. They can eavesdrop all messages in public channels and also modify, add, and remove messages on the network. The attacker can, at any time, corrupt $(t-1)$ of the user's agents (no more than threshold t), in which case the attacker knows all the long-term secrets (such as private keys or master shared keys).

In the proposed protocol, we consider some privacy requirements such as unlinkability, identity privacy, and user data privacy: Unlinkability means that an adversary cannot distinguish a user who is authenticating from any (other) user who has authenticated in the past. Identity privacy means that an adversary cannot determine if a given authentication credential belongs to a specific user. User data privacy means that an adversary cannot learn anything about the user's sensitive authentication data (i.e., biometric data, password).

6.2.3 High-Level View

To build a fully decentralized authentication architecture, we need to setup a small distributed shared database (to store (protected) credentials) between nodes. Data is highly available, but nobody has control over the database. Furthermore, users would never want to modify data in the past. User data needs to be immutable, and data should be publicly accessible. We employ a public append-only ledger in order to fulfill our requirements. A ledger (blockchain) maintains the integrity of the dataset and guarantees a consistent view of the data for every party. Every participant can append information to the ledger and, once uploaded, nobody can delete or modify the data. One of our goals is that users are relieved from the burden of storing all their sensitive data on their personal devices. Instead, they can rely on the protection offered by the ledger and other cryptography tools, which leads to storing the data securely.

Definition 58 (DAMFA)**.** *A DAMFA system consists of a global transaction ledger instead of a single party representing the organization. Moreover, the DAMFA scheme consists of the following phases:*

- Setup: In the setup phase, we define the public parameters and execute the following algorithm: U generates a private key and executes threshold secret sharing (TSS) on

the private key to generate shares of that secret. The user stores the secret shares among their personal agents (similar to the initialization of TOPRF [JKKX17], done via a distributed key generation for discrete-log-based systems, e.g. [GJKR99]).

- Registration: In the registration phase, the user U first selects a password pw and collects their biometric Bio at a sensor. Then, U runs the TOPRF protocol by interacting with personal agents to reconstruct the TOPRF secret key. After that, the IdP issues a membership credential that shows that U is a valid member (e.g., employee, account holder, subscribed user, etc). For this purpose, U sends a request with a pseudonym and a (non-interactive) zero-knowledge proof (NIZK) which indicates they are the owner of the pseudonym (they know the secret key that belongs to the pseudonym) and authenticate themself to the IdP. Then, U receives a membership credential which is a signature on their pseudonym.

  The user U creates a pseudonym nym_u^o and verification information, namely a protected credential PC_i, by encrypting the membership credential with some attributes and the TOPRF secret key. Subsequently, U computes a NIZK proof that (1) the credential PC_i and the pseudonym contain the same secret key and (2) proof of knowledge of the signature on attributes which is issued by the ID provider (i.e. she has valid group membership). Note that the user can execute these actions in an offline state because no interaction with the public ledger is required. Finally, nodes accept the credential to the ledger if and only if this proof is valid.

- Authentication: The user U attempts to access the services of a SP in an anonymous and unlinkable way. SP authenticates the user if and only if the user provides a valid credential. First, a service provider sends an authentication request (which is a signature) to U. The user inserts the password pw^* and the biometric Bio^* and runs the TOPRF protocol by interacting with personal agents to reconstruct the TOPRF secret value. U first scans the public ledger to obtain the accumulator AC, which is a set $\mathsf{PC} = \{\mathsf{PC}_1, \ldots, \mathsf{PC}_n\}$ consisting of all credentials belonging to a specific IdP. Then, U finds their own protected credential PC_i^* within this set (via the pseudonym nym_u^o). U decrypts PC_i^* using the TOPRF secret key and recovers the initial credential (a signature from IdP). U presents the credential under a different pseudonym nym_u^v by proving in zero-knowledge that (1) they know a credential PC_i on the ledger from IdP, (2) the credential opens to the same secret key as their own pseudonym nym_u^v, and (3) they prove possession of a membership credential from IdP (the signature), cf. [GGM14b]. SP scans the public ledger to obtain the accumulator AC which is a set $\mathsf{PC} = \{\mathsf{PC}_1, \ldots, \mathsf{PC}_n\}$ consisting of all credentials belonging to a specific organization. Then, it checks the validity of the candidate credential by finding the candidate credential in the set $\mathsf{PC}_i^* \in \mathsf{PC}$ and checking proof of knowledge on the credential and pseudonym.

6.2.4 The DAMFA Functionality

We formally define the proposed scheme's security by presenting its ideal functionality that is implemented via a trusted party F_{TOPRF} with a public ledger. All communication take place through this ideal trusted party. In the UC framework [CHK+05, CDT19], there may be some copies of the ideal functionality running in parallel. Each one is supposed to have a unique session identifier (SID). Each time a message is sent to a specific copy of functionality, such that this message contains the SID of the copy that is intended for. As noted in [JKKX17], we also use the ticketing mechanism, which ensures that in order to test a password and biometric guess, the attacker must impersonate $t+1$ agents. To this end, they define a counter $tx(p, \mathsf{PA_i})$ for each $\mathsf{PA_i} \in \mathsf{SI}$ in which the parameter p is also used to identify it. In addition, when an agent $\mathsf{PA_i} \in \mathsf{SI}$ completes its interaction, the functionality increases the counter $tx(p, \mathsf{PA_i})$. On the other hand, when a user, either honest or corrupt, completes an interaction that is associated to $\mathsf{PA_i}$, $tx(p, \mathsf{PA_i})$ decreases by 1. It ensures that for any honest agent $\mathsf{PA_i}$, the number of user-completed OPRF evaluations with $\mathsf{PA_i}$ is no more than the number of agent-completed OPRF evaluations of $\mathsf{PA_i}$. It sets $t+1$ agent tickets for accessing the proper TOPRF result by reducing (non-zero) ticket counters $tx(p, \mathsf{PA_i})$ for an arbitrary set of $t+1$ agents in SI. The ideal functionality as:

Registration

- Upon receiving $(\mathsf{Reg}, \mathsf{sid}, \mathsf{SI}, \mathsf{pw}, \mathsf{Bio})$ for $|\mathsf{SI}| = \mathsf{PA_i}_n$ from U, records this message and sends $(\mathsf{Reg}, \mathsf{U}, \mathsf{sid}, \mathsf{SI})$ to $\mathcal{A}^*$ (Ignores other Reg cmd). Computes a secret key K using TOPRF protocol F_{TOPRF} and if $|\mathsf{SI} \cap \mathsf{CorrSrv}| \geq t+1$ then sends $(K, \mathsf{pw}, \mathsf{Bio})$ to $\mathcal{A}^*$.

- Upon receiving $(\mathsf{SReg}, \mathsf{sid}, \mathsf{PA_i})$ from $\mathcal{A}^*$, if a record $\langle \mathsf{Reg}, \mathsf{U}, \mathsf{sid}, \mathsf{SI}, \mathsf{pw}, \mathsf{Bio} \rangle$ exists and $\mathsf{PA_i} \in \mathsf{SI}$ then marks $\mathsf{PA_i}$ as active and sends $(\mathsf{SInit}, \mathsf{sid})$ to $\mathsf{PA_i}$.

- Upon receiving $(\mathsf{UReg}, \mathsf{sid})$ from $\mathcal{A}^*$, if the record $\langle \mathsf{Reg}, \mathsf{U}, \mathsf{sid}, \mathsf{SI}, \mathsf{pw}, \mathsf{Bio} \rangle$ exists and all agents in SI are marked active, then runs a commitment scheme F_{Com} and an encryption F_{Enc} to get (τ_i, γ_i) respectively, and sets the pseudonym as $\mathsf{nym}_u^o = \tau_i$ and $\mathsf{PC}_i = \gamma_i$ as the credential. It records $\langle \mathsf{nym}_u^o, \mathsf{PC}_i, \mathsf{U}, \mathsf{SI}, K \rangle$, sends $(\mathsf{sid}, \mathsf{nym}_u^o, \mathsf{PC}_i)$ and $(\mathsf{RegComplete}, \mathsf{sid}, \mathsf{SI})$ to its public ledger and $\mathcal{A}^*$ respectively.

Authentication

- Upon receiving $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{SR}, \mathsf{pw}', \mathsf{Bio}')$ for $|\mathsf{SR}| = t+1$ from U^*, retrieves $\langle \mathsf{Reg}, \mathsf{U}, \mathsf{sid}, \mathsf{SI}, \mathsf{pw}, \mathsf{Bio}, K \rangle$, records $\langle \mathsf{Auth}, \mathsf{U}^*, \mathsf{sid}, \mathsf{SI}, \mathsf{SR}, \mathsf{pw}, \mathsf{pw}', \mathsf{Bio}, \mathsf{Bio}' \rangle$ and sends $(\mathsf{Auth}, \mathsf{U}^*, \mathsf{sid}, \mathsf{ssid}, \mathsf{SR})$ to $\mathcal{A}^*$. Ignores future Auth commands involving the same ssid.

- Upon receiving $(\mathsf{SAuth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{PA_i})$ from $\mathcal{A}^*$, if $\mathsf{PA_i} \in \mathsf{SR}$ is marked active then sets $tx(\mathsf{PA_i})$++ (sets it to 1 if it is undefined) and sends $(\mathsf{SAuth}, \mathsf{sid}, \mathsf{ssid})$ to $\mathsf{PA_i}$.

Password and Biometric Test

- After receiving $(\mathsf{TestPwBio}, \mathsf{sid}, \mathsf{PA_i}, \mathsf{pw}^*, \mathsf{Bio}^*)$ from $\mathcal{A}^*$, if $tx(\mathsf{PA_i}) > 0$ then sets $\mathsf{tested}(\mathsf{pw}) = \mathsf{tested}(\mathsf{pw}^*)$ and $(\mathsf{Bio}) = \mathsf{tested}(\mathsf{Bio}^*) \cup \mathsf{PA_i}$ and $tx(\mathsf{PA_i}) := tx(\mathsf{PA_i}) - 1$, retrieves $\langle \mathsf{Reg}, \mathsf{U}, \mathsf{SI}, \mathsf{pw}, \mathsf{Bio}, K \rangle$ and if $|\mathsf{SI} \cap (\mathsf{tested}(\mathsf{pw}^*) \wedge \mathsf{tested}(\mathsf{Bio}^*) \cup \mathsf{CorrSrv})| \geq t+1$ and if $\mathsf{pw}^* = \mathsf{pw}$ and $\mathsf{Bio} = \mathsf{Bio}^*$, then returns sk to $\mathcal{A}^*$ and marks the record compromised and responses to $\mathcal{A}^*$ with "correct guess", else returns FAIL.

Authentication for Service Provider

- $\mathsf{GetCredList}()$: Every participant can obtain all data in the public ledger of the trusted party via submitting a "retrieve" request to F_{DAMFA}. SP then retrieves the intended credential PC_i issued by nym_u^o from F_{TOPRF} and accepts functionality's assertion only if $\mathsf{PC}_i \subset \mathsf{PC}$.

- Key generation: Upon receiving $(\mathsf{UAuth}, \mathsf{sid}, \mathsf{ssid}, P_i, \mathsf{SR}, sk)$ for $|\mathsf{SR}| = t + 1$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{Auth}, P, \mathsf{sid}, \mathsf{ssid}, \mathsf{SI}, \mathsf{SR}, \mathsf{pw}, \mathsf{Bio}, \mathsf{pw}', \mathsf{Bio}' \rangle$, where $P \in \{\mathsf{U}, \mathsf{SP}\}$ then do:
  - If this record is compromised so that $\mathsf{pw}^* = \mathsf{pw}$ and $\mathsf{Bio}^* = \mathsf{Bio}$ or $(\mathsf{SR} \subseteq \mathsf{CorrSrv})$, then output (sid, sk) to player P_i.
  - Else, if this record is fresh, and if there is a record $(P, \mathsf{pw}', \mathsf{Bio}', sk')$ with $\mathsf{pw}' = \mathsf{pw}$ and $\mathsf{Bio}' = \mathsf{Bio}$, then sends sk' (a random key) to player P_i.
  - In any other case, picks a random key sk and sends (sid, sk) to P_i.

Definition 59 (Secure DAMFA)**.** *Let* Π *be a probabilistic polynomial time protocol for the* DAMFA *functionality. We say that* Π *is secure if for every PPT real world adversary* $\mathcal{A}$ *attacking* DAMFA*, there exists a PPT ideal world simulator* $\mathcal{S}$ *such that for both the real and ideal world interactions, outputs of registration and authentication phases are computationally indistinguishable:* $\mathsf{Real}_{\mathcal{A}}(1^\lambda) \approx \mathsf{Ideal}_{\mathcal{S}}(1^\lambda)$.

6.2.5 Our Construction

We build our protocol using existing building blocks 6.1 and basic primitives in the Preliminaries. In fact, it is a generic structure that can be realized using an anonymous credential. Here we have tried to choose and combine them wisely, leading to efficient construction.

Setup Phases

We select a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that is efficiently computable, non-degenerate, and three groups with prime order p. We let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $g_t = e(g_1, g_2)$ the generator of $\mathbb{G}_T$. Note that it is assumed to support one-way Bio-hash function $H_1()$ which resolves the recognition error of general

hash functions [JLG04]. We consider two additional hash functions as $H_2 : M \rightarrow \{0,1\}^{\lambda}$ and $H_g : M \rightarrow \mathbb{G}_1$. We publish $params \leftarrow (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, p, h_{nym}, H_1, H_2, H_g)$ as the set of system parameters where $h_{nym} \in \mathbb{G}_1$. The user U generates a private key K, then executes a secret sharing construction scheme on K to create secret keys for each personal agent $\langle k_1, k_2, \ldots, k_n \rangle \leftarrow \mathsf{TSS}(K)$. U stores secret shares among personal agents.

Registration Phase

To register a user to the system, U first chooses a password pw and scans her biometric impression Bio at the sensor. Then, U runs the following steps to register herself in the system.

- A user runs TOPRF protocol [JKKX17] with agents to compute the secret value $usk = F_K(\mathsf{pw}, \mathsf{Bio})$ as follows:
  - The user U picks a random number $r \in Z_p$ and computes $A = H_g(\mathsf{pw}, H(\mathsf{Bio}))^r$ and sends the message $M_1 = \{A\}$ to all PA_i.
  - Upon receiving the message $M_1 = \{A\}$ from the user, each PA_i computes $b_i = A^{k_i} = H_g(\mathsf{pw}, H_1(\mathsf{Bio}^*))^{\lambda_i \cdot k_i \cdot r}$ by Lagrange interpolation coefficients and secret key k_i (s.t. $K = \sum_{i \in \mathsf{SR}} \lambda_i \cdot k_i$). They return the message $M_2 = \{b_i\}$ to U.
  - After receiving all the messages b_i from personal agents, U computes: $C = \prod_{i \in \mathsf{SR}} b_i^{r^{-1}} = H_g(\mathsf{pw}, H_1(\mathsf{Bio}))^K \rightarrow usk = h(\mathsf{pw}, C)$.
- In order to obtain a membership credential from IdP, we use PS signatures protocol [PS16b] to derive a signature on a hidden committed message as follows:
  - $\mathsf{KeyGen}(pp)$: The IdP runs this algorithm to generate private and public keys. This algorithm selects $(x, y, y_1) \leftarrow Z_p$, computes $(X, Y, Y_1) \rightarrow (g_1^x, g_1^y, g_1^{y_1})$ and $(X', Y', Y_1') \rightarrow (g_2^x, g_2^y, g_2^{y_1})$, and sets $sk \rightarrow (X, y, y_1)$ and $pk \rightarrow (g_1, g_2, Y, X', Y')$.
  - $\mathsf{Protocol}$. A user first selects a random $r_2 \leftarrow Z_p$ and computes $C = g_1^{r_2} \cdot Y^{usk}$, which is a commitment on her secret key. She then sends C to the IdP. They both run a proof of knowledge of the opening of the commitment (authentication). If the signer is convinced, the IdP selects a random $u \leftarrow Z_p$ and returns $\sigma \leftarrow (\sigma_1 = g_1^u, \sigma_2 = (X \cdot C \cdot Y_1^m)^u)$. The user can now unblind the signature σ and get a valid signature over her secret key and the message m_1 by computing $\sigma \leftarrow (\sigma_1, \sigma_2 / (\sigma_1)^{r_2})$ described in [PS16a].
  - Verify. To verify this signature, the user can execute this algorithm and compute: $\mathsf{Verify}(pk, m, \sigma)$: $e(\sigma_1, X' \cdot Y'^{usk} \cdot Y_1'^{m_1}) = e(\sigma_2, g_2)$.
- $\mathsf{CreatePC}$. The user generates a protected credential with TOPRF secret key usk derived from the password and the biometric: U picks a random number $s \in Z_p$ to generate a pseudonym as $\mathsf{nym}_u^o = g_1^s \cdot h_{nym}^{usk}$ and computes an El-Gamal encryption of the credential σ with secret TOPRF values usk into a ciphertext as: $\mathsf{PC}_i = [\sigma]_{usk}$.

- Proof. A NIZK proof of knowledge of the credential (PS signature [PS16b]) works as follows: U selects random $r_3, t_1 \leftarrow Z_p$ and computes $\sigma' \leftarrow (\sigma_1^{r_3}, (\sigma_2 \cdot \sigma_1^{t_1})^{r_3})$. U sends $\sigma' = (\sigma'_1, \sigma'_2)$ to the verifier and carries out a zero-knowledge proof of knowledge (such as the Schnorr's interactive protocol) of m, usk and t_1 such that:

$$\pi = \mathsf{ZKPoK} \left\{ \begin{array}{l} (s, m_1, t_1, usk) : \mathsf{nym}_u^o = g_1^s \cdot h_{nym}^{usk} \ \wedge \mathsf{PC}_i = Enc_{usk}(\sigma) \\ \wedge\, e(\sigma'_1, Y)^{usk} \cdot e(\sigma'_1, g_2)^{t_1} \cdot (\sigma'_1, Y_1)^{m_1} = \dfrac{e(\sigma'_2, g_2)}{e(\sigma'_1, X)} \end{array} \right\}.$$

- At the end of this phase, U submits the resulting values $(\mathsf{PC}_i, \pi, \mathsf{nym}_u^o)$ to the public ledger nodes where π is a proof of knowledge on the nym_u^o and the PC_i. If the signature verifies successfully, output 1, otherwise 0. The nodes should accept values to the ledger if this algorithm returns 1.

Authentication Phase

In this phase, a user authenticates herself to the service provider and establishes a session key with the service provider. The following steps are executed by U, $\mathsf{PA_i}$, and SP:

- First of all, the server chooses a secret key $y \leftarrow Z_p$ and computes $Z \leftarrow g_1^y$. Then, SP generates a signature σ_s on message Z (i.e. Schnorr's signature [Sch90]) using its secret key and sends the message $M_1 = \{Z, \sigma_s\}$ to the user.

- When receiving a pair (Z, σ_s), the client verifies whether σ_s is valid on message Z under the SP's public key. If σ_s is valid, U inserts pw^* and scans her personal biometric impression Bio^* at the sensor.

- The user interacts with personal agents and runs the necessary steps to compute the TOPRF protocol $F_K(\mathsf{Bio}^*, \mathsf{pw}^*) = usk = h(\mathsf{pw}^*, \prod_{i \in \mathsf{SR}} b_i^{r^{-1}})$. Then, U decrypts ciphertext $[\sigma]_{usk}$ with the TOPRF secret key usk to recover the credential σ.

- Show. The user generates a NIZK π to demonstrate that the credential is well-formed and corresponds to her pseudonym's secret values. This proof establishes three key points: (1) she possesses a credential on the ledger from the IdP, (2) the credential contains the secret key linked to her pseudonym, and (3) she has possession of a valid credential (signature). To achieve this, we utilize the bilinear maps accumulator [CKS09] to accumulate the group elements $g_1, \ldots, g_n$ instead of using integers $1, \ldots, n$. Additionally, we employ Camenisch et al.'s efficient zero-knowledge proof of knowledge, such as Schnorr's protocol [Sch90, FS86], to verify that a committed value is within the accumulator. Further details on how this proof functions can be found in [CKS09, CMZ14b].

  U runs the following steps to authenticate herself:

  – The user selects a random number $r_4 \in Z_p$ to generate a pseudonym $\mathsf{nym}_u^v = g_1^{r_4} \cdot h_{nym}^{usk}$ for communication with service providers.

– U picks random numbers $d, t_2 \leftarrow Z_p$ and computes a randomized commitment credential (like in the previous step) as $\sigma' \leftarrow (\sigma_1^{r_2}, (\sigma_2 \cdot \sigma_1^{t_2})^{r_2})$.

– Then, U calculates $D = g_1^d$, a secret session key $\mathsf{SK} = Z^d = g_1^{y \cdot d}$ and $\mathsf{Hmac}(\mathsf{SK}, D, Z)$.

– For a set of credentials PC, U computes an accumulator and witness as $\mathsf{AC} = \mathsf{Accumulate}(\mathsf{pp}, \mathsf{PC})$ and $\omega = \mathsf{GenWitness}(\mathsf{pp}, \mathsf{PC}, \mathsf{PC}_i^*)$, carries out a zero-knowledge proof of knowledge of the credential, and outputs the following proof of knowledge π such that:

$$\mathsf{ZKPoK}\left\{\begin{array}{l}(usk, \omega, d, t_2, m, r_4) : \mathsf{AccVerify}(\mathsf{pp}, \mathsf{AC}, \omega) = 1 \wedge \\ e(\sigma_1', Y)^{usk} \cdot e(\sigma_1', g_2)^{t_2} \cdot (\sigma_1', Y)^m = \dfrac{e(\sigma_2', g_2)}{e(\sigma_1', X)} \wedge \\ \mathsf{PC}_i = Enc_{usk}(\sigma) \wedge D = g_1^d \ \wedge \mathsf{nym}_u^v = g_1^{r_4} \cdot h_{nym}^{usk}\end{array}\right\}.$$

Finally, U sends the message $M_4 = \{\mathsf{nym}_u^v, D, \mathsf{Hmac}, \pi\}$ to the service provider.

- After receiving the message $M_4 = \{\mathsf{nym}_u^v, D, \mathsf{Hmac}, \pi\}$ from the user, the service provider first scans through the ledger to obtain a set PC consisting of all credentials belonging to IdP. First, SP computes the accumulator $AC = \mathsf{Accumulate}(\mathsf{pp}, \mathsf{PC})$. Then, it verifies that $\pi = 1$ is the aforementioned proof of knowledge on PC_i and nym_u^v using the known public values. If the proof verifies successfully, output 1, SP computes the session key as follows: $\mathsf{SK} = D^y = g_1^{y \cdot d}$.

Then, SP computes $\mathsf{Hmac}^*(\mathsf{SK}, D, Z)$ and checks $\mathsf{Hmac} = \mathsf{Hmac}^*$. If $\pi = 1$ and Hmac holds, SP accepts SK as the session key and also the user is authentic.

Note that we can simply send σ' alongside the message of the proof of knowledge. With this, we can prove the construction is a Σ-protocol (see [PS16b] to see how proof of knowledge of PS signature works). Moreover, users do not need to store sensitive data in their devices (i.e., data can be protected using the key extracted using a password and biometrics and stored in a ledger for simple accessibility).

Theorem 6.2.1. *Our proposed protocol is secure against any non-uniform PPT adversary corrupting $t - 1$ many personal agents $\mathsf{PA_i}$ by assuming that the El-Gamal encryption, zero knowledge proof of signature, and the TOPRF protocol are secure and also the hash function is collision resistant.*

Proof Sketch. Our construction DAMFA is modular and relies directly on the TOPRF and the zero-knowledge proof. The security is then straightforwardly inherited from those algorithms:

The credential security requires that no adversary is able to present a credential (guess passwords and biometrics) and generate a session key, which they have not had any access to. If we use a TOPRF on passwords and biometric of users, then the security properties of TOPRF would make it hard to guess. The proof is once again two-fold:

- First, the authentication is done through a zero-knowledge proof. At this step, either the adversary presents an invalid credential or manages to build a valid proof. Hence, the adversary breaks the soundness of the underlying proof of knowledge we used, or otherwise, uses a valid credential.
- At this step, we now assume the adversary wins by using a valid credential. We now rely on the obliviousness of the TOPRF. We interact with a TOPRF challenge to answer every adversarial request, and at the end, we can use the (valid) credential output by the adversary to break the TOPRF obliviousness, which leads to the conclusion.

Anonymity. During the registration phase, when a user reveals her pseudonym but does not (intentionally) reveal her secret key usk, no adversary should learn any information about the secret key or the identity. Besides, during the authentication phase, a user proves her credential using zero-knowledge proof, which reveals no additional information about her secret key and identity to the SP.

Proof is as follows. The simulator $\mathcal{S}$ is essentially an ideal-world adversary that interacts with the functionality F_{DAMFA} and the environment ξ. We also assume that our zero-knowledge proof of signature includes an efficient extractor and a simulator and also the signature is unforgeable. In order to ensure that the environment's perspective in the ideal-world is indistinguishable from its perspective in the real-world, the simulator must interact with the real-world adversary $\mathcal{A}$ by emulating all other entities for $\mathcal{A}'$. The simulator closely follows the actions of adversary $\mathcal{A}'$ in most aspects.

Description of the simulator. Once the adversary registers a new user to the system via storing a tuple $(\mathsf{nym}_u^o, \mathsf{PC}_i, \pi_i)$ to the bulletin board, the simulator registers this user in the ideal world via the following process. It makes an interface between honest parties in the real world (which is U and $n-t+1$ personal agents $\mathsf{PA_i}$ where $i = t, \ldots, n$ wlog. since all personal agents in our scheme are same) and corrupted parties in the ideal-world (which are SP and t personal agents $\mathsf{PA_{ic}}$ where $\mathsf{ic} = (1, \ldots, t)$. The simulator behaves as:

Registration

- Upon receiving $(\mathsf{Reg}, \mathsf{sid}, \mathsf{U}, \mathsf{SI})$ from F_{DAMFA}, ignores it if $|\mathsf{SI}| \neq \mathsf{PA}_n$. Otherwise, records $\langle \mathsf{U}, \mathsf{sid}, \mathsf{SI} \rangle$ and sends $(\mathsf{Send}, (\mathsf{sid}, 0), \mathsf{U}, \mathsf{PA_i}, \mathsf{SI})$ to $\mathcal{A}'$ for all $\mathsf{PA_i} \in \mathsf{SI}$. If F_{TOPRF} sends $(K, \mathsf{pw}, \mathsf{Bio})$, records it.

  Remark 4. *$\mathcal{S}$ simulates $\mathsf{PA_{ic}}$ in the ideal world, and receives whatever they receive from F_{DAMFA}.*

- Upon receiving $(\mathsf{sid}, \mathsf{PA_{ic}}, \mathsf{PC}_i, \mathsf{nym}_u^o, \pi_i)$ from $\mathcal{A}'$ for some $\mathsf{PA_i} \in \mathsf{SI}$, $\mathcal{S}$ checks its records to see if it has information about $(\mathsf{U}, k_{\mathsf{ic}}, \mathsf{nym}_u^o)$ in its list of users. If such a user with nym_u^o exists, $\mathcal{S}$ retrieves the associated key K for $(\mathsf{U}, k_{\mathsf{ic}}, \mathsf{nym}_u^o)$ and

proceeds accordingly. The simulator then employs the knowledge extractor to obtain usk. If it is not on the list, $\mathcal{S}$ follows the protocol to register nym_u^o as a user by choosing a random password pw^* and Bio^*. It generates secret shares k'_{ic} on K for each corrupted personal agent, records $\langle \mathsf{Reg}, \mathsf{U}, \mathsf{sid}, \mathsf{SI}, \mathsf{pw}^*, \mathsf{Bio}^*, k_{\mathsf{ic}}, K \rangle$ and sends $\langle k_{\mathsf{ic}} \rangle$ to $\mathsf{PA}_{\mathsf{ic}} \in \mathsf{SI}$ and $\mathcal{A}'$.

- Upon receiving $(\mathsf{RegComplete}, \mathsf{sid}, \mathsf{SI})$ from $\mathcal{A}'$, retrieves $\langle \mathsf{Reg}, \mathsf{U}, \mathsf{sid}, \mathsf{SI}, \mathsf{pw}^*, \mathsf{Bio}^*, k_{\mathsf{ic}}, K \rangle$, computes a pseudonym nym_u^v and a credential $\mathsf{PC}'_i = h \cdot g^{usk}$ where $usk_{\mathsf{ic}} = F_K(\mathsf{pw}^*, \mathsf{Bio}^*)$. It records $\langle \mathsf{nym}_u^v, \mathsf{PC}'_i, \mathsf{U}, \mathsf{SI}, usk_{\mathsf{ic}} \rangle$ and sends $(\mathsf{sid},\ \mathsf{PC}'_i,\ \mathsf{nym}_u^v,\ \pi_i)$ to its public ledger and $\mathcal{A}'$ where π_i is proof of knowledge. $\mathcal{S}$ stores $(\mathsf{pw}^*, \mathsf{Bio}^*, K, usk_{\mathsf{ic}}, \mathsf{nym}_u^v, \mathsf{PC}'_i, \pi_i)$ in its list of credentials.

  Remark 5. *When an honest user initiates the process of obtaining a credential through the functionality, the simulator generates a credential and employs the signature of the knowledge extractor to simulate the corresponding proof. Afterward, it forwards the credential* $(\mathsf{PC}'_i, \pi_i, \mathsf{nym}_u^v)$ *to the ledger.*

Authentication

- Upon receiving $(\mathsf{Auth}, \mathsf{U}^*, \mathsf{sid}, \mathsf{ssid}, \mathsf{SR})$ where $|\mathsf{SR}| \geq t+1$ from $\mathcal{A}'$, retrieves $\langle \mathsf{nym}_u^v, \mathsf{PC}'_i, \mathsf{U}, \mathsf{SI}, usk_{\mathsf{ic}} \rangle$ corresponding to U as stored in the registration phase. If there is a set $(\mathsf{Bio}, \mathsf{pw}, K)$ stored in the registration phase and usk_{ic} is defined, then executes the TOPRF protocol with each personal agent using the password pw^* and Bio^* and receives $\rho_{\mathsf{ic}} = T(p, (\mathsf{pw}^*, \mathsf{Bio}^*))$ from F_{TOPRF} and sends $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{U}, \mathsf{SR})$ to $\mathcal{A}'$.

- Upon receiving $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{U}, \rho_{\mathsf{ic}})$ from F_{TOPRF}, $\mathcal{S}$ recovers SR and usk_{ic} corresponding to U as stored during the registration phase in the database (ignores this message if no corresponding tuples exist). $\mathcal{S}$ checks $\rho_{\mathsf{ic}} = usk_{\mathsf{ic}}$ and if each $\mathsf{PA}_{\mathsf{ic}}$ used the correct corresponding $\mathsf{share}_{\mathsf{ic}} = (usk_{\mathsf{ic}}\ ,\ k_{\mathsf{ic}})$ values. Ignores this message if either of the following conditions fails: if $\rho_{\mathsf{ic}} = usk_{\mathsf{ic}}$ then $|S|tx(p, S) > 0| > t$ or all servers in SR are honest. Otherwise, sends $(\mathsf{Auth}, \mathsf{sid}, \mathsf{SR}, \mathsf{pw}^*, \mathsf{Bio}^*, sk)$ to F_{DAMFA} where sk is a random secret key and sets for $(flag, \mathsf{pw}^*, \mathsf{Bio}^*, sk)$ as follows:
  - **Case 1**: Shares $\mathsf{share}_{\mathsf{ic}} = (\rho_{\mathsf{ic}}, k_{\mathsf{ic}})$ are employed by the adversary in the real protocol. $\mathcal{S}$ detects this by verifying that $usk_{\mathsf{ic}} = \rho_{\mathsf{ic}}$. Therefore, $\mathcal{S}$ sets $(flag, \mathsf{pw}^*, \mathsf{Bio}^*, sk) = (1, ., .)$ and sends $(usk_{\mathsf{ic}}, k_{\mathsf{ic}})$ in its database to F_{DAMFA} where $usk_{\mathsf{ic}}, k_{\mathsf{ic}}$ was sent by F_{DAMFA}.
  - **Case 2**: Otherwise, incorrect $usk_{\mathsf{ic}}, k_{\mathsf{ic}}$ employed by the adversary in the real protocol. $\mathcal{S}$ detects this by verifying that $usk_{\mathsf{ic}} \neq \rho_{\mathsf{ic}}$. So, $\mathcal{S}$ sets $(flag, \mathsf{pw}^*, \mathsf{Bio}^*, sk) = (0, ., .)$ and defines x as the set of values pw and Bio in the dictionary such that $T(p^*, (\mathsf{pw}, \mathsf{Bio}))$ is defined. For every x in lexicographic order, sets $v := T(p^*, x)$ and checks if $v = usk_{\mathsf{ic}}$. If so, sets $(flag, \mathsf{pw}^*, \mathsf{Bio}^*, sk) := (2, x, sk^*)$ and breaks the loop. If the above loop processes all pw and Bio without breaking, sets $(flag, \mathsf{pw}^*, \mathsf{Bio}^*, sk) = (0, ., .)$.

- On receiving $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{SR}, x = \{pw^*, \mathsf{Bio}^*\})$ from party $P \in (U, \mathcal{A}')$ and $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, P, \rho_{\mathsf{ic}})$ from $\mathcal{A}'$, recovers usk_{ic} corresponding to U as stored in step 1. It ignores this message if either of the following conditions fails: If $\rho_{\mathsf{ic}} = usk_{\mathsf{ic}}$ then $|S|tx(p, S) > 0| > t$ or if all servers in SR are honest. Otherwise, picks $T(p^*, x) \leftarrow \{0,1\}^l$ if it has not been defined and sends $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, T(p^*, x))$ to $\mathcal{A}'$. If $\rho_{\mathsf{ic}} = usk_{\mathsf{ic}}$ (without resulting in the failure of conditions) then adds every $\mathsf{PA_i} \in \mathsf{SR}$ to $\mathsf{tested}(x)$ and sends $(\mathsf{TestPwBio}, \mathsf{sid}, \mathsf{PA_i}, \mathsf{pw}^*, \mathsf{Bio}^*)$ to F_{DAMFA}. If F_{DAMFA} replies sk, then records it.

  Remark 6. *In the ideal world, F_{DAMFA} utilizes the ideal user-provided password and biometric test. If the adversarial personal agents in the real world behave honestly, it implies that the simulator has provided correct pairs (usk_i, k_i). As a result, the calculated credentials and pseudonyms will be valid and stored in the ledger, as they are computed using the genuine password and biometric information. However, if the personal agents act maliciously in the real world, the simulator ($\mathcal{S}$) would have detected this during the previous step and provided incorrect pairs to F_{DAMFA} in the ideal world. Consequently, the responses in both worlds would be invalid.*

- Upon receiving $(\mathsf{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{SR}, \mathsf{nym}_u^v, \mathsf{PC}_i)$ from F_{DAMFA}, $\mathcal{S}$ forwards $\langle \mathsf{nym}_u^v, \mathsf{PC}_i \rangle$ to the $\mathcal{A}'$ in the real world.

The Indistinguishability

- $\mathsf{Game}_{\mathsf{Real}}$. This is the real world, the system constructed in this work is run between $n - t + 1$ honest parties and t parties controlled by the adversary.

- $\mathsf{Game}_{\mathbf{1}}$. This is identical to $\mathsf{Game}_{\mathsf{Real}}$ except that the encryption generated in the registration phase by honest users is replaced with a simulated one. Indistinguishability between $\mathsf{Game}_{\mathsf{Real}}$ and Game_1 comes from the El-Gamal encryption security properties.

- $\mathsf{Game}_{\mathbf{2}}$. This is identical to Game_1 except that in TOPRF, each share (b_i and usk) generated by honest users using an actual password pw and biometric Bio is replaced by pw^* and Bio^* chosen randomly. Since, $\mathcal{S}$ does not have the correct password and biometric. Indistinguishability between Game_1 and Game_2 comes from the indistinguishability of the TOPRF algorithm and TSS construction.
  - Reduction 1: The security of TOPRF ensures that an adversarial personal agent, acting as a sender, cannot differentiate between the receiver's (simulated user's) genuine input, consisting of a password pw and biometric data Bio, and any other randomly chosen pair of password pw^* and biometric Bio^*. This property guarantees user privacy and data confidentiality.
  - Reduction 2: The security of TSS guarantees that if fewer than the threshold number of agents are compromised, they cannot reconstruct the secret or verify if the shares are associated with the same secret. Consequently, the

Table 6.1: Comparison of Public Ledger Instantiations

Properties	**Namecoin**	**Ethereum (Rinkeby)**
Initial Data Size	$\approx$ 5.08 GB	$\approx$ 5.3 GB
Initial Sync Time	$\approx$ 3h	$\approx$ 3h
Cost	0.069 USD	0.0225 USD
Confirmation Time	10 min / 2 h	a few seconds / 3 min

adversary cannot efficiently distinguish this behavior from a real scenario without compromising one more agent, making offline attacks less feasible.

- Game_3. This game is identical to Game_2 except that an authentication response (nym_o^v and PC_i^*) which are two random group elements generated by the adversary will be rejected if the extracted secret key does not fulfill the requirements. Indistinguishability between Game_2 and Game_3 comes from the verified consistency of the bilinear pairing algorithm and the simulation breaks the soundness of the underlying proof of knowledge we used before (assuming that there is no hash collision).

- Game_4. This is the world simulated by $\mathcal{S}$. It is not hard to check that $\mathsf{Game}_{\text{ideal}}$ is identical to Game_4.

We already known that the possibility of TOPRF and ZKPoK proofs to break is negligible.

6.3 Implementation

In this section, we illustrate the practicability of the proposed protocol. To this end, we provide the public ledger part which is realized by well-known blockchains, namely Namecoin and Ethereum. The results are summarized in Table 6.1. Here, *Initial data size* shows the size of the blockchain needed for downloading and storage. *Initial sync time* is the time required to sync and connect to the blockchain. *Confirmation time* is the time required to confirm that the data are uploaded in the blockchain.

6.3.1 Namecoin implemention

The public ledger can be implemented by a blockchain system. One way to realize a public ledger is to use the Namecoin blockchain. Namecoin allows registering names and stores related values in the blockchain which is a securely distributed shared database. It also enables a basic feature to query the database and to retrieve the list of existing names and associated data. Thus, we can store credentials, scan them based on namespace and then verify them. We execute the following steps in order to participate in the Namecoin system and store credentials by the namecoin *id* as pseudonyms:

- We need to install a Namecoin client that has a full copy of the Namecoin blockchain and keep it in sync with the P2P network by fetching and validating new blocks from connected peers. We use implementation of the Namecoin client [tea16], which can be controlled by HTTP JSON-RPC, command line, or graphical interface. It spontaneously connects to the Namecoin network and downloads the blockchain.

- The Namecoin client also creates the user's wallet which includes the private key of Namecoin address of the user.

- To save credentials in the blockchain, the user needs to register a namespace "`id/name`" as the owner of the name by paying a very small fee (0.0064 USD as of March 2018). An *id* name can be registered using the Namecoin graphical interface or commands "name_new" and "name_firstupdate". The following description shows how the *id* name in Namecoin namespace is registered and how those names can be accessed.

  ```
  namecoind name-new id/3608a30756b0...
  ```

  The output will look like this:

  ```
  [ "0e0e03510b0b0b7dbba6e301e519693f6
  8062121b29f3cd3a6652c238360d0d0",
  "9f213ff4a582fd65" ]
  ```

  This transaction shows a hashed version of the name, salted with a random value (which is "9f213..." for transaction ID "0e0e0351...").

- The user can store arbitrary data as descriptions (which contains a credential) for Namecoin keys using JSON format: the following codes can be a simple example of the JSON value of an identity name:

  ```
  namecoind name_firstupdate id/3608...
  ```

  Output:

  ```
  { "description" : "28790de641755e77d1
  3382229156f5c26a9dd8a9673006b...",
  "namecoin" : "NBvmSUQbRGu..." }
  ```

- Subsequently, the update has been confirmed and transactions have been added to the blockchain. The user has a fully valid credential. To show the credential, SP scans through the list of added names and retrieves all credentials via a graphical interface or commands like the following code:

  ```
  namecoind name_list
  ```

  Output:

  ```
  [ { "name" : "id/3608a30756b07e...",
  "value" : "28790de641755e77d13382
  229156f5c26a9dd8a9673006b15...",
  ```

```
"address" : "NBvmSUQbRGunCS...",
"expires_in" : 36000 } ]
```

Cost. Initially, a reasonable transaction fee of either 0.00 or 0.01 NMC is charged. We can choose this fee based on how fast we want to process a transaction.

Latency. Namecoin and Bitcoin both attempt to generate blocks every 10 minutes; on average, it takes nearly 5 minutes to see the data appear on the blockchain. In practice, it then takes the necessary time to solidify the transactions and the data to be verified. For Namecoin, it takes about 2 hours to confirm that the data are uploaded in the blockchain (12 confirmations). That is why `name_firstupdate` will only be accepted after a mandatory waiting period of 12 additional blocks.

Remark 7. *Note that these costs and delays occur only once during the setup and registration phases. They do not affect the authentication phase. Thus, we focus on the computation time of the authentication phase that is frequently used in the authentication system (see Sections 6.3.3).*

6.3.2 Ethereum

Ethereum allows us to test our decentralized application on a local blockchain; we use a test network called Rinkeby to build our decentralized application. We can connect to the Ethereum blockchain and even perform operations such as mine blocks, send transactions and deploy smart contracts by running an Ethereum node.

- We run the Ethereum wallet (minst or geth command line) in order to access to Ethereum protocol and deploy our smart contract.
- To start, we need to sink the Rinkeby network locally and download blockchain which takes a few hours.
- Create an account:

  ```
  Enter a password for your Rinkeby
  account by geth command line or
  Ethereum graphic (Minst).

  Geth Version: 1.8.1-stable
  creates an account using geth
  command: geth account new
  ```

- Next, obtain some Ether so that transactions can be sent. Since we used the Rinkeby testnet, their Ether can be obtained for free at the faucet website. Ether is used to pay transaction fees.

- We can deploy smart contracts to store our credentials and names into them. For this purpose, we write our first smart contract in Solidity (Solidity is a high-level contract language that is planned to target the Ethereum Virtual Machine (EVM)) and deploy it through Mist. A simple example code is:

```
pragma solidity 0.4.2

contract Test {
string public $NYM$;
string public $Z$;

function Test(string $-NYM$,string $-Z$)
{
v1 = $-NYM$;
v2 = $-Z$;
}
}
```

- We can also see the option to watch previously deployed contracts and tokens. We can click on "Watch Contracts" bottom and enter the contract's name and contract address.

Cost. All transactions need some amount of gas to motivate processing. A transaction fee is between 0 to 0.000424 ETHER (as of March 2018) depending on how fast we want to approve the blockchain transaction.

Latency. Ethereum creates a new block every few seconds so that the data will appear on the blockchain instantly. As mentioned in Ethereum Blog, 10 confirmations is sufficient to achieve a similar security degree as that of 6 confirmations in Bitcoin. It takes around 3 minutes to confirm the transaction/data. Note that these costs and delays occur only once during the setup and registration phases.

6.3.3 Performance of the Authentication System

We now examine the performance of our anonymous authentication system. There are two main steps: the registration phase and the authentication phase. However, since time-critical operations in both registration and authentication phases are the same, we concentrate our evaluation on the efficiency of these processes. These processes include OPRF, issuing/receiving a credential, and proving knowledge of the signature and pseudonym. To simplify the evaluation criteria of the experiment results, we only assume a simple policy with a threshold $t = 2$ for two agents. The experiment is based on a laptop with Intel Core i5-6200U CPU 2.30GHz, 8.00 GB RAM, and 64-bit Ubuntu OS in Java 8 ,

Table 6.2: Performance of the authentication protocol

Sub-Protocol	Duration
OPRF	30 ms
ProveNym	6 ms
IssCred	25 ms
ProveCred	33 ms

Table 6.3: Comparison of single sing-on schemes.

Schemes	Decent.	PV	OA	Anony.	MF	FD	SD
SAML [One19]	×	×	◐	◐	✓	×	×
OpenID [RR06]	×	×	◐	◐	×	×	×
PRIMA [ABS18]	×	×	◐	◐	×	×	≈
IRMA [AvdBH+17]	✓	≈	◐	●	✓	×	✓
EL PASSO [ZKS+20]	≈	✓	◐	●	✓	×	✓
NextLeap [Hal17]	✓	×	◐	◐	✓	×	✓
DAMFA	✓	✓	●	●	✓	✓	✓

building upon the upb.crypto library[1] [BBB+18]. This library offers elliptic curve math and several useful building blocks for the anonymous credential like Sanders signatures [PS16b], Pedersen's commitment [Ped91], Nguyen's accumulator [Ngu05], Shamir secret sharing, generalized Schnorr protocols, proofs of partial knowledge [CDS94], Damgård's technique for concurrently black-box secure Sigma protocols, the Fiat-Shamir heuristic [FS86]. Table 6.2 shows the computational performances of the protocols over 50 iterations. For issuing and proving protocols in such a way that a certain policy is satisfied by a credential, we assume equality of two attributes as Policy: StuID = "11111" and GENDER = "male" and credential: certifying only these attributes.

6.3.4 Computational and Communication complexity

We analyze the communication and the computation complexity of our proposed protocol using the size of each element exchange involved in our protocol, the number of exponentiation needed for issuing a credential (executed only once in the registration phase) and the proving of a credential (the most frequently executed phase), respectively. We show the following efficiency analysis in Table 6.4. r, t, $\mathsf{E}_{\mathbb{G}_1}$ and P denote the number of attributes that can be certified, the number of agents that need to be connected, the cost of exponentiation in $\mathbb{G}_1$ and the cost of a pairing computation, respectively. By $\mathsf{P}OK\{E_{\mathbb{G}_1}[n]\}$ (resp. $\mathsf{P}OK\{P[n]\}$), we denote the cost of proving knowledge of n secrets involved in a

[1] Available at https://github.com/cryptimeleon

Table 6.4: DAMFA computation and communication complexity

Trans.	TOPRF		IssueCred		ProveCred	
	User	PA_i	User	IdP	User	SP
Compu.	2 $E_{\mathbb{G}_1}$	$E_{\mathbb{G}_1}$	$(r+1)E_{\mathbb{G}_1} + POK\{E_{\mathbb{G}_1}[r+1]\}$	$2E_{\mathbb{G}_1} + Ver(POK)$	$2E_{\mathbb{G}_1} + POK\{P[r+1]\}$	$Ver(POK)$
Comm.	$(2t)\|\mathbb{G}_1\|$		$\|\mathbb{G}_1\| + \|POK\|$		$2\|\mathbb{G}_1\| + \|POK\|$	

multi-exponentiation (resp. pairing-product) equation, and $Ver(POK)$ indicates the cost of verifying this proof.

6.3.5 Comparison

We provide a comparison of DAMFA with some of the most popular SSO schemes in Table 6.3 [2]. We compare DAMFA with the above schemes in terms of Decentralization (Decent.), Passive verification (PV), Multi-Factor (MF), (semi-) Formal Definitions (FD), Anonymity (Anony.), and Selective Disclosure (SD) attributes. **Decent** denotes the decentralization of the SSO schemes (i.e., user authentication process no longer depends on a trusted third party). We provide this by applying a distributed transaction ledger and the blind issuing protocol. **PV** shows that service providers can verify users (who have registered a particular credential) without requiring interaction with an identity provider. We fulfill this property using a distributed transaction ledger and anonymous credentials. **Anonymity** guarantees that no one can trace or learn information about the user's identity during the authentication process. We fulfill this property by applying NIZNP + SP signature + Pseudonym. Here, ● denotes that it is infeasible for IdP's to track users' sign-on activity onto different SPs. Also, it shows that it is impossible to correlate multiple accounts created from the same credential on different SPs. Subsequently, ◐ indicates that either IdP's or SPs can create a correlation between different accounts of the same user. **FD** demonstrates if proposed schemes provide a formal security definition. In this case, DAMFA is the only scheme that provides a ***formal security definition and proof***. **SD** allows to disclose a subset of user attributes and proves statements about their attributes. Finally, to protect the user's private information against offline (OA) attacks, we use the TOPRF primitive. Here, ◐ means that other related schemes are resistant against offline attacks as long as IdP does not compromise or the theft/loss/corruption of a user's device does not happen when they use this device as 2FA token. ● means that resistance to offline attacks

[2] Anonymity: NextLeap relies on unlinkable credentials. However, blinded credentials should be stored at IdP, which allows IdP to perform user tracking. Also, in PRIMA, sign-on across multiple SPs can be linked. Other schemes do not support unlinkable credentials. —Offline attacks ◐: the related schemes only fulfilled offline attack if IdP is honest. In IRMA, the user's device (i.e., IRMA app) should be secure to provide OA and anonymity. Otherwise, any adversary who gets these can simply impersonate the user (we addressed this open problem in IRMA). — Selective disclosure: PRIMA supports proving statements about attributes, particularly when they are displayed as extra attributes signed by IdP. — Passive verification ≈: In IRMA, SPs still require to interact with an IRMA API server during the authentication.

is satisfied even in the presence of a corrupted IdP or user's device.

6.4 Summary

In this chapter, we proposed a decentralized authentication and key exchange system DAMFA (SSO scheme) under TOPRF protocol and standard cryptographic primitives. The proposed scheme builds upon a trustworthy global append-only ledger that does not rely on a trusted server. DAMFA fulfills the following properties:

- *Decentralization property* means the process of user authentication no longer depends on a trusted party (or a trusted device). To realize such a distributed ledger, we propose using the blockchain system already in real-world use with Ethereum or Namecoin instead of using a single-party for authentication process. Furthermore, with the help of other primitive like TOPRF, users are relieved of the responsibility to store their sensitive data, such as password, biometric, a single secret key or credentials. This significant advancement not only enhances security but also minimizes the risk of personal device compromise.
- *Passive verification* means that service providers who have access to the shared ledger can verify users without requiring interaction with an identity provider.
- *Single sign-on property* ensures that a user logs in with a single ID into the identity provider and then gains access to any of several related systems. So, users do not need to register with each service provider individually.
- *Anonymity* guarantees that no one can trace or learn information about the user's identity during the authentication process. Finally, we evaluated that our protocol is efficient and practical for authentication systems.

Moreover, we provided comparison of our scheme (DAMFA) with some of the most prominent SSO schemes. To demonstrate a more detailed analysis of the performance of our scheme, we analyzed the communication and the computation complexity of our proposed protocol using the size of each elements exchange involved in our protocol and the number of exponentiation, respectively. We proved our construction's security via ideal-real simulation, showing the impossibility of offline dictionary attacks. Finally, we demonstrated that our protocol is efficient and practical through a prototypical implementation and implemented the public ledger using Ethereum and Namecoin blockchains.

7 Recovery of Encrypted Mobile Device Backups (IDs)

Including electronic identity (eID) such as passports or driving licenses in smartphones transforms them into a single point of failure: loss, theft, or malfunction would prevent their users even from identifying themselves e.g. during travel. Therefore, a secure backup of such identity data is paramount, and an obvious solution is to store encrypted backups on cloud servers. Unfortunately, users will be highly unlikely to remember a cryptographically strong password in the – typically rare but then crucial – case of recovering their eID onto a new device. In this chapter, we propose a new secret key reconstruction protocol that allows clients to recover their secret key from a partially trusted server using biometric authentication (e.g. fingerprint) and auxiliary devices. To this end, we build upon recently popular Password-Protected Secret Sharing (PPSS) schemes with a Fuzzy Extractor to recover the secret key required for decrypting backups from multiple key shares and a biometric identifier. We prove the security of our proposed protocol in the random oracle model where parties can be corrupted separately at any time. The user's biometric and secret keys remain safe as long as both the server and the auxiliary device are not corrupted at the same time. An initial performance analysis shows that it is efficient for this use case.

7.1 Introduction

Moving various aspects of electronic identity (eID) into mobile devices is a growing trend; the standardization of international mobile driving licenses (mDLs) [ISO] and photo ID documents [HRM16] are already being implemented on smartphones[1] as well as mobile payment wallets [GHR+15] and tokens for two-factor authentication[2]. Other (less security-critical) aspects include loyalty cards for shops, public transport tickets, or for simple age verification in various use cases [BBG+13, HRM16, SE16]. The general approach is to transform these elements of eID from formerly physical cards into (often hardware-backed) software components on mobile devices, e.g. as so-called 'secure applets' or 'trusted applications' kept in a tamper-resistant environment to potentially increase both usability and security [BBG+13, NEA14]. However, this trend also creates a major single point of failure. Loss, theft, or simple malfunction of the smartphone becomes highly

[1] Our research group is currently implementing a prototype of the Austrian mobile driving license on Android smartphones with off-line verification, strong privacy guarantees, and scalable revocation. Details will be published in future work.

[2] E.g. by implementing the FIDO U2F or UAF protocols on smartphones with fingerprint sensors.

problematic when the user relies on the smartphone for identification and payment as well as communication. It is therefore clear that all such critical elements need to be backed up, allowing owners to recover them on a new device if necessary — potentially under time pressure and outside of their normal, trusted environments (e.g. when losing a device during travel).

Informally, we define our main user scenario as follows: An individual, Alice, using her smartphone as a digital identity such as a passport and credit card wallet. The smartphone regularly creates a backup of all encrypted data, including payments and eID data. If her phone is stolen, she needs to acquire a new, compatible device and restore her private eID and payment data within a short time frame, probably under great stress. She should have the ability to recover her secret key to use her private eID data. Note that certain additional complexities, such as locking/wiping/revoking her stolen phone, paying for her new device before recovering the virtual wallet, or verifying the authenticity of the new device (which can be a challenging task even for previously used devices [HRM16]), are out of the scope of the current work.

To date, the problem of backing up smartphones (not specifically eIDs) has been typically approached with implicitly trusted cloud services by the respective device or OS manufacturer. Although these services may potentially be made secure with significant technical effort (cf. the recent public presentation of the Apple cloud keystore [Krs16]), they still require complete trust in the operator. Although an organization may try to prevent itself from being able to extract previously stored key material with tamper-resistant hardware, the implementations, and processes for new backups can always be changed without users being able to notice. Adding current issues of legal uncertainty in various countries concerning key escrow and encryption regulations, we argue that this level of trust in a for-profit company subject to a (potentially foreign) legal and political system is misplaced, especially with the implications of handling eIDs.

In this chapter, we try to reduce the required level of trust in cloud services for the backup and recovery of security- and privacy-critical data on smartphones, with a particular focus on the use case of eIDs. The obvious (and naive) approach is to directly derive a cryptographic key from a user-provided password and locally encrypt/decrypt and sign/verify all backup data before sending it to the cloud service. In this case, the service provider would only need to be trusted for providing *availability*, but not for keeping *confidentiality* of the stored data (and *integrity* violations could at least be detected). Current approaches to full-device backup typically use such a method (including both the Android and iOS platforms at this time). However, the well-known difficulty of remembering passwords with high entropy [YBAG04] is even more of a problem for recovery of eID: such a recovery password would only be used very rarely (if at all) and often under duress. At the same time, it needs to be of higher entropy than typical login passwords, because it is the only element keeping a rogue (or legally compelled) service provider from violating the confidentiality of the backup data by simply brute-forcing a weak password. Therefore, a simple password-based key derivation function (PKDF) does not seem to be an appropriate solution and on-device encryption methods have already been extended by including a

hardware-based key part in the derivation function (first on iOS, now also on Android platforms).

Unfortunately, for decrypting backup data on a new device during recovery, we cannot rely on a trusted execution environment (TEE) or other secure hardware to be in possession of a key part to contribute during key derivation, as we assume the original device (from which the backup had been created) to be completely unavailable (Alice's phone was stolen). Our approach, therefore, relies on the following two aspects to enable authenticated recovery from partially trusted cloud services:

- The owner authenticates biometrically, and these biometric identifiers are part of the key derivation function based on a fuzzy extractor. Individuals, therefore, do not have to remember strong passwords. However, we do not assume biometric identifiers (specifically fingerprint data within the scope of this chapter) to be confidential against sufficiently dedicated adversaries.

- To increase the entropy of the resulting cryptographic key, an additional key part is added to the key derivation, akin to the device-specific, hardware-based keys currently used for on-device encryption. As we cannot rely on a single secure hardware component to be available, we split this key into shares that need to be combined during recovery. For instance, Alice can keep one of these shares online in a cloud service, and carry a second one printed as a QRcode with her during traveling or on an auxiliary device.

Recently proposed Password-Protected Secret Sharing (PPSS) schemes allow a user to reconstruct a high-entropy secret from a single (human-memorable) password, by communicating with at least $t + 1$ honest servers (among n possible ones) where the best attacks are online brute force attacks [FK00, BJSL11, JKK14, CEN15]. However, they still suffer from some problems, for example, inefficiency. This is because the PPSS schemes need to communicate with n servers and potentially rely on computationally complex operations like zero-knowledge proofs [JKKX16, JKK14]. In addition, the schemes are not usable for our main scenario, as the user still needs to remember her password until the recovery phase.

To address these problems, we propose a new construction to restore the secure key. In our protocol, the user can securely recover the secret key to newly got devices using biometric and fuzzy extractor cryptography. In the security analysis section, we show that the proposed scheme is secure against different kinds of attacks. In the end, we illustrate the efficiency of our protocol in the performance section.

Our major contributions are as follows:

- We design a new architecture and protocol to reconstruct a secret encryption/decryption/signature key for cloud backup using biometric authentication and a fuzzy extractor.

- To the best of our knowledge, the new protocol is the first provably-secure secret key reconstruction protocol with biometrics in case the user's mobile device gets stolen. Similar to PPSS schemes, our scheme allows eliminating a trusted server as well as removing the secure element required to keep the secret key.
- Third, we illustrate a formal security model for our proposed protocol. Then, we carry out a detailed security analysis to show the proposed protocol is provably secure and satisfies the security requirements of this cloud backup architecture.
- In the performance section, we demonstrate that our protocol is more efficient than the PPSS schemes and practical for cloud backup environments.

7.2 Building block and Notations

In the proposed protocol, we use the elliptic curve diffie-hellman problem to exchange information securely and also use a fuzzy extractor with biometrics to reconstruct the secret key. To ensure consistency with conventional elliptic curve notation, we adopt additive notation in this chapter (see Table 7.1 for our notations used in this chapter). Consequently, we consider the following discrete logarithm problems (DLP) based on an elliptic curve.

7.2.1 Mathematical Problems

Elliptic Curve Discrete Logarithm Problem (ECDLP): For two points $P, Q \in G$, it is difficult to find integer $x \in Z_q$ to fulfill equation $Q = x \cdot P$.

Elliptic Curve Diffie-Hellman Problem (ECDH): Given two points $R, Q \in G_1$, the goal of the ECDH problem is to calculate point $xy \cdot P$ in polynomial time, where $Q = y \cdot P$, $R = x \cdot P$ and x, y are two unknown elements in Z_q.

7.2.2 Fuzzy Extractor

The fuzzy extractor is provided by Dodis et al. [DRS04]. In this section, we briefly describe the basic concepts and the notions related to the fuzzy extractor system.

Definition 60 (Fuzzy Extractor)**.** *A fuzzy extractor is presented with two procedures* $(\mathsf{Gen}, \mathsf{Rep})$*. The fuzzy extractor is formally defined as follows:*

- (Gen) receives biometric B entered by the user, then the procedure will output a random string σ and a random auxiliary string ϑ. Note that, (Gen) is a probabilistic generation procedure.

- (Rep) receives a close biometric entered B^* and the random auxiliary string ϑ, then the procedure will recoverσ. Note, (Rep) is a definitive reproduction procedure. In other words, if $dis(B, B^*) \leq t$. where t is the difference tolerance. Then, $\mathsf{Rep}(B^*, \vartheta) = \sigma$.

We store ϑ to recover σ from the biometric, when B and B^* are adequately close, the string σ can be reproduced entirely. Then, we can use σ as an encryption/authentication key. A strong fuzzy extractor can extract $L = \mid \sigma \mid$ nearly random bits and the probability to guess the biometric key data $\sigma \in \{0,1\}^L$ by an attacker is approximately $\frac{1}{2^L}$.

7.3 System Model

7.3.1 Network Model

The system model of the proposed protocol for a cloud backup architecture includes four types of participants: a mobile user U_i, an authentication server AS as well as F as an auxiliary device.

- AS: It is responsible for the registration of users and providing general information to the registered users. In addition, it generates the global system parameters.
- U_i: A mobile user who sends an authentication request message to the auxiliary device and AS. After user's verification, users gets their information from AS and the auxiliary device. Then they use this information to restore their secret key using the biometrics.
- F: Auxiliary device of the user (such as a laptop, smartphone or a tablet). It helps the user by providing some auxiliary information to return the secret key.

Table 7.1: The notions

Symbol	**Description**
σ	Master key of the user
$SK = h(\sigma.P_{pub_u})$	Secret key of the user
y	private key of the user
$P_{pub_u} = y \cdot P$	Public key of the user
d_{AS}	Private key of the authentication server
$P_{pub_s} = d_{AS} \cdot P$	Public key of the authentication server
K	Private key of the device
$P_{pub_F} = K \cdot P$	Public key of the device
$h()$	A one-way hash function
$H()$	A one-way Bio-hash function
$Enc_k()/Dec_k$	An encryption/decryption function

Note that users can select several devices to keep the same information on them where users can recover the secret key if one of these devices is online.

7.3.2 Threat Model

In order to prove the security of our protocol, we determine the capabilities and possible actions of the attacker. For this purpose, we categorize the adversary into three classes depending on the various types of data accesses. Finally, we will show our scheme is secure under the strongest attacker $\mathcal{A}_3$.

$\mathcal{A}_1$: We consider a probabilistic polynomial-time (ppt) attacker that has perfect control of the communication channels: can eavesdrop on all transformation messages in the public channels, and also modify, change, remove, and add to the network.

$\mathcal{A}_2$: The attacker can at any time attempt to corrupt a party, in which case the attacker knows all the long-term secrets (such as private keys or master shared keys)

$\mathcal{A}_3$: This attacker gets all the internal state information related to a communication entity, including the long-term key of either the AS or the auxiliary device.

Since we assume that the original smartphone is no longer available and there is no custom data stored on the new device, we do not consider any adversary inside either the old or the new smartphone (no hardware modification or malware) within the scope of this chapter. We explicitly assume the mobile device to be secure, including its user interaction channels [May14].

To clarify the threat model, a user U_i can initiate the creation of an account with username and biometric with the authentication server AS and the device F to restore a secret SK protected with biometric B_i and some extra information. We assume that if at least either the authentication server or the device is honest, the biometric B_i and the secret key SK remain secure from the adversary. If both AS and F are corrupted, the adversary has access to all information stored on them, multiple reconstruction queries may be going on concurrently. Thus, the best chance for the adversary to obtain the secret key is attacking the user's biometric (False match). However, in the security analysis section, we show that if the user chooses a strong and appropriate biometric match threshold, the possibility that the adversary extracts the secret key is near zero. Nevertheless, in our threat model, we consider that one of the authentication servers or the device is honest.

7.4 The Proposed Scheme

Based on the fuzzy extractor [DKRS06, DRS04], we present the notion of the secret key reconstruction using biometrics when users lose their phones as a new concept in the area of cloud backup. The proposed protocol consists of two phases: In the initialization phase, where the proposed protocol operations and system setup are defined, we assume that the authentication server stores a table for each user (before their lost their phone), and also the secret key was $SK = h(\sigma.y.P)$ which the user wants to restore. The reconstruction secret key phase will be in online mode with the authentication server and the auxiliary

device. Some notions used during the chapter and a system setup used in the proposed protocol are described as following below:

7.4.1 Assumptions

Our protocol requires an assumption: the authentication server can register its public key and then the user can look up the public key. This is a reasonable assumption in the reconstruction phase because there is only one authentication server. In addition, the authentication server and the device include the maximum number of reconstruction requests, which ensures no online guessing attack.

7.4.2 System Setup Phase

AS executes the following steps to generate the system private key and the system parameters.

- AS chooses two large prime numbers p, q, an elliptic curve $E(F_p)$ defined on F_p for example Curve25519, and a generator P with the order q.
- AS randomly chooses an element $d_{AS} \in Z_q$ as the private key and computes the corresponding public key $P_{pub_{AS}} = d_{AS} \cdot P$.
- AS selects two secure hash functions $h : \{0,1\}^* \rightarrow Z_q^*$ for instance SHA3-512 and one-way Bio-hash function $H()$.
- AS publishes $\{p, q, E(F_p), P, P_{pub_{AS}}\}$ and saves d_{AS} secretly.

7.4.3 Initialization

The goal of users is to generate a key SK so that they can recover it with the help of the authentication server, just using their biometrics. The user thus runs an initialization protocol with the authentication server and the auxiliary device. Finally, users end up with a random key SK and some information. As shown in Figure 7.1, the following steps are executed by users, AS and F.

The users imprint their personal biometric impression B_i on a sensor. Then, U_i computes $(\sigma, \vartheta) = \mathsf{Gen}(B_i)$, where σ is the random secret key the user wants to use to reconstruct the secret key SK and the auxiliary string ϑ for the commitment which is used to restore secret σ. Then, the user generates $(\vartheta_F, \vartheta_S) \longleftarrow \mathsf{ShareGen}(\vartheta)$ such as [CEN15, Sha79b], and $L = h(H(B_i) \parallel \vartheta_F)$ then shares these ϑ_S, ϑ_F among AS and F respectively and sends L only to AS. In addition, U_i computes $Com = h(\sigma, \vartheta, SK, P_{pub_u} = y \cdot P)$ and like auxiliary string makes the split of hash value Com as $(C_S, C_F) \longleftarrow \mathsf{ShareGen}(Com)$. Finally, U_i sends the set of C_S and ϑ_S to the AS and also sends other parts C_F and ϑ_F encrypted to the auxiliary device as following:

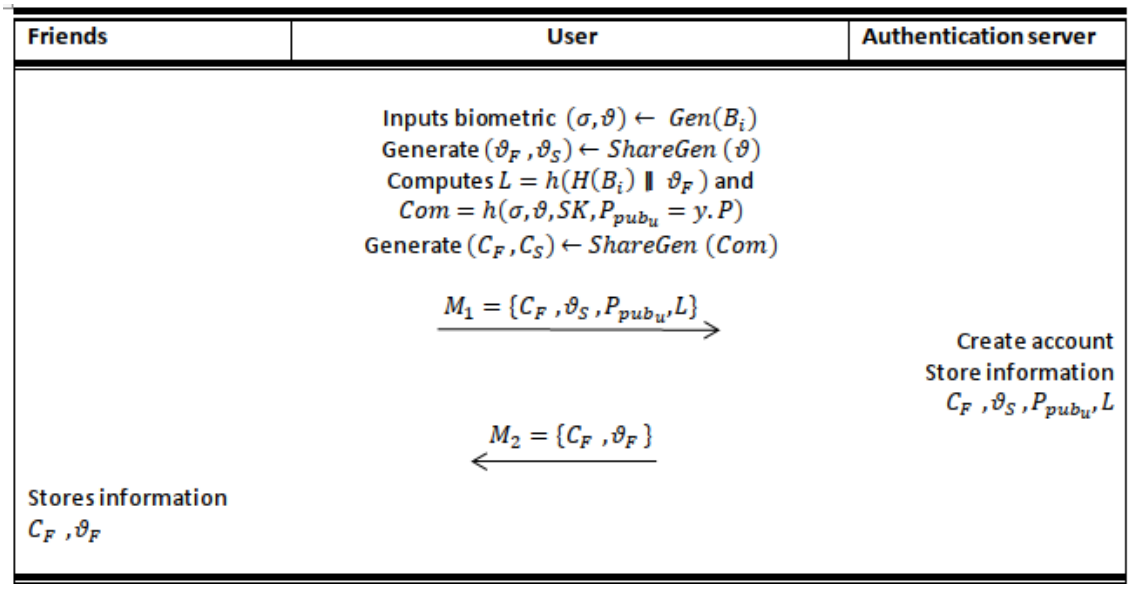


Figure 7.1: Initialization Phase

- U_i generates a random element r_i and computes the scalar multiplication $A = H(B_i) \cdot r_i \cdot P$ sends it to F. Upon receiving A from the user, F computes $b = K \cdot A = K \cdot H(B_i) \cdot r_i \cdot P$ using the secret key K and returns b to U_i.
- After receiving the messages b from the auxiliary device, U_i computes

$$C = r_i^{-1} \cdot b = H(B_i) \cdot K \cdot P \tag{7.1}$$
$$D = h(C) \tag{7.2}$$
$$V = Enc_D(C_F, \vartheta_F). \tag{7.3}$$

At the end, the data V is stored on the auxiliary device and also the server creates a account with the values $\{C_S, \vartheta_S, P_{pub_u}, L\}$ for each user.

7.4.4 Reconstruction Phase

In the initialization phase, we assume that all the communications were safe, and the data is not modified during the communication. In the reconstruction phase, the adversary has control over the network and can forward, alter, delay, replay, or delete any message. The adversary can also provide fake data and may corrupt either the authentication server or the device. The user needs to restore the secret key for a new device in order to restore eID. First, the user sends the authentication request to the auxiliary device and receives part of information ϑ_F and C_F from F. Then, the user sends an authentication message to the authentication server in order to verify the user. After verification by the authentication

server, the user receives another part of information ϑ_S and C_S. In the end, the user can compute SK by combining received information. As shown in Figure 7.2, the following steps are executed by the user, AS and F.

- The user generates random number r_j and computes $A = r_j \cdot H(B_i) \cdot P$. Then, U_i sends a request authentication A to the auxiliary device $M_1 = \{A\}$. Then, the user and the device make use of the following steps to send the other part of auxiliary number ϑ_F and part of commitment information Com C_F, so the auxiliary device calculates:

$$b = K \cdot A = K \cdot r_j \cdot H(B_i) \cdot P \tag{7.4}$$

  Finally, the auxiliary device sends $M_2 = \{b, V\}$ to the user.

- After receiving the message from the device, the user computes:

$$C = r_j^{-1} \cdot b = H(B_i) \cdot K \cdot G \tag{7.5}$$

$$D = h(C) \tag{7.6}$$

$$\{C_F, \vartheta_F\} = Dec_D\{V\}. \tag{7.7}$$

- U_i generates the verification code by inserting his biometric and computes $L^* = h(H(B_i) \parallel \vartheta_F)$. Then, the user encrypts $Enc_{pub_{AS}}\{L^*, A, T_1\}$ where T_1 is the current time-stamp. U_i sends the login request $M_3 = \{E_{pub_{AS}}(L^*, A)T_1\}$ to the AS.

- After receiving the authentication message from the user, AS acquires the current timestamp T_2 and checks if $(T_2 - T_1) > \triangle$, where $\triangle$ is the maximum time interval for transmission delay. If so, then AS rejects the login request; otherwise decrypts the message and checks $L^* = L$ holds, the user is authenticated and then AS continues calculations:

$$S = d_{AS}.A \tag{7.8}$$

$$\omega = Enc_{h(S)}(\vartheta_S, C_S) \tag{7.9}$$

$$H_2 = h(\vartheta_S, C_S, S, P_{pub_{AS}}, \omega) \tag{7.10}$$

  Finally, AS sends the message $M_4 = \{P_{pub_{AS}}, H_2, \omega\}$ to the user.

- Upon receiving the message $M_4 = \{P_{pub_{AS}}, H_2, \omega\}$, U_i computes

$$S^* = H(B_i) \cdot r_j \cdot P_{pub_{AS}} \tag{7.11}$$

$$\{\vartheta_S, C_S\} = Dec_{h(S^*)}(\omega) \tag{7.12}$$

$$H_2^* = h(\vartheta_S, C_S, S^*, P_{pub_{AS}}, \omega). \tag{7.13}$$

  Then it checks whether H_2 and H_2^* are equal. If they are not equal, U_i terminates the session, otherwise, the user combines ϑ_S and ϑ_F to reconstruct the shared ϑ. Also user combines C_S and C_F to reconstruct Com.

- The user computes $\mathsf{Rep}(B_i^*, \vartheta) = \sigma$ If the value of ϑ is changed, robust fuzzy extraction detects it immediately. The user computes the secret key as $SK = h(\sigma \cdot P_{pub_u})$ and $Com^* = h(\sigma, \vartheta, SK, P_{pub_u} = y \cdot P)$. Finally, it checks whether $Com^* = Com$ holds or not. If it holds, the secret key is correct.

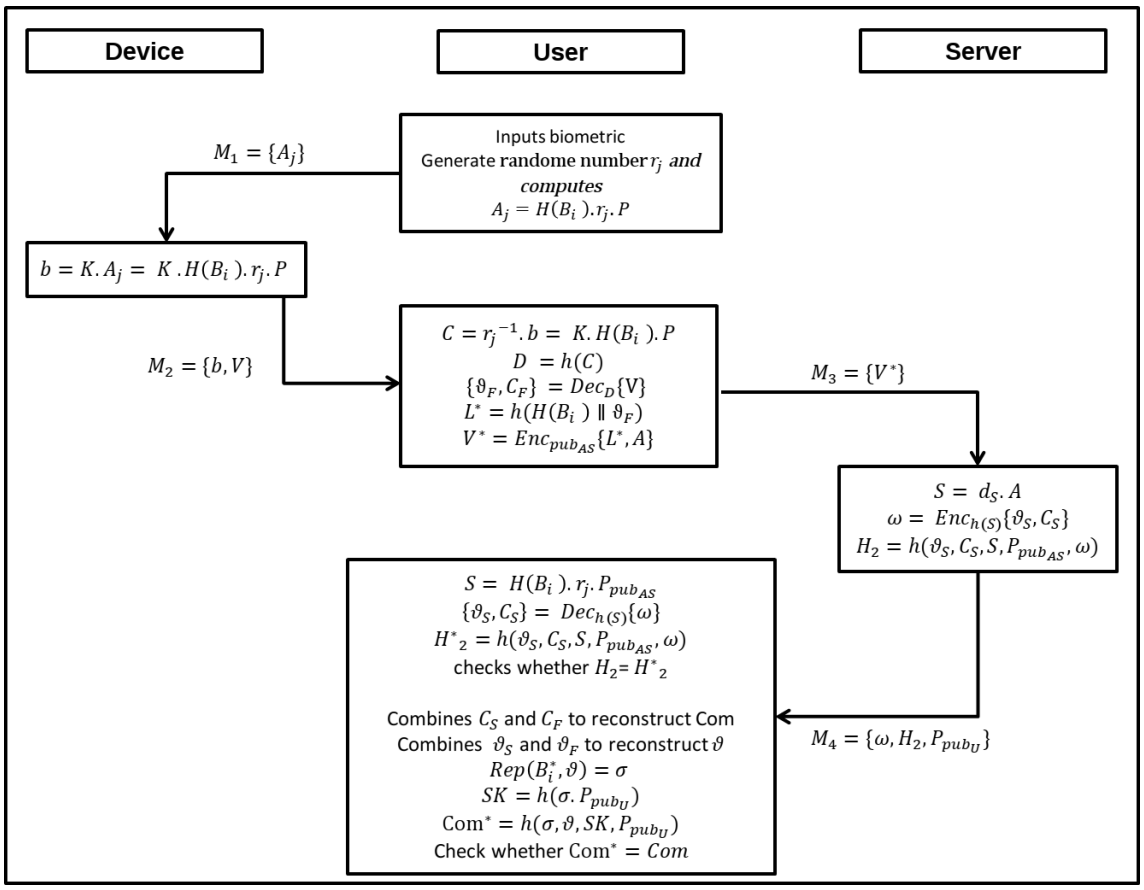


Figure 7.2: Reconstruction Phase

7.5 Security Analysis

We illustrate our protocol is provably secure in a strong security model and prove all security requirements under the random oracle model [BPR00b, CK01, MN15]. The adversary tries to retrieve the user's secret key from the authentic messages and AS database. Generally, we will demonstrate that our scheme is secure against secret key attacks and other related attacks.

7.5.1 Security Model

In the security model, we analyze the security of the proposed scheme by the attacker A_3 to demonstrate that the possibility of breaking the secret key in our scheme is negligible. In this case, the attacker has the ability to compromise one of the authentication servers or devices and reveal their private key [AFP05, MMK17]. The adversary's capabilities are modeled by the various following queries. Before we mention the queries, we introduce two types of participants in our scheme: the Mobile P_i, and the authentication server AS. Note that $\prod_{P_i}^{p}$ and $\prod_{AS}^{s}$ are the instances p and s of P_i and AS_i, respectively. These are explained as the oracles.
The formal security of the retrieve secret key model is based on a game involving a challenger C and a polynomial time adversary $\mathcal{A}$, which will be described below. During the game, the attacker $\mathcal{A}$ is permitted to make the following queries that are responded to by the challenger C.

$h(m)$: C maintains a list L_{h_i} , which is initialized empty. Upon receiving the query, C checks if L h i contains (m, r). If so, C returns r to $\mathcal{A}$; otherwise, C picks a number r randomly, stores (m, r) in L_{h_i} and returns r to $\mathcal{A}$.

Execute ($\prod_{P_i}^{p}$,$\prod_{AS}^{s}$): It is performed by $\mathcal{A}$ in order to get the messages transmitted among two truthful parties. This is modeled as an eavesdropping attack.

Send ($\prod_{P_i}^{p}$,$\prod_{AS}^{s}$): This query is appropriate for modeling an active attack. $\mathcal{A}$ has the ability to modify the message transmitted and send it to a parties instance $\prod_{P_i}^{p}$ and $\prod_{AS}^{s}$ and waits to receive a response message.

Corrupt (P): This query models corruption ability of the adversary $\mathcal{A}$. It produces the private key of the participant P which can be AS or F and also stolen/obtained data stored into them.

Test ($\prod_{P_i}^{p}$): This oracle query is defined to simulate symmetric secret keys semantic security. Upon receiving the query, C chooses a random bit $b \in \{0, 1\}$. If $b = 1$, C returns the secret key of $\prod_{P_i}^{p}$ to $\mathcal{A}$; otherwise ($b = 0$), C generates a random number and returns it to $\mathcal{A}$.

Semantic security, In the random model, the adversary is challenged in an experiment to distinguish between an instance's real user secret key SK and a random number. After carrying out the above queries, $\mathcal{A}$ makes its guess b' of b generated in Test-query. We say $\mathcal{A}$ breaks the security of the scheme, if $\mathcal{A}$ correctly guess $b^{'}$ and wins the game where $b' = b$.

Let $Succ^{P}$ denote the event in which the adversary can successfully guess as $b^{'}$ and wins the game. The advantage of $\mathcal{A}$ in breaking the security of the protocol is defined to be: $\mathcal{A}_P = [2 \cdot \Pr[Succ^{P}] - 1]$. We say that our proposed protocol provides reasonable security

if the advantage $\mathcal{A}_P \leq \in$ is negligible.

Theorem 7.5.1 (Encryption/Decryption Secret Key Security)**.** *Considering the assumptions, our scheme is provably secure against a polynomial-time adversary for deriving the secret key of a user in the random oracle model.The probability that the adversary breaks the secret key security of the proposed scheme P by $\mathcal{A}$ is*

$$\mathcal{A}_P \leq \frac{q_h^2}{|HASH|} + 2 \cdot q_s \cdot max\{(\frac{1}{2^l}, \epsilon_{bm} \cdot \frac{1}{2^m})\} + 2 \cdot \mathcal{A}_P^{ECDLP}(t)$$

Where q_h, q_{send}, $|HASH|$, and $\mathcal{A}_P^{ECDLP}(t)$ define the number of Hash queries, the number of Send queries, the range space of the hash function and the advantage of $\mathcal{A}$ in breaking the ECDLP problem respectively. Let $l =\mid \sigma \mid$ and $m =\mid \vartheta_F \mid, \mid \vartheta_S \mid$, the length of string in the biometrics key σ and ϑ_F (or ϑ_S) respectively.

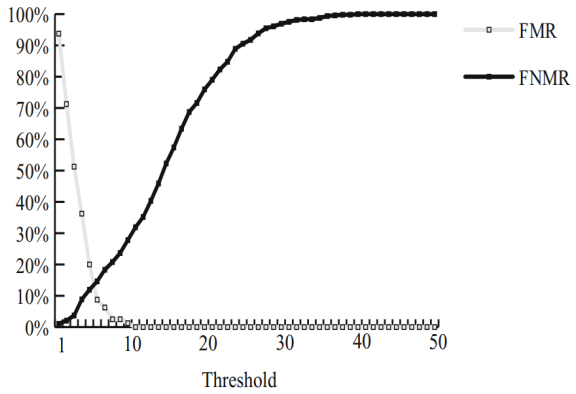


Figure 7.3: False match rate (FMR) and False non match rate (FNMR) evaluation using fuzzy extractor and fingerprint on FVC2000 1a database [AJH07].

7.5.2 Security Proof of the Protocol

Security Proof: We assume, there is a PPT algorithm C that can break a one-way hash function and $ECDLP$ on the problem by cooperating with the adversary. We denote a sequence of games Game_i, where $i = [0, 4]$. Suppose, $Succ_i$ defines the event where the adversary gets success in guessing the bit b in Game_i and wins the game. The game will start from Game_0 as a real attack against the proposed scheme P and ends with the game Game_4 that maintains a negligible advantage of breaking the recovery secret key security of the proposed scheme.

Game_0 This game represents the real attack by the adversary $\mathcal{A}$ against the protocol P in the random oracle model. At the beginning of this game, the bit b is chosen at random. By definition, we have

$$\mathcal{A}_P = [2 \cdot \Pr[Succ^P] - 1]$$

Game_1: In this game, we simulate all the oracles (Execute, Send, and Test oracles) for each query and keep three lists to store the answers to the oracles. $\mathcal{A}$ has to make a decision whether the output of Test is the real secret key or a random number. From the simulation, we can see that the transcript distribution of the game Game_0 and Game_1 are indistinguishable from the real experiment. Therefore, message eavesdropping cannot help to increase the winning possibility of the $\mathcal{A}$'s game. We have,

$$\Pr[Succ_0] = \Pr[Succ_1]$$

Game_2: In this game, we simulate all the oracles in game Game_1, except that we halt all executions in which a collision occurs in the transcript. It transforms Game_1 into Game_2 by adding the simulation of both the Send and Hash oracles. Game_2 creates an active attack where the adversary tries a participant into accepting a forged message. The adversary calls several Hash queries to find the hash collisions. Thus, the games Game_1 and Game_2 are indistinguishable unless the collusions of group points and hash value happen. According to the birthday paradox result [BMP00b] gives the following:

$$\Pr[Succ_1] - \Pr[Succ_2] \leq \frac{q_h^2}{2 \cdot |HASH|}$$

Game_3 This game Game_2 is converted to Game_3 by simulating the corrupt oracle and getting long-term key and data stored from AS or F. Then, the attacker tries to obtain the secret key SK by guessing the values σ. Note that a strong fuzzy extractor is used in our protocol P, which can extract at most l nearly random bits. The guessing probability of the biometric key $\sigma\varepsilon\{0,1\}^L$ by $\mathcal{A}$ is approximately $\frac{1}{2^l}$ [DRS04,ODG15]. Furthermore, we should consider the possible accidental guessing of a "false positive" case with probability ϵ_{bm}. For instance, as shown in Finger 7.3, the authors used the fuzzy extractor on the fingerprint database for results and analysis. Obviously, if the user chooses an appropriate threshold of around 10, we can obtain a high level of security against offline guess attacks. The possibility that a false match happens is approximately 0 and 70 percent usability [AJH07].

Furthermore, the adversary needs to guess ϑ_F or ϑ_S to find string value ϑ (since each of AS or F only has one part of the value ϑ). As a whole, if the length of the string ϑ_F or ϑ_S is $m =\mid \vartheta_F \mid, \mid \vartheta_S \mid$ bits, the guessing probability under this case is at most

$$\Pr[Succ_2] - \Pr[Succ_3] \leq q_s \cdot max\{(\frac{1}{2^l}, \epsilon_{bm} \cdot \frac{1}{2^m})\}$$

Game_4: In the last game, the notion of this security feature is that the adversary $\mathcal{A}$ cannot obtain the secret encryption key even if $\mathcal{A}$ can run corrupt oracle models. The adversary $\mathcal{A}$ goal is to compute the secret key SK in the above case by asking Execute $\prod_{P_i}^{p}$, $\prod_{AS}^{s}$ queries and corresponding hash queries.

The session transcripts $M_3 = \{P_{pub_{AS}}, H_2, \omega\}$ and $M_2 = \{V, b\}$ are available to the adversary where $V = Enc_{h(D)}\{\vartheta_F, C_F\}$ and $b = K \cdot A = K \cdot H(B_i)r_j \cdot P$. $\mathcal{A}$ should ask Hash query to win, and ECDLP problem is broken. However, the secret key SK

is computed as $SK = h(\sigma \cdot y \cdot P)$, and computing SK using these values and available transcripts is computationally infeasible to the adversary. Thus, in this case, the adversary needs to solve the $ECDLP$ problem to compute the secret key SK.

As a result, the game Game_3 and the game Game_4 are indistinguishable as long as the $ECDLP$ assumption holds, due to the random self-reducibility of the $ECDLP$ problem. We can obtain SK in the list L_a with the probability $\frac{1}{q_h}$. Let $t^{'}$ be the running time in all, and we can see that $t^{'} = O(t + (q_s + q_e)T_{pm})$ where T_{pm} denotes the elliptic curve point multiplication operation in ECC and q_e Execution queries. So we have

$$\Pr[Succ_3] - \Pr[Succ_4] \leq O(q_h \cdot \mathcal{A}_p^{ECDLP}(t^{'}))$$

In addition, whatever the bit b involved in the Test-query, the answer is random. Therefore, $\mathcal{A}$ gains no advantage to guess the correct bit b, we get, $\Pr[Succ_4] = 1/2$ From first equation, note that

$$\frac{1}{2} \cdot \mathcal{A}_p = \mid \Pr[Succ_0] - \frac{1}{2} \mid$$

From all the games, we have

$$\mid \Pr[Succ_0] - \frac{1}{2} \mid \leq \frac{q_h^2}{2|HASH|} + max\{q_s(\frac{1}{2^l}, \epsilon_{bm} \cdot \frac{1}{2^m})\} + O(q_h \cdot \mathcal{A}_p^{ECDLP}(t^{'})$$

Thus, using two Equations above, we have

$$\mathcal{A}_P \leq \frac{q_h^2}{|HASH|} + 2 \cdot q_s \cdot max\{(\frac{1}{2^l}, \epsilon_{bm} \cdot \frac{1}{2^m})\} + 2 \cdot \mathcal{A}_P^{ECDLP}(t)$$

7.5.3 Discussion

Security of Key Recovery: In the proposed scheme, the secret key $SK = h(\sigma \cdot P_{pub_u})$ is established to encrypt/ decrypt eID data. To compute the secret key, the adversary needs to know the biometrics of the user and secret value ϑ at the same time. On the other hand, according to the proof of **Theorem 1**, the adversary cannot compute these values without breaking the ECDLP problem, hash function, and compromising the security of both the authentication server and the device. Thus, the security of the secret key is ensured in the proposed scheme.

Although we assume that at least one of the authentication servers or the device should be honest to remains the security of the secret key. However, even if both of them are corrupt, the adversary only has a chance to find biometric by false matching. Nevertheless, as shown in figure 7.3, if we select a suitable threshold for example $T = 10$ in this experiment, the probability to find a false match happens is near zero.

Stolen Verifier Attack: In our scheme, $\{C_S, \vartheta_S, P_{pub_u} = y.P, L\}$ is stored in a verifier

table for each of the users that is maintained by the authentication server. However, even if the adversary has access to the information, he cannot obtain any secret information of U_i without knowing biometrics and ϑ_F. The security of this information is ensured through the usage of the Elliptic Curve Discrete Logarithm Problem (ECDLP) and a robust hash function. As a result, our scheme could withstand the stolen verifier attack.

Impersonation Attack: Suppose, the adversary wants to pass the fake login message $E_{P_{pub_{AS}}}\{L = h(H(B_i) \parallel \vartheta_F)\}$ to the authentication server. According to the proof of **Theorem 1**, the adversary cannot successfully pass a fake login message because they do not possess the secret values ϑ_F and biometric B_i. Even if the adversary obtains ϑ_F by compromising the device, they can only conduct an online brute-force attack with the authentication server (to verify whether the guessed biometric is correct or not). It's important to note that timestamps are utilized to prevent replay attacks. Therefore, our proposed protocol can effectively resist user impersonation attacks.

Privileged Insider Attack: In the registration phase, the authentication server AS stores a random string ϑ_S and $\{P_{pub_u} = y.P, L = h(H(B_i) \parallel \vartheta_F)\}$. Based on **Theorem 1**, it is very difficult for the authentication provider AS to get the user's master key σ without the knowledge of their biometric and ϑ_F even if it uses off-line guessing attacks because the length of ϑ string is large [DKRS06]. Thus, our scheme is secure against privileged insider attacks.

Modification Attack: In the reconstruction phase, all messages from the device and AS are protected under the Diffie Hellman keys D and S and also hash function H_1 and H_2. Thus, the user can discover any modification about the response message by checking if hash values hold. Therefore, the proposed protocol can resist the modification attack.

7.6 Performance

In this section, we analyze the performance of the proposed scheme. We compute the computation cost in the reconstruction phase, and also we compare our scheme with some of the most efficient related works Abdalla et al. [ACNP16], Jarecki et al. [JKKX16], Bagherzandi et al. [BJSL11], Jarecki et al. [JKK14], Camenisch et al. [CLN12], and Camenisch et al. [CLLN14]. For the convenience of evaluating computational cost, let $T_{ecc}, T_h, T_s, T_{hp}, T_e$ be the time cost of executing an elliptic curve point multiplication, hash function operation, an encryption/decryption operation, a hash to point function operation, and an exponentiation operation respectively. We also presume that the execution time needed for a fuzzy extractor in the worst case is almost equal to an elliptic curve point multiplication [HKLS14]. Thus, the user phone in the proposed scheme needs to apply four elliptic curve point multiplication operations, seven general hash function operations one encryption/decryption operation: $4.T_{ecc} + 7.T_h + 1T_s$. Also, the verifier requires to

Table 7.2: Comparison between our protocol and PPSS schemes

Scheme	(t+1,n)	ROM/STD	Client	Inter-server	Msg.	Commun.	Complexity Client|Server
[CLN12]	(2,2)	Std/ROM	CRS	PKI	8	O(1)	O(1)
[JKK14]	any	ROM	CRS	none	2	O(n)	2t+3|3
[BJSL11]	any	ROM	PKI	PKI	3	O(n)	8t+17|16
[CLLN14]	any	ROM	Std	PKI	10	O(n^2)	14t+24|7t+28
[JKKX16]	any	ROM	CRS	none	2	O(n)	t+2|1
Our	(2,2)	ROM	CRS/PKI	none	2	O(1)	O(1)

carry out one elliptic curve point multiplication operation, one decryption operation, and two general hash function operations as $1.T_{ecc} + 2.T_h + 1.T_s$. Mobile devices should carry out one elliptic curve point multiplication operation as $1.T_{ecc}$. Instead, in Abdalla et al.'s One-More-Gap-Diffie-Hellman-based PRF protocol [ACNP16] which is quite similar to the first protocol from [JKKX16] the user needs to compute one hash to point, two exponentiations, and one hash function for each server connection (client cost is $t + 2$ exponentiations). This means that the user should compute $(m + 1)T_e + mT_h + T_{hp}$ which $m = t + 1$ is the number of the honest servers.

In Table 7.2, to compare the computational complexity with related works, we extended the table from [JKK14]. The last column counts (multi)exponentiations in a prime-order group performed by the client and each server in the reconstruction protocol. The costs in the PPSS schemes refer to an optimistic scenario with no adversarial interference. The "total comm." column counts the number of transmitted group elements and objects of length polynomial in the security parameter. The "msgs column" indicates the number of messages per server.

7.6.1 Analysis

In recent years many research papers have implemented cryptography operations, we use the results of execution timings for different operations introduced in [XWW+17, HKLS14]. The hardware platform is a Windows 7 64-bit PC, Intel Core i5-3210M CPU of 2.5 GHz, 8GB RAM, they obtain the running time for cryptographic operations using MIRACAL, a standard cryptographic library [MIR15].
The existing experimental values of these operations are one hash operation requires 0.068 ms (millisecond), one block encryption/decryption requires 0.56 ms, one modular exponentiation requires 3.043 ms, and one scalar multiplication on an elliptic curve requires 2.501 ms. Thus, the user execution time in the proposed scheme is almost $4 \times 2.501 + 7 \times 0.068 + 0.56 = 11.04$. In addition, verifier execution time is about $1 \times 2.501 + 2 \times 0.068 + 0.56 = 3.197$ (ms) and for the device side, this time is $1 \times 2.501 = 2.501$ (ms).

Table 7.3 indicates the computational performance of the reconstruction phase with various thresholds. On the other hand, we ignore operations other than exponentiations

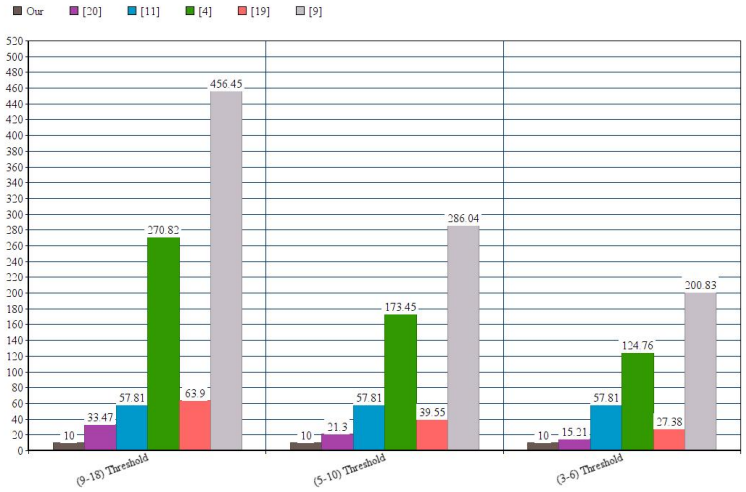


Figure 7.4: User's runtime of various protocols (ms)

and elliptic curve point multiplications because their cost is insignificant in the comparison. The threshold is the number of servers that need to be honest among all the servers. This number depends on the implementation of the protocol. For example, we consider three different thresholds for showing the performance of the relevant works. Note that this is the minimum computations for the user, it can be more according to the number of the server response to the user. According to Figure 7.4, we can see that the improvements of the proposed scheme over relevant works are impressive on the user side. Moreover, the costs of the related works increased rapidly by raising the threshold. It is clear that the proposed scheme has less computation cost than related schemes. So, our scheme is suitable for resource-constrained smart devices.

7.7 Summary

In this chapter, we proposed a new protocol to retrieve the secret key (when a smartphone is lost or stolen, which means the user has also lost the secret key) in order to restore encrypted eID from a cloud backup server. Our proposed scheme allows a mobile user to access multiple eID services from cloud service providers with a new mobile device without the need to completely re-enroll in their eID system, which improves usability. Moreover, our protocol only uses biometric authentication for the user, so they do not need to remember their password. Security analysis shows that the proposed scheme is secure under a realistic threat model, which includes adversarial control of cloud services. Finally, we showed that the computation costs of the proposed protocol are reasonably

Table 7.3: Computation costs comparison (millisecond)

(n, t) Threshold				
Schemes		(9-18)	(5-10)	(3-6)
[CLLN14]	User	150 $T_e \approx 456.45$	$94T_e \approx 286.04$	$66T_e \approx 200.83$
	Server	$91T_e \approx 276.91$	$63T_e \approx 191.70$	$49T_e \approx 149.10$
[JKK14]	User	$21T_e \approx 63.90$	$13T_e \approx 39.55$	$9T_e \approx 27.38$
	Server	$3T_e \approx 9.12$	$3T_e \approx 9.12$	$3T_e \approx 9.12$
[BJSL11]	User	$89T_e \approx 270.82$	$57T_e \approx 173.45$	$41T_e \approx 124.76$
	Server	$16T_e \approx 48.68$	$16T_e \approx 48.68$	$16T_e \approx 48.68$
[CLN12]	User	$19T_e \approx 57.81$	_	_
	Server 1	$26T_e \approx 79.11$	_	_
	Server 2	$30T_e \approx 91.29$	_	_
[JKKX16]	User	$11.T_e \approx 33.47$	$7.T_e \approx 21.30$	$5.T_e \approx 15.21$
	Servers	$9.T_e \approx 27.38$	$5.T_e \approx 15.21$	$3.T_e \approx 9.12$
Our	User	$3.T_{ecc} + T_{fe} \approx 10$	_	_
	Device	$1.T_{ecc} \approx 2.50$	_	_
	Server	$1.T_{ecc} \approx 2.50$	_	_

cheap to make it practical on current mobile devices. Thus, the proposed scheme is more feasible and appropriate than previously proposed PPSS protocols concerning security and usability for recovering eID on mobile devices.

8 Practical Realization (Implementation)

8.1 Introduction

The AC python prototype [1] provides an implementation of the anonymous credentials methods together, primitives, and protocols used and proposed throughout the thesis. With its modular and flexible design, the package offers a unified interface called 'ac.py' that provides a seamless API for working with various anonymous credential schemes while meeting the basic requirements.

The package is further enhanced by two modules: *utils* and *zkp*, which offer fundamental functionality and zero-knowledge proofs for other modules, respectively. Additionally, two folders, primitives and protocols, contain cryptographic primitives (a set of primitives explained in each chapter and the preliminaries chapter, which can be used to build custom anonymous credential systems) and AC protocols (i.e., DAC, TDAC, $\mathsf{IhMA}_{\mathsf{ATMS}}$, and $\mathsf{IhMA}_{\mathsf{AtoSa}}$), respectively.

Although AC protocols share a common aim in providing an AC interface, the differences in their constructions entail certain trade-offs in terms of functionality and efficiency, as discussed in Table 8.1. We compare them in terms of the multi-authority setting **MA**, issuer hiding **IH**, and threshold setting **Th**. For instance, DAC supports generic delegation, while TDAC is designed for the controlled delegation with more restricted delegation power. $\mathsf{IhMA}_{\mathsf{ATMS}}$ and ($\mathsf{IhMA}_{\mathsf{ATMS}}$ are designed for multi-authority settings and provide IH while they do not provide delegation. More details on the trade-offs between the schemes are explained in the respective chapter.

Table 8.1: Comparison of our AC schemes

	Del	**MA**	**IH**	**Th**
Chapter 3 ($\mathsf{IhMA}_{\mathsf{ATMS}}$)	×	✓	✓	×
Chapter 3 ($\mathsf{IhMA}_{\mathsf{AtoSa}}$)	×	✓	✓	×
Chapter 4 (DAC)	✓	×	×	×
Chapter 5 (TDAC)	✓	×	×	✓

Overall, the choice of the concrete construction depends on the specific use case or application and the priorities set in the overall system.

[1] https://github.com/mir-omid/AC-Package

8.2 Architecture

The anonymous credential package is composed of several modules and components as described below:

```
ac_package/
├── primitive/
│   ├── atosa.py
│   ├── amts.py
│   ├── set_commit.py
│   ├── spe_enc.py
│   ├── spseq_uc.py
│   └── spseq_signs.py
├── protocols/
│   ├── mac_atosa.py
│   ├── mac_amts.py
│   ├── dac.py
│   └── tdac.py
├── zkp.py
├── utils.py
└── ac.py
tests/
```

As mentioned, the "primitives" folder contains the primitives required for the anonymous credential system, while the "protocols" folder contains modules that implement the anonymous credential protocols. Additionally, an easy way to learn how to use the library is to explore the tests (including tests for all components). To describe the remaining aspects of the library, we provide the following information:

- *utils.py*: Provides basic functions and utilities required by other modules.
- *amts.py*: Implements the ATMS scheme with the randomization of keys, tags, messages, and signatures.
- *atosa.py*: Provides a class implementation of the AtoSa scheme with the randomization of keys and tags.
- *spseq-uc.py*: Implements SPSEQ-UC signatures, which are used in delegatable anonymous credentials (DAC).
- *spe-enc.py*: Provides a class implementation of a threshold delegatable subset predicate encryption TDSPE scheme.
- *spseq-signs.py*: Implements SPSEQ signatures (i.e., FHS [FHS19] and Mercurial [CL19]).

- *set-commit.py*: Provides an implementation of set commitments. Additionally, the module implements a cross-set commitment, enabling the aggregation of witnesses across multiple commitments into a single witness. It also includes a variant of set commitment that allows set commitments in tag-based Diffie-Hellman messages.

- *zkp.py*: Provides a variant of Schnorr-style proofs used in our protocols, such as the Schnorr proof (non-interactive using the Fiat-Shamir heuristic) of statements $ZK(x, m_1, ..., m_n; h = P^x \wedge h_1^{m_1}, \ldots, h_n^{m_n})$ and a generalized version of Damgard's technique that extends interactive proof for obtaining malicious-verifier interactive zero-knowledge proofs of knowledge, among others.

We provide detailed documentation of each module in the tests folder, which explains the details of the modules and how to use them.

8.3 Dependencies

Our library is based upon the `bplib` library[2] and petlib [3] with `OpenSSL` bindings[4] to improve speed. We use the popular pairing friendly curve BN256 which provides efficient type 3 bilinear groups at a security level of around 100 bits.

8.4 AC Interfaces (APIs)

Now let's look at the AC interface that provides a set of basic methods that are required by an anonymous credential system. The methods are as follows:

- *setup*: This abstract method is used to set up the anonymous credential system and generate public parameters.

- *user-keygen*: This method is a default implementation for user key generation.

- *nym-gen*: This method creates a pseudonymous for the user public key.

- *isuser-keygen*: This method is a default implementation for issuers' key generation.

- *issue-cred*: This abstract method is used to issue a credential for a user.

- *proof-cred*: This abstract method is used to generate proof for a credential.

- *verify-proof*: This abstract method is used to verify the validity of a proof.

[2] https://github.com/gdanezis/bplib
[3] https://github.com/gdanezis/petlib
[4] https://github.com/dfaranha/OpenPairing

The AC interface encompasses four distinct concrete implementations, each possessing its own set of unique properties. However, despite their differences, the methods within these implementations share a common structure for keys and attributes. In this shared structure, messages are represented as vectors of strings, while keys are represented as vectors of integers. These concrete implementations are as follows:

- *dac*: This implementation provides additional methods for delegation.
  - *delegator* $\leftrightarrow$ *delegatee*: Create a delegatable credential from user U to a user R.
- *tdac*: This implementation provides additional methods for delegation and threshold.
  - *agg-cred*: Aggregate the threshold number of credentials and create a single credential.
  - *delegate*: Create a delegatable credential (non-interactive) from user U to a user R.
- *mac-atosa* and *mac-atms*: This implementation provides an implementation of AC system.
  - *gen-policies*: Generate keys policies for issuer hiding.
  - *agg-cred*: Aggregate different credentials from various issuers.

Note that each implementation of a protocol may have its own unique approaches and methods. For instance, in the case of the *tdac*, its setup phase and credentials issuing are designed to work in a threshold setting. On the other hand, the *mac-atosa* and *mac-atms* protocols are used to perform showing credentials with respect to IH, each with its own distinct characteristics and benefits.

8.5 Summary

In summary, the Python package implements the anonymous credentials, primitives, and protocols mentioned in this thesis. The package includes all the necessary building blocks to construct these new protocols and showcase their practical applications and proof of concept. It provides an abstract interface for AC systems, primitives that can be utilized to construct custom credential systems, and concrete implementations of the AC interface for four distinct anonymous credential systems. This package aims to demonstrate the practicality of AC systems, ensure consistency and clarity in presenting the new anonymous credentials, and provide guidance on integrating and connecting them.

9 Conclusion

In this thesis, we address the need for privacy-enhancing technologies in the context of digital identity. While digital credentials play a crucial role in enabling access to various online and offline services, relying on centralized identity providers raises concerns regarding user privacy and data security. To mitigate these issues, we propose using Anonymous Credentials (AC) as a foundation for authentication and authorization, allowing individuals to retain control over their personal information.

The research focuses on several key aspects, leading to the development and optimization of AC schemes. First, we introduce the concept of Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA), which addresses the need for independent credentials from multiple issuers without disclosing the issuers' identities. This protects user privacy, especially in decentralized settings. The proposed solution involves the development of two new primitives: Aggregate Signatures with Randomizable Tags and Public Keys (AtoSa) and Aggregate Mercurial Signatures (ATMS), which enhance the functionality of IhMA by providing aggregation, randomization, and tag capabilities.

Additionally, we present a novel Delegatable Anonymous Credential (DAC) scheme that allows credential owners to delegate their credentials while ensuring attributes, anonymity, and the ability to restrict further delegations. The DAC credentials remain of constant size, and the scheme utilizes Structure-Preserving Signatures on Equivalence Classes on Updatable Commitments (SPSEQ-UC) to achieve unlinkable showings and public randomization.

Furthermore, we introduce Threshold Delegatable Anonymous Credentials (TDAC), representing the first decentralized and delegatable AC system with threshold issuance. The scheme leverages a Threshold Delegatable Subset Predicate Encryption (TDSPE) scheme, enabling partial decryption keys to be issued by multiple authorities and supporting users' generation of full decryption keys.

We propose a decentralized scheme that combines AC systems and Oblivious Pseudorandom Function (OPRF) with Multi-Factor Authentication (MFA) to address privacy-preserving single sign-on. This eliminates the reliance on a single trusted third party for user authentication and allows for anonymous authentication within user groups.

Lastly, we present a secure protocol for recovering encrypted mobile device backups (eID) in case of smartphone loss or malfunction. Using a Fuzzy Extractor, clients can leverage biometric authentication and auxiliary devices to recover their secret keys from partially trusted servers.

The proposed solutions are accompanied by rigorous security definitions, formalized notations, efficient instantiations, and performance benchmarks based on implementation.

This research contributes to the development of privacy-preserving technologies for digital identity, offering practical and secure approaches to protect user privacy while enabling the benefits of digital credentials in various domains.

Open Issues and Future Work.

- *Anonymous credentials in the standard model*: While we have made significant progress in the field of anonymous credentials (AC), there are still several open issues that merit further investigation. One important area of future work is the exploration of AC schemes in the standard model. Currently, our AC constructions rely on idealized assumptions such as random oracles and GGM models. Extending the study of AC schemes to the standard model would provide a more rigorous analysis of their security properties and strengthen their practical applicability.

- *ACs based on post-quantum cryptography*: As quantum computing continues to advance, it is crucial to explore the development of AC schemes that are resistant to attacks from quantum adversaries. Future work can focus on investigating and designing AC protocols that are based on post-quantum cryptographic primitives, such as short and randomizable signatures based on post-quantum hard problems.

- *Revocation in anonymous credentials*: In terms of functionalities, extending our AC system with revocation mechanisms is also an interesting direction to explore. In many scenarios, the revocation of credentials represents an important property; however, revocation mechanisms pose a significant challenge in designing anonymous credential systems. Consequently, efficient revocation mechanisms for our approach that do not rely on re-issuing is an interesting question for future research.

- *Fully dynamic aggregate signatures*: Another interesting open question is whether it is possible to ATMS and AtoSa signatures in Chapter 3 in a fully dynamic setting, i.e., there are no assumptions about prior knowledge of messages and verification keys, nor requirement for a stateful issuer to keep track of the signed information aux.

- *Exploring new application for our new primitives*: In addition to addressing the open issues mentioned above, another interesting future work can also focus on exploring new applications for the novel primitives developed in this thesis. The AtoSa and ATMS primitives have versatile features that extend the functionality of signatures. Investigating and identifying new domains where these primitives can be effectively utilized would expand the scope of anonymous credentials and their impact on privacy-preserving technologies.

Bibliography

[ABS18] Muhammad Rizwan Asghar, Michael Backes, and Milivoj Simeonovski. Prima: privacy-preserving identity and access management at internet-scale. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[ACNP16] Michel Abdalla, Mario Cornejo, Anca Nitulescu, and David Pointcheval. Robust password-protected secret sharing. In *European Symposium on Research in Computer Security*, pages 61–79. Springer, 2016.

[AFG+16] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. 29(2):363–421, April 2016.

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In *International Workshop on Public Key Cryptography*, pages 65–84. Springer, 2005.

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. pages 473–484, 2010.

[AHS11] Gergely Alpár, Jaap-Henk Hoepman, and Johanneke Siljee. The identity crisis. security, privacy and usability issues in identity management. *arXiv preprint arXiv:1101.0427*, 2011.

[AJH07] Arathi Arakala, Jason Jeffers, and K Horadam. Fuzzy extractors for minutiae-based fingerprint authentication. *Advances in Biometrics*, pages 760–769, 2007.

[AMMM18] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. Pasta: Password-based threshold authentication. In *ACM Conference on Computer and Communications Security (CCS)*, pages 2042–2059. ACM, 2018.

[AvdBH+17] Gergely Alpár, Fabian van den Broek, Brinda Hampiholi, Bart Jacobs, Wouter Lueks, and Sietse Ringers. Irma: practical, decentralized and privacy-friendly identity management using smartphones. *HotPETs 2017*, 2017.

[BB18] Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. pages 221–239, 2018.

[BBB+18] Kai Bemmann, Johannes Blömer, Jan Bobolz, Henrik Bröcher, Denis Diemert, Fabian Eidens, Lukas Eilers, Jan Haltermann, Jakob Juhnke, Burhan Otour, Laurens Porzenheim, Simon Pukrop, Erik Schilling, Michael Schlichtig, and Marcel Stienemeier. Fully-featured anonymous credentials with reputation system. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018)*, number 42, pages 1–10. IEEE, 2018.

[BBG+13] Patrik Bichsel, Bud Bruegger, Alberto Crespo Garcia, Thomas Gross, André Gutwirth, Moritz Horsch, Detlef Houdeau, Charles Bastos Rodriguez, and Tarvi Martens. Survey and analysis of existing eid and credential systems. Deliverable D32.1, Apr 2013.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In *Cryptographers' Track at the RSA Conference*, pages 226–243. Springer, 2006.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. pages 132–145, 2004.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. pages 108–125, 2009.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. pages 117–136, 2016.

[BCET21a] Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. Anonymous transactions with revocation and auditing in hyperledger fabric. In *International Conference on Cryptology and Network Security*, pages 435–459. Springer, 2021.

[BCET21b] Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. Anonymous transactions with revocation and auditing in hyperledger fabric. In *Cryptology and Network Security - 20th International Conference, CANS*, volume 13099, pages 435–459. Springer, 2021.

[BCHB+09] Patrik Bichsel, Jan Camenisch, Tom Heydt-Benjamin, Dieter Sommer, and Greg Zaverucha. Cryptographic protocols of the identity mixer library. 2009.

[BCL16] Julien Bringer, Hervé Chabanne, and Roch Lescuyer. Software-only two-factor authentication secure against active servers. In *International Conference on Cryptology in Africa (AFRICACRYPT)*, pages 285–303. Springer, 2016.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. https://eprint.iacr.org/2020/081.

[BDLP98] Jørgen Brandt, Ivan Damgård, Peter Landrock, and Torben P. Pedersen. Zero-knowledge authentication scheme with secret key exchange. 11(3):147–159, June 1998.

[BEK+20] Jan Bobolz, Fabian Eidens, Stephan Krenn, Daniel Slamanig, and Christoph Striecks. Privacy-preserving incentive systems with highly efficient point-collection. pages 319–333, 2020.

[BEK+21] Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In *International Conference on Cryptology and Network Security*, pages 158–178. Springer, 2021.

[BELO15] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. pages 23–41, 2015.

[BF20] Balthazar Bauer and Georg Fuchsbauer. Efficient signatures on randomizable ciphertexts. pages 359–381, 2020.

[BFGP22] Daniel Bosk, Davide Frey, Mathieu Gestin, and Guillaume Piolle. Hidden issuer anonymous credential. *Proc. Priv. Enhancing Technol.*, 2022(4):571–607, 2022.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. pages 403–422, 2011.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. pages 416–432, 2003.

[BHKS18] Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Signatures with flexible public key: Introducing equivalence classes for public keys. pages 405–434, 2018.

[BJDF16] Alberto Blanco-Justicia and Josep Domingo-Ferrer. Privacy-aware loyalty programs. *Computer Communications*, 82:83–94, 2016.

[BJS10] Lujo Bauer, Limin Jia, and Divya Sharma. Constraining credential usage in logic-based access control. pages 154–168, 2010.

[BJSL11] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 433–444. ACM, 2011.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted Key Exchange: Password-Based Protocols SecureAgainst Dictionary Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 1992.

[BM93] Steven M. Bellovin and Michael Merritt. Augmented Encrypted Key Exchange: A Password-based Protocol Secure Against Dictionary Attacks and Password File Compromise. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 244–250. ACM, 1993.

[BMP00a] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably Secure Password-authenticated Key Exchange Using Diffie-Hellman. In *Proceedings of the International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 156–171. Springer, 2000.

[BMP00b] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology—Eurocrypt 2000*, pages 156–171. Springer, 2000.

[Boy08] Xavier Boyen. The uber-assumption family (invited talk). pages 39–56, 2008.

[BP13] Fabrice Benhamouda and David Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013.

[BPR00a] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Proceedings of the International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 139–155. Springer, 2000.

[BPR00b] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology—EUROCRYPT 2000*, pages 139–155. Springer, 2000.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. pages 62–73, 1993.

[BS22] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. 2022.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. pages 321–334, 2007.

[Cam17] Dell Cameron. Over 560 Million Passwords Discovered in Anonymous Online Database (2017), 2017.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[CCD+17] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. pages 901–920, 2017.

[CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. pages 683–699, 2017.

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. pages 262–288, 2015.

[CDL+20] Jan Camenisch, Manu Drijvers, Anja Lehmann, Gregory Neven, and Patrick Towa. Short threshold dynamic group signatures. pages 401–423, 2020.

[CDM00] Ronald Cramer, Ivan Damgård, and Philip D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. pages 354–372, 2000.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO '94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[CDT19] Jan Camenisch, Manu Drijvers, and Björn Tackmann. Multi-protocol uc and its use for building modular and efficient protocols. *IACR Cryptol. ePrint Arch.*, 2019:65, 2019.

[CEN15] Jan Camenisch, Robert R Enderlein, and Gregory Neven. Two-server password-authenticated secret sharing uc-secure against transient corruptions. In *IACR International Workshop on Public Key Cryptography*, pages 283–307. Springer, 2015.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Annual International Cryptology Conference*, pages 19–40. Springer, 2001.

[Cha85] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. pages 515–534, 2007.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Phil MacKenzie. Universally Composable Password-based Key Exchange. In *Proceedings of the Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 404–421. Springer, 2005.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 453–474. Springer, 2001.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. pages 100–119, 2013.

[CKLM14a] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable Signatures: New Definitions and Delegatable Anonymous Credentials. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 199–213. IEEE, 2014.

[CKLM14b] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. pages 199–213, 2014.

[CKP+22] Elizabeth Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. Cryptology ePrint Archive, Paper 2022/839, 2022. https://eprint.iacr.org/2022/839.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *International Workshop on Public Key Cryptography*, page 481. Springer, 2009.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. pages 93–118, 2001.

[CL03] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. pages 268–289, 2003.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. pages 56–72, 2004.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. pages 78–96, 2006.

[CL19] Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. pages 535–555, 2019.

[CL21] Elizabeth C. Crites and Anna Lysyanskaya. Mercurial signatures for variable-length messages. 2021(4):441–463, October 2021.

[CLK21] Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. Cryptology ePrint Archive, Report 2021/1680, 2021. https://eprint.iacr.org/2021/1680.

[CLLN14] Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *International Cryptology Conference*, pages 256–275. Springer, 2014.

[CLN12] Jan Camenisch, Anna Lysyanskaya, and Gregory Neven. Practical yet universally composable two-server password-authenticated secret sharing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 525–536. ACM, 2012.

[CLN15] Jan Camenisch, Anja Lehmann, and Gregory Neven. Optimal distributed password verification. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 182–194. ACM, 2015.

[CLPK22] Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In *IACR International Conference on Public-Key Cryptography*, pages 409–438. Springer, 2022.

[CMZ14a] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. pages 1205–1216, 2014.

[CMZ14b] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1205–1216. ACM, 2014.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. pages 1445–1459, 2020.

[CRS+21] Valerio Cini, Sebastian Ramacher, Daniel Slamanig, Christoph Striecks, and Erkan Tairi. Updatable signatures and message authentication codes. pages 691–723, 2021.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). pages 410–424, 1997.

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, 2018.

[DKL+23] Jack Doerner, Yashvanth Kondi, Eysa Lee, LaKyah Tyner, et al. Threshold bbs+ signatures for distributed anonymous credential issuance. *Cryptology ePrint Archive*, 2023.

[DKRS06] Yevgeniy Dodis, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. In *Annual International Cryptology Conference*, pages 232–250. Springer, 2006.

[DMM+18] Dominic Deuber, Matteo Maffei, Giulio Malavolta, Max Rabkin, Dominique Schröder, and Mark Simkin. Functional credentials. 2018(2):64–84, April 2018.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *International conference on the theory and applications of cryptographic techniques*, pages 523–540. Springer, 2004.

[FGHP09] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. pages 309–324, 2009.

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. pages 233–253, 2015.

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. 32(2):498–546, April 2019.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. pages 152–168, 2005.

[FK00] Warwick Ford and Burton S Kaliski. Server-assisted generation of a strong secret from a password. In *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, pages 176–180. IEEE, 2000.

[FKS17] Daniel Fett, Ralf Küsters, and Guido Schmitz. The web SSO standard openid connect: In-depth formal security analysis and security guidelines. In *Computer Security Foundations Symposium (CSF)*, pages 189–202. IEEE, 2017.

[FMA14] Nils Fleischhacker, Mark Manulis, and Amir Azodi. A modular framework for multi-factor authentication and key exchange. In *International Conference on Research in Security Standardisation*, pages 190–214. Springer, 2014.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques (Eurocrypt)*, pages 186–194. Springer, 1986.

[Fuc11] Georg Fuchsbauer. Commuting signatures and verifiable encryption. pages 224–245, 2011.

[FVY14] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A Decentralized Public Key Infrastructure with Identity Retention. *IACR Cryptology ePrint Archive*, 2014.

[Gem15] Gemalto. 2014 Year of mega breaches & identity theft: Findings from the breach level index, 2015.

[GGM14a] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. 2014.

[GGM14b] Christina Garman, Matthew Green, and Ian Miers. Decentralized Anonymous Credentials. In *The Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2014.

[Gha16] Essam Ghadafi. Short structure-preserving signatures. pages 305–321, 2016.

[GHR+15] Mohsen Guizani, Daojing He, Kui Ren, Joel JP Rodrigues, Sammy Chan, and Yan Zhang. Security and privacy in emerging networks: Part ii [guest editorial]. *IEEE Communications Magazine*, 53(8):40–41, 2015.

[GJKR99] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, pages 295–310. Springer, 1999.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. 20(1):51–83, January 2007.

[GK10] Adam Groce and Jonathan Katz. A new framework for efficient password-based authenticated key exchange. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 516–525. ACM, 2010.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, pages 524–543. Springer, 2003.

[GL06] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. *Journal of Cryptology*, 19(3):241–340, 2006.

[GP16] Vindu Goel and Nicole Perlroth. Yahoo Says 1 Billion User Accounts Were Hacked, 2016.

[Gro15] Jens Groth. Efficient fully structure-preserving signatures for large messages. pages 239–259, 2015.

[GRWZ20] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. pages 2007–2023, 2020.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. pages 415–432, 2008.

[Hal17] Harry Halpin. Nextleap: Decentralizing identity with privacy for secure messaging. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pages 1–10, 2017.

[HJ12] Dick Hardt and Michael Jones. The oauth 2.0 authorization framework: Bearer token usage. 2012.

[HKLS14] Debiao He, Neeraj Kumar, Jong-Hyouk Lee, and R Sherratt. Enhanced three-factor security protocol for consumer usb mass storage devices. *IEEE Transactions on Consumer Electronics*, 60(1):30–37, 2014.

[HP22] Chloé Hébant and David Pointcheval. Traceable constant-size multi-authority credentials. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*, volume 13409 of *Lecture Notes in Computer Science*, pages 411–434. Springer, 2022.

[HRM16] Michael Hölzl, Michael Roland, and René Mayrhofer. Real-world identification: Towards a privacy-aware mobile eid for physical and offline verification. In *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*, pages 280–283. ACM, 2016.

[HRMM18] Michael Holzl, Michael Roland, Omid Mir, and René Mayrhofer. Bridging the gap in privacy-preserving revocation: Practical and scalable revocation of mobile eids. In *Proceedings of the 33rd Annual ACM Symposium on Applied*

Computing, SAC '18, page 1601–1609, New York, NY, USA, 2018. Association for Computing Machinery.

[HRMM20] Michael Hölzl, Michael Roland, Omid Mir, and René Mayrhofer. Disposable dynamic accumulators: toward practical privacy-preserving mobile eids with scalable revocation. *International Journal of Information Security*, 19(4):401–417, 2020.

[HS14] Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. pages 491–511, 2014.

[HS21] Lucjan Hanzlik and Daniel Slamanig. With a little help from my friends: Constructing practical anonymous credentials. In *Proc. of CCS '21*, pages 2004–2023. ACM, 2021.

[HW15] Debiao He and Ding Wang. Robust biometrics-based authentication scheme for multiserver environment. *IEEE Systems Journal*, 9(3):816–823, 2015.

[HW18] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. pages 197–229, 2018.

[HXB+14] Xinyi Huang, Yang Xiang, Elisa Bertino, Jianying Zhou, and Li Xu. Robust multi-factor authentication for fragile communications. *IEEE Transactions on Dependable and Secure Computing*, 11(6):568–581, 2014.

[ILV11] Malika Izabachène, Benoît Libert, and Damien Vergnaud. Block-wise P-signatures and non-interactive anonymous credentials with efficient attributes. pages 431–450, 2011.

[ISO] ISO/IEC 18013-5. Personal identification – ISO-compliant driving licence – Part 5: Mobile driving licence (mDL) application.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 233–253. Springer, 2014.

[JKKX16] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: how to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 276–291. IEEE, 2016.

[JKKX17] Stanisław Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: Cost-minimal Password-Protected Secret Sharing based on Threshold OPRF. In *International Conference on Applied Cryptography and Network Security*, pages 39–58. Springer, 2017.

[JKSS16] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. Device-enhanced password protocols with optimal online-offline protection. In *Proceedings of the ACM on Asia Conference on Computer and Communications Security (ASIA CCS)*, pages 177–188. ACM, 2016.

[JKSS18] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. Two-Factor Authentication with End-to-End Password Security. In *International Conference on Practice and Theory of Public Key Cryptography (PKC)*. Springer, Cham, 2018.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *Proceedings of the Conference on Theory of Cryptography (TCC)*, pages 577–594. Springer, 2009.

[JLG04] Andrew Teoh Beng Jin, David Ngo Chek Ling, and Alwyn Goh. Biohashing: two factor authentication featuring fingerprint data and tokenised random number. *Pattern recognition*, 37(11):2245–2255, November 2004.

[KL17] Dmitry Khovratovich and Jason Law. Sovrin: digital identities in the blockchain era, 2017.

[KLAP20] Hyoseung Kim, Youngkyung Lee, Michel Abdalla, and Jong Hwan Park. Practical dynamic group signature with efficient concurrent joins and batch verifications. Cryptology ePrint Archive, Report 2020/921, 2020. https://eprint.iacr.org/2020/921.

[KMMS17] Jonathan Katz, Matteo Maffei, Giulio Malavolta, and Dominique Schröder. Subset predicate encryption and its applications. pages 115–134, 2017.

[Krs16] Ivan Krstić. Behind the scenes' ios security talk, 2016.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. pages 146–162, 2008.

[Lin11] Yehuda Lindell. Anonymous authentication. *Journal of Privacy and Confidentiality*, 2(2), 2011.

[LLY13] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Aggregating CL-signatures revisited: Extended functionality and better efficiency. pages 171–188, 2013.

[LMPY16] Benoît Libert, Fabrice Mouhartem, Thomas Peters, and Moti Yung. Practical "signatures with efficient protocols" from simple assumptions. pages 511–522, 2016.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. pages 74–90, 2004.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. pages 465–485, 2006.

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. pages 568–588, 2011.

[MAR18] Mustafa A Mustafa, Aysajan Abidin, and Enrique Argones Rúa. Frictionless authentication system: Security & privacy analysis and potential solutions. *arXiv preprint arXiv:1802.07231*, 2018.

[May14] Rene Mayrhofer. An architecture for secure mobile devices. *Security and Communication Networks*, 2014.

[MBG+23] Omid Mir, Balthazar Bauer, Scott Griffy, Anna Lysyanskaya, and Daniel Slamanig. Aggregate signatures with versatile randomization and issuer-hiding multi-authority anonymous credentials. 2023. https://eprint.iacr.org/2023/1016.

[MIR15] MIRACLć. Multiprecision integer and rational arithmetic cryptographic library,", 2015.

[MMHN18] Omid Mir, René Mayrhofer, Michael Hölzl, and Thanh-Binh Nguyen. Recovery of encrypted mobile device backups from partially trusted cloud servers. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–10, 2018.

[MMK17] Omid Mir, Jorge Munilla, and Saru Kumari. Efficient anonymous authentication with key agreement protocol for wireless medical sensor networks. *Peer-to-Peer Networking and Applications*, 10(1):79–91, 2017.

[MN15] Omid Mir and Morteza Nikooghadam. A secure biometrics based authentication with key agreement scheme in telemedicine networks for e-health services. *Wireless Personal Communications*, 83(4):2439–2461, 2015.

[MRM20] Omid Mir, Michael Roland, and René Mayrhofer. Damfa: Decentralized anonymous multi-factor authentication. In *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '20, page 10–19, 2020.

[MRM22] Omid Mir, Michael Roland, and René Mayrhofer. Decentralized, privacy-preserving, single sign-on. *Security and Communication Networks*, 2022:1–18, 2022.

[MSBM23] Omid Mir, Daniel Slamanig, Balthazar Bauer, and René Mayrhofer. Practical delegatable anonymous credentials from equivalence class signatures. *Proc. Priv. Enhancing Technol.*, 2023(3):488–513, 2023.

[MSM+18] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. DelegaTEE: Brokered Delegation Using Trusted Execution Environments. In *27th USENIX Security Symposium*, pages 1387–1403, 2018.

[MSM23] Omid Mir, Daniel Slamanig, and René Mayrhofer. Threshold delegatable anonymous credentials with controlled and fine-grained delegation. *IEEE Transactions on Dependable and Secure Computing*, pages 1–16, 2023.

[NEA14] Thomas Nyman, Jan-Erik Ekberg, and N Asokan. Citizen electronic identities using tpm 2.0. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pages 37–48. ACM, 2014.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, 2005.

[OBM+18] Aleksandr Ometov, Sergey Bezzateev, Niko Mäkitalo, Sergey Andreev, Tommi Mikkonen, and Yevgeni Koucheryavy. Multi-factor authentication: A survey. *Cryptography*, 2(1):1, 2018.

[ODG15] Vanga Odelu, Ashok Kumar Das, and Adrijit Goswami. A secure biometrics-based multi-server authentication protocol using smart cards. *IEEE Transactions on Information Forensics and Security*, 10(9):1953–1966, 2015.

[One19] OneLogin, Inc. SAML Toolkits, 2019.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. pages 214–231, 2009.

[Pau12] Ian Paul. LinkedIn Confirms Account Passwords Hacked, 2012.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference (Crypto)*, pages 129–140. Springer, 1991.

[PS16a] David Pointcheval and Olivier Sanders. Short randomizable signatures. pages 111–126, 2016.

[PS16b] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Cryptographers' Track at the RSA Conference*, pages 111–126. Springer, 2016.

[PZ08] David Pointcheval and Sébastien Zimmer. Multi-factor Authenticated Key Exchange. In *6th International Conference on Applied Cryptography and Network Security (ACNS)*, pages 277–295. Springer, 2008.

[PZ11] Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1. 1. *Technical Report, Microsoft Corporation*, 2011.

[RPJ+18] Vera Rimmer, Davy Preuveneers, Wouter Joosen, Mustafa A Mustafa, Aysajan Abidin, Enrique Argones Rúa, et al. Frictionless Authentication Systems: Emerging Trends, Research Challenges and Opportunities. *arXiv preprint arXiv:1802.07233*, 2018.

[RR06] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16, 2006.

[RWGM22] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. zkcreds flexible anonymous credentials from zksnarks and existing identity infrastructure. *Cryptology ePrint Archive*, 2022.

[SAB+19] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. 2019.

[San20] Olivier Sanders. Efficient redactable signature and application to anonymous credentials. pages 628–656, 2020.

[San21] Olivier Sanders. Improving revocation for group signature with redactable signature. pages 301–330, 2021.

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 459–474. IEEE, 2014.

[Sch90] Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 239–252. Springer, 1990.

[SE16] Kris Shrishak and Asst Prof Dr Zekeriya Erkin. Enhancing the privacy of users in eid schemes through cryptography. *Literature Survey, Delft University of Technology, Delft*, page 45, 2016.

[Sha79a] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[Sha79b] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SJSN14] Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[SKI10] SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Anonymous password-authenticated key exchange: New construction and its extensions. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 93(1):102–115, 2010.

[tea16] Namecoin team. Namecoind, sourcecode of the namecoin-client reference implementation, August 2016.

[VYT05] Duong Quang Viet, Akihiro Yamamura, and Hidema Tanaka. Anonymous password-based authenticated key exchange. In *Proceedings of the International Conference on Cryptology in India (INDOCRYPT)*, pages 244–257. Springer, 2005.

[WCW+17] Ding Wang, Haibo Cheng, Ping Wang, Xinyi Huang, and Gaopeng Jian. Zipf's law in passwords. *IEEE Transactions on Information Forensics and Security*, 12(11):2776–2791, 2017.

[XWW+17] Qi Xie, Duncan S Wong, Guilin Wang, Xiao Tan, Kefei Chen, and Liming Fang. Provably secure dynamic id-based anonymous two-factor authenticated key exchange protocol with extended security model. *IEEE Transactions on Information Forensics and Security*, 12(6):1382–1392, 2017.

[YAXY19] Rupeng Yang, Man Ho Au, Qiuliang Xu, and Zuoxia Yu. Decentralized blacklistable anonymous credentials with reputation. *Computers & Security*, 85:353–371, 2019.

[YBAG04] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security and privacy*, 2(5):25–31, 2004.

[YHCL15] Xun Yi, Feng Hao, Liqun Chen, and Joseph K Liu. Practical threshold password-authenticated secret sharing protocol. In *European Symposium on Research in Computer Security*, pages 347–365. Springer, 2015.

[YZ08] Jing Yang and Zhenfeng Zhang. A new anonymous password-based authenticated key exchange protocol. In *International Conference on Cryptology in India (INDOCRYPT)*, pages 200–212. Springer, 2008.

[ZKS+20] Zhiyi Zhang, Michał Król, Alberto Sonnino, Lixia Zhang, and Etienne Rivière. El passo: Privacy-preserving, asynchronous single sign-on. *arXiv preprint arXiv:2002.10289*, 2020.

[ZXSM17] Rui Zhang, Yuting Xiao, Shuzhou Sun, and Hui Ma. Efficient multi-factor authenticated key exchange scheme for mobile communications. *IEEE Transactions on Dependable and Secure Computing*, page 1, 2017.