Privacy-Preserving Protocols: Advancing Security and Flexibility with Policy-Based Sanitizable Signatures and Fair Exchange Mechanisms

by

Ismail Sami Abdelaziz Afia
B.Sc., Benha University, Egypt, 2004
M.Eng., Nile University, Egypt, 2020

A Dissertation Submitted in Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

We acknowledge and respect the Lək̓ʷəŋən (Songhees and Xʷsepsəm/ Esquimalt) Peoples on whose territory the university stands, and the Lək̓ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Privacy-Preserving Protocols: Advancing Security and Flexibility with Policy-Based Sanitizable Signatures and Fair Exchange Mechanisms

by

Ismail Sami Abdelaziz Afia
B.Sc., Benha University, Egypt, 2004
M.Eng., Nile University, Egypt, 2020

Supervisory Committee

Dr. Riham AlTawy, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Issa Traore, Department Member
(Department of Electrical and Computer Engineering)

Dr. Yun Lu, Outside Member
(Department of Computer Science)

ABSTRACT

This dissertation presents advancements in privacy-preserving protocols, focusing on two research areas: policy-based sanitizable signature schemes and fair exchange mechanisms.

Sanitizable signature schemes allow designated parties to modify or sanitize signed messages while preserving the message's authenticity. We present the Unlinkable Policy-Based Sanitizable Signature (UP3S) scheme, which addresses a significant deficiency in existing policy-based sanitizable signature schemes, the lack of unlinkability. A crucial security property, particularly in privacy-sensitive applications, unlinkability guarantees that distinct sanitized versions of a given message cannot be linked to the original message or to each other, even across multiple sanitization operations.

Building upon UP3S, we investigate extending its capabilities to support fine-grained control over message modifications. This involves enabling multiple modification policies for a single message and facilitating the delegation of sanitization rights. To this end, we propose the Traceable Policy-Based Signature (TPBS) scheme, which forms the basis for the Extended Policy-Based Sanitizable Signature (EP3S). EP3S offers a flexible and secure framework for policy-based sanitizable signatures, incorporating enhanced control over message modifications and sanitization-rights delegation.

In the area of fair exchange mechanisms, our contributions focus on privacy-preserving exchanges of both digital and physical assets. We introduce V2VFx, a privacy-preserving framework for the fair exchange of physical assets, specifically in vehicle-to-vehicle energy trading.

Together, these contributions advance the state of privacy-preserving protocols by addressing key limitations in existing schemes and extending their applicability.

Contents

List of Tables

List of Figures

List of Acronyms

ABE	Attribute-Based Encryption
ABS	Attribute-Based Signature
DTABS	Decentralized Traceable Attribute-Based Signature
EU-CMA	Existential Unforgeability under Chosen-Message Attacks
GS	Group Signature
NIZK	Non-Interactive Zero-Knowledge Proofs
PBS	Policy-Based Signatures
PB3S	Policy-Based sanitizable signature
PS	Pointcheval-Sanders RDS Scheme
RDS	Rerandomizable Digital Signature
SE-NIZK	Simulation-Sound Extractable NIZK
3S	Sanitizable Signature Schemes
TABS	Traceable Attribute-Based Signature
TPBS	Traceable Policy-Based Signature
UP3S	Unlinkable Policy-Based Signature Scheme
E3S	Extended Sanitizable Signature
EP3S	Extended Policy-Based Sanitizable Signature

Dedication

To my wife, AlShaimaa Abdelhamid, my life companion - without your unwavering support, none of this would have been possible.

To my late father, Prof. Dr. Sami Afia - your dream has finally come true.

To my mother, Mrs. Boshra Abdelsattar - thank you for your endless prayers and encouragement, which lit my path when I needed it most.

To my children, Roshan, Youssof, and Yassin - thank you for your love and patience throughout this entire journey.

To Dr. Marianne Azer and Dr. Ahmed Abdelkhalek - thank you for believing in me when it mattered.

And to my supervisor, Dr. Riham AlTawy - I am deeply grateful for your constant support and guidance throughout this journey.

Chapter 1

Introduction and Motivation

In an era of increasing privacy concerns, the demand for effective privacy-preserving technologies has become essential. Two particularly active areas of research within this domain are privacy-preserving digital signature schemes and private fair exchange mechanisms.

Digital signatures have long been a fundamental component of secure communication and transactional systems, ensuring the authenticity and integrity of digital documents. However, traditional digital signature schemes often fall short when it comes to protecting the signer's identity, ensuring the privacy of the signed content, and providing efficient, flexible control over the use and modification of signed messages.

To address these shortcomings, **privacy-preserving signature schemes** have emerged as a promising solution. These cryptographic protocols are designed to authenticate digital messages while protecting aspects of privacy, such as the signer's identity and/or message content. Examples include sanitizable signatures [9], ring signatures [85], group signatures [13], and blind signatures [40]. Each of these schemes offers distinct privacy features, but they still face scalability, flexibility, and accountability challenges.

Sanitizable Signature Schemes (3S) were introduced to protect the privacy of signed

content by enabling selective information disclosure to authorized parties while preserving the privacy of other message components [9]. These schemes are particularly valuable in contexts governed by the security principle of need-to-know, such as healthcare and data management. For example, in electronic health records, a signed patient record might contain Personally Identifiable Information (PII), diagnosis, and treatment plan. While the accounting department may require access to some PII and the treatment plan for billing, they should not have access to the diagnosis. Conversely, the research department might need the diagnosis and treatment plan but not the patient's PII. 3S facilitates this selective disclosure of patient records while maintaining their authenticity. In 3S, a semi-trusted entity, the sanitizer, is authorized to modify specific parts of a signed message without invalidating the original signature. However, traditional 3S schemes often lack fine-grained control over sanitization rights, do not support delegation of these rights, and do not provide control over the content of modified parts.

Policy-Based Sanitizable Signature Schemes (PB3S) [89] were introduced as a natural extension of 3S schemes to address some of these limitations. PB3S schemes enable signers to assign sanitization rights to any entity that complies with a predefined access policy, providing more **granular control** over sanitization rights. Unlike traditional 3S schemes, PB3S schemes do not require sanitizers to be known by the signer in advance, significantly enhancing their flexibility. However, existing PB3S schemes, such as P3S [89], suffer from several significant drawbacks, including the lack of unlinkability, which guarantees that distinct sanitized versions of a given message cannot be linked to the original message, suboptimal efficiency, and scalability challenges when accommodating multiple sanitizers.

Another extension of 3S schemes is the **Extended Sanitizable Signature Scheme (E3S)**,

which provides signers with the ability to restrict the sanitizer's choice of modifications [65]. E3S introduces several enhancements to 3S, such as limiting the possible modification of a designated part of the message to a predefined set of changes, forcing the same modifications on different designated parts of the message, limiting the number of alterations of designated parts, and restricting the number of versions a sanitizer can produce from an original signed message. However, all proposed E3S schemes are defined in a single sanitizer setting and require interaction between the signer and the sanitizer before signature generation, and sometimes require that the sanitizer remains online during this phase.

Another active area of research within the domain of effective privacy-preserving technologies is privacy-preserving ***fair exchange*** protocols. In a fair exchange protocol, two parties aim to exchange assets such that, at the end of the protocol, either both parties successfully obtain the counterpart's asset or neither party does. Fair exchange protocols fundamentally address the simultaneity problem [54], which traditionally necessitates the involvement of an external trusted third party (TTP). However, reliance on a TTP introduces major challenges, particularly from a privacy perspective: first, establishing a fully trustworthy relationship with the TTP can be difficult; and second, even a benign TTP may inadvertently compromise sensitive information about the parties, their assets, or the nature of the transaction. As privacy becomes increasingly critical, there is a growing demand for fair exchange protocols that minimize or eliminate trust in external entities while rigorously preserving the privacy of both participants and transaction details.

1.1 Problem Statement and Motivation

The existing literature on **Policy-Based Sanitizable Signature (PB3S) schemes** and the privacy-preserving **fair exchange** protocols reveals several limitations that hinder their effectiveness and efficiency.

The only current proposal for Policy-Based Sanitizable Signature schemes, **P3S** [89], suffers from significant drawbacks. Firstly, P3S allows linking sanitized signatures to the same original message, posing a security risk by potentially enabling the inference of combined knowledge about the original message. Additionally, P3S requires re-executing the setup algorithm for each new message, resulting in suboptimal efficiency. To maintain transparency, where distinguishing between freshly signed and sanitized messages is infeasible, the signature size grows linearly with the number of sanitizers, which negatively affects scalability.

Moreover, P3S uses restrictive modification policies that only specify which parts of the message authorized sanitizers can modify, without allowing the signer to control the content of modifications. It also lacks the capability for signers to define multiple modification policies for the same message and does not support the delegation of sanitization rights to other sanitizers with additional restrictions.

The current **Policy-Based Signature (PBS)** scheme [12] could potentially address some of these limitations if it were incorporated as the main building block in a PB3S scheme by providing anonymity for the sanitizer. The current and only proposed PBS protocol [12] lacks traceability and is vulnerable to key misuse. Specifically, signers can deny responsibility for a signed message, and policy key holders may share their keys with unauthorized individuals, allowing them to sign messages under the issuer's name without accountability.

These limitations present significant challenges for designing policy-based privacy-preserving sanitizable signature schemes, affecting their integrity, privacy, scalability, and accountability. Addressing these challenges is essential for ensuring robust privacy, efficient operation, and accountability in digital signature schemes.

On the other hand, the privacy-preserving fair exchange presents its own challenges, particularly in maintaining fairness, privacy, and integrity without relying on trusted third parties. However, blockchain technology offers a potential solution to replace TTP. Existing blockchain-based protocols often address only specific privacy aspects, such as transaction amount or identity, while neglecting others like offer, location, and contractual terms. Furthermore, many solutions rely on off-chain trusted third parties (TTPs) for critical operations, introducing single points of failure and trust dependencies that undermine decentralization. Alternatively, approaches based on consortium blockchains introduce substantial deployment overheads and limit scalability. Even among these, many lack a robust fair exchange mechanism that prevents premature transaction abortion, exposing both buyers and sellers to potential losses. Finally, a common deficiency is the absence of formal security proofs, which are crucial for establishing confidence in these solutions.

Motivation Given the limitations of existing PB3S schemes and the challenges of fair exchange, there is an urgent need for advanced privacy-preserving protocols that address these shortcomings. Specifically, there is a need for:

- **Policy-Based Sanitizable Signature Schemes** that provide unlinkability, fine-grained control over modifications, support for multiple modification policies, and the ability to delegate sanitization rights, while also ensuring accountability for both signers and key holders.

- **Privacy-Preserving and Fair Trading Protocols** that facilitate secure, verifiable,

and fair exchanges of digital or physical assets, without compromising privacy or efficiency, and without relying on trusted third parties.

1.2 Research Objectives

The objective of this research is to develop advanced privacy-preserving protocols that address the limitations of existing schemes. The research is divided into two main areas: **policy-based sanitizable signature schemes** and privacy-preserving **fair exchange mechanisms**. Specifically, the research aims to:

- **Enhance Policy-Based Sanitizable Signature Schemes** by introducing unlinkability, fine-grained control over message modifications, support for multiple modification policies, and enabling delegation of sanitization rights, thereby providing greater flexibility, privacy, and accountability in applications such as healthcare and digital identity management. In this regard:
  - We introduce **UP3S**, a policy-based sanitizable signature scheme that achieves unlinkability, does not require new secrets to be shared with future sanitizers before sanitizing each message, and maintains a fixed signature size for a given sanitization policy.
  - We propose **TPBS**, a Traceable Policy-Based Signature scheme that ensures traceability and prevents key misuse while maintaining signer anonymity. TPBS serves as the foundation for developing up next research contribution EP3S.
  - We propose **Extended Policy-Based Sanitizable Signature (EP3S)**. EP3S expands the signer's control over modification policies while maintaining unlinkability. EP3S allows signers to control not only who can sanitize specific parts

of a message but also what modifications are permitted. Furthermore, EP3S enables the definition of multiple modification policies for the same message and allows authorized sanitizers to delegate their sanitization rights with additional restrictions.

- **Advancing Privacy-Preserving and Fair Trading Protocols** where we introduce **V2VFx**, a protocol for the fair exchange of physical assets, focusing on vehicle-to-vehicle energy trading. V2VFx ensures that energy transactions are conducted with privacy, fairness, and integrity in a decentralized manner.

Both contributions of this dissertation, enhancing policy-based sanitizable signature schemes and advancing privacy-preserving fair exchange protocols, are driven by a common overarching objective: strengthening privacy-preserving mechanisms in decentralized, trust-minimized environments. While they address different facets of privacy protection, they are unified by a shared vision of enhancing users' ability to control sensitive information and interactions without relying on trusted third parties.

Specifically, the enhancements to policy-based sanitizable signatures (UP3S, TPBS, and EP3S) focus on giving signers fine-grained, unlinkable control over how their signed messages can be modified and by whom, ensuring privacy, accountability, and flexibility even after signature issuance. In parallel, the development of the V2VFx protocol addresses the challenge of conducting fair asset exchanges while preserving transaction privacy and minimizing trust assumptions. Both lines of work are synergistic: strong, flexible signature mechanisms like UP3S and EP3S are critical building blocks for enabling decentralized, privacy-preserving exchange frameworks like V2VFx. Together, they form a comprehensive advancement of privacy-preserving protocols that can support a wide range of emerging applications in secure communications, digital identity management, and decentralized

marketplaces.

Looking ahead, future work can further investigate the integration of advanced policy-based sanitizable signature schemes within decentralized fair exchange systems, leveraging the fine-grained control and unlinkability properties to enable more adaptive, privacy-respecting transaction frameworks.

1.3 Organization of the Thesis

The remainder of this proposal is organized as follows. Chapter 2 provides the literature background and the previous work for the privacy-preserving protocols proposed in this thesis. Chapter 3 provides key cryptographic building blocks that are used as the foundation of various proposed protocols of this thesis. Chapter 4 provides the proposal for an Unlinkable Policy-Based Signature Scheme (UP3S) [5]. Chapter 5 provides the proposal for a Traceable Policy-Based Signature scheme [4]. Chapter 6 provides the proposal for an Extended Policy-Based Sanitizable Signatures scheme [6]. Chapter 7 provides the proposal for a Privacy-Preserving Fair Exchange Framework for Vehicle-to-Vehicle Energy Trading (V2VFx).

Chapter 2

Background

This chapter provides the necessary background on sanitizable signature schemes (3S) and their fundamental security properties, including unforgeability, immutability, privacy, accountability, and transparency. It then presents a review of the literature on Policy-Based Sanitizable Signature Schemes (PB3S), Policy-Based Signatures (PBS), and Extended Sanitizable Signature Schemes (E3S). Finally, the chapter provides the concept of fair exchange with a particular emphasis on its role in ensuring both fairness and privacy.

2.1 Sanitizable Signature Schemes

Sanitizable signature schemes (3S) allow the signer of a message to designate a semi-trusted entity called the sanitizer to alter a signed message in a controlled way, and yet the original signature of the message is verified successfully [9]. The original signer of the message controls the modification process by defining which blocks of the message are allowed to be modified. Sanitizable signature schemes enabled numerous applications where the modification of a signed message is required without interaction with its signer. Such applications

include outsourced databases, multicast transmissions, secure routing, privacy-preserving document disclosure, and privacy-preserving dissemination of patient data in healthcare applications [9, 28, 75].

The standard security notions of sanitizable signatures include unforgeability, immutability, privacy, accountability, and transparency which are informally defined as follows.

- **Unforgeability** This notion implies that an adversary with no access to either the signer or the sanitizer secret keys cannot generate a verifiable signature under honestly generated keys.
- **Immutability**. This security notion implies that an adversary with no access to the signer's secret key can not alter inadmissible blocks.
- **Privacy**. This notion implies that it is infeasible to use sanitized signatures to recover information about the sanitized parts of the message.
- **Accountability**. This security notion ensures that the origin of a signature whether signed or sanitized is undeniable. More precisely, accountability implies that if a signer (resp. sanitizer) did not sign (resp. sanitize) a message, then a malicious sanitizer (resp. signer) should not be able to accuse the signer (resp. sanitizer).
- **Transparency**.This notion requires that no adversary can distinguish between sanitizable signatures created by the signer or the sanitizer.

Additionally, unlinkability has been presented by Brzuska *et al.* as a required security notion for privacy-preserving 3S-based applications [9, 28]. Intuitively, unlinkability ensures that associating different sanitized signatures with a source original message, i.e., linking the two sanitized versions of the same message, is not feasible. Hence, concluding

combined information about the original message is prevented. For instance, in healthcare applications, if we have two sanitized message signature pairs of a specific patient's medical records where one of the messages contains only the personal information of the patient and the other is an anonymized version of the same patient's health records, linking both message signatures may lead to the reconstruction of the full medical records of the patient. Consequently, within the literature on sanitizable signature schemes [10, 32, 36, 42], constructing unlinkable ones has been the objective of the works in [30, 31, 49, 71].

Privacy-preserving protocols aim not only to secure data but also to minimize the exposure of relationships between different actions, documents, or participants. Unlinkability is a critical security property in this context. In the domain of sanitizable signatures, unlinkability guarantees that distinct sanitized versions of a signed message cannot be linked either to each other or to the original signature, thereby preventing the inference of sensitive correlations. Without unlinkability, even partial knowledge or multiple views of sanitized data could enable adversaries to reconstruct confidential information or deanonymize users. As privacy threats increasingly rely on cross-correlation and inference attacks, achieving unlinkability is fundamental to protecting user privacy, particularly in sensitive applications such as healthcare, finance, and decentralized trading systems. Consequently, unlinkability has become a key design goal in modern privacy-preserving protocols and forms a central motivation for the enhancements proposed in this dissertation.

Broadly, in the literature, several sanitizable signature schemes have been presented, which are classified by Bilzhause *et al.* [20] into four major categories as follows, i) schemes that provide additional security properties such as non-interactive public accountability [29] and invisibility [31, 32], ii) schemes that support multiple signers and sanitizers [27, 36, 37, 70], iii) schemes that limit the sanitizer ability to alter admissible blocks to signer chosen

values [35, 45], and iv) schemes that allow the sanitization of encrypted data [10, 42].

2.2 Policy-Based Sanitizable Signature Schemes

Although sanitizable signature schemes are a great tool that fits privacy-preserving applications, they do not offer the signer granular fine-grained control over defining who can sanitize the message. Sanitizable signature schemes are usually defined in a single-sanitizer setting where the sanitizer is known in advance to the signer before the signature generation process. Conversely, trapdoor sanitizable signature schemes -a subclass of the sanitizable signature schemes- enable the signer to grant sanitization rights to sanitizer(s) after signature generation [36, 37]. However, such schemes often require interaction between the sanitizer and the signer after signature generation to obtain trapdoor information [70]. To tackle the aforementioned limitation, Samelin and Slamanig proposed the first Policy-Based Sanitizable Signature Scheme called P3S, where sanitization rights are assigned to any sanitizer that fulfills a predefined access policy [89]. Hence, sanitization is enabled based on possible sanitizer(s) attributes determined by the signer rather than sanitizers' public keys that may be unknown to the signer at the time of signing. Accordingly, sanitizers are not required to be known to the signer before signature generation. P3S employs a Policy-Based Chameleon hash (PCH) [43] and a dynamic-group-signature similar primitive as its building blocks [14]. PCH allows sharing of the encryption of the trapdoor information of a chameleon hash function with possible sanitizers of a given message using an attribute-based encryption algorithm (ABE) where the sanitization policy controls who can decrypt the trapdoor [86]. More precisely, if a sanitizer possesses the attributes that satisfy the sanitization policy defined by ABE, it can retrieve the chameleon hash trapdoor information and

then sanitize the message by modifying its chameleon hash parameters. On the other hand, P3S accountability is achieved by a group signature similar primitive in which the signer/sanitizer of a given message provides the encryption of their public key in addition to a non-interactive zero-knowledge (NIZK) proof that the encryption hides either the signer's or the sanitizer's public key. The use of PCH in P3S facilitates linking two signatures together because the message hash is not changed with each sanitization process. Moreover, for each new message, the P3S setup has to be executed, where the encryption of the PCH trapdoor secret key is shared with all sanitizers, which does not lead to the most efficient instantiation. We also observe that to maintain the transparency security notion where it is infeasible to distinguish a freshly signed message from a sanitized one, the size of the group signature in P3S should grow linearly with the number of sanitizers, which may further affect the system's scalability

2.3 Policy-Based Signatures

Another class of privacy-preserving protocols is Policy-Based Signatures (PBS). In PBS schemes, a signer can produce a valid signature of a message only if the message satisfies a specific hidden policy [12]. PBS schemes allow an issuer to delegate signing rights to specific signers under a particular policy (by sharing a signing policy key). Yet, the produced signature is verifiable under the issuer's public key. Besides unforgeability, the standard security notion for signature schemes, the privacy of the PBS scheme ensures that signatures do not reveal the policy under which they were created.

Generally speaking, PBS schemes aim to extend the functionality of digital signature schemes by offering some form of delegation of signing rights under the issuer's policy

signing key. Although there exist some primitives that offer signing rights delegation, such as group signatures (GS) [13] and Attribute-Based Signatures (ABS) [5,73], PBS introduces some distinct features that other primitives do not fulfill. For instance, in GS schemes, a member signs any message on behalf of the whole group. However, PBS schemes give the issuer fine-grained control over who is allowed to sign which messages. PBS hides the policy under which the signature is created and requires that the signed message conforms to the hidden policy. The authors in [12] show how PBS could be used to construct group signatures and attribute-based signatures. Furthermore, they show how the PBS scheme naturally fits privacy-preserving protocols such as signatures of knowledge [39], ring signatures [85], mesh signatures [25], proxy signatures [50], and anonymous credentials [33].

PBS framework allows delegation, where a signer holding a key for some policy can delegate such a key to another signer with possible restrictions on the associated policy. Delegation enables the signing of messages that satisfy both the original and restricted policies. The holder of the newly generated key can further delegate such a key to another signer with possible further restrictions and so on. Such a delegatable PBS scheme suites applications in hierarchical settings. For instance, if an issuer in a certain organization granted one of the managers the signing rights of contracts with clients X, Y, and Z, such a manager can delegate these signing rights to a team leader in his unit. Furthermore, the manager may restrict such rights and limit the team leader to signing contracts with client Z only.

The standard security requirements of PBS schemes are unforgeability and privacy [12]. Unforgeability ensures that an adversary cannot create a valid signature without having a policy key where the signed message conforms to such a policy. Privacy guarantees that a signature does not reveal the policy associated with the key. Privacy also implies unlinkability, where an adversary cannot decide whether two signatures were created using the

same policy key.

2.4 Traceable Policy-Based Signature Schemes (TPBS)

Although the PBS privacy definition ensures full signer anonymity, it permits key misuse without accountability. For instance, a signer of a given message may deny their responsibility for such a signature, especially in a delegatable setting where signers delegate their signing keys to others, signing accountability becomes of vital value. Furthermore, policy key holders (delegated or not) may share their keys with anyone that authorizes them to sign messages under the issuer's name without any sort of liability over the signed message.

In an attempt to tackle the aforementioned problem, Xu *et al.* have proposed a traceable policy-based signature scheme [96]. In their proposal, the user's identity is attached to the policy. More precisely, the issuer generates signing keys for the user ensuring that the user's identity is part of the key, i.e. generating the signing keys for $id||p$, where id denotes the user identity and p denotes the policy under which the signer is allowed to sign a specific message. To sign a message, the signer first encrypts their identity under the public key of an opener and provides a Non-Interactive Zero Knowledge (NIZK) proof of the issuer signature on $id||p$ such that p permits the message and id has been correctly encrypted to the given ciphertext. The generated signature contains the ciphertext in addition to the resulting NIZK proof. To trace a message to its original signer, the opener decrypts the ciphertext using its decryption key to reveal the signer's identity. Although Xu *et al.*'s proposal provides traceability, it does not protect against frameability because the issuer generates the signing keys of the scheme users. Moreover, attaching user identities to the policy seems counter-intuitive to the original goal of PBS schemes, where the signing rights are granted to users

who have access to a policy key which allows them to sign messages that conform to the policy. Consequently, the issuer has to issue multiple signing keys to each scheme user to include their identities for the same policy. According to Xu *et al.* [96], a direct consequence of such an approach for traceability, is that the proposed scheme does not support policy key delegation because the policy is tied to a specific identity. More precisely, if a key holder delegates their key in the form $p' = id||p_1||p_2$ the signature generated with p' will always be traced back to the original key holder id.

2.5 Extended Sanitizable Signatures Schemes (E3S)

Since original sanitizable signature schemes give the sanitizer full power over the modification of the admissible blocks, E3S aim to restrict that sanitizer's choice of modifications. E3S schemes are introduced by Klonowski and Lauks [65] and later enhanced in the work of Canard and Jambert [35] where different extensions for sanitizable signature schemes are proposed; such as a scheme that limits possible modification of an admissible block to a specific set (LimitSet), a scheme that forces the same changes on different admissible blocks (EnforceModif), a scheme that limits the number of modifications of admissible blocks (LimitNbModif) and a scheme that limits the number of versions the sanitizer can produce from an original signed message (LimitNbSanit). Later Derler and Slamanig [45] revised the privacy definition of E3S schemes and proposed a black-box construction of an efficient and secure E3S scheme that supports the *LimitSet* extension. Although these attempts mainly aim to enforce a certain modification policy on the sanitizable message, they did not provide the signer with fine-grained control over the modification policy since each modification policy requires a different scheme. Moreover, all the proposed schemes

are defined in a single sanitizer setting and require an interaction between the signer and sanitizer prior to signature generation and sometimes force the sanitizer to be online during such a phase [35, 65]. In addition, if the sanitizer violates the preset modification policy of the scheme, the key of the sanitizer is leaked publicly which may compromise accountability of previously signed messages [24]. Furthermore, all of these schemes are built over the concept of chameleon hash (a trapdoor collision-resistant hash function [69]) which limits their ability to provide unlinkability [5]. In an attempt to provide an unlinkable E3S scheme, Bossuat and Bultel [24], and Zhan *et. al.* [98] have proposed LimitNbModif E3S schemes that provide unlinkability and invisibility (the verifier cannot learn what are the admissible blocks from a signature) and the latter allows the signer to set sanitization expiration time during signature generation (the sanitizer can only perform sanitization before the expiration time). However, both of the proposed schemes are also defined in a single-sanitizer setting and lack signer fine-grained control over the modification policy. Furthermore, in the work of Zhan *et. al.* [98] The sanitizer's key is generated by some trusted authority which enables frameability.

2.6 Fair Exchange

Fair exchange is a critical concept in digital/physical trading transactions, ensuring that two parties can exchange assets, information, or services in a way that guarantees fairness, even in the absence of trust. The core principle of fair exchange is that either both parties fulfill their obligations or neither does, preventing situations where one party could exploit the other by receiving goods or services without providing payment, or vice versa. This problem is especially pronounced in online or decentralized environments where there is no central

authority to enforce contracts or mediate disputes.

Fair exchange protocols address these challenges by ensuring that assets or information are exchanged simultaneously, or that one party cannot gain an unfair advantage if the other party aborts or behaves maliciously. Traditional fair exchange solutions often rely on trusted third parties (TTPs) to oversee and enforce the terms of the exchange. However, the reliance on TTPs introduces challenges related to trust, cost, and potential centralization risks, which are counter to the principles of decentralization and autonomy sought in many modern systems.

Cryptography and blockchain technology advancements have enabled the development of fair exchange protocols that eliminate the need for TTPs. These protocols often leverage mechanisms like cryptographic commitments, zero-knowledge proofs, and smart contracts. Cryptographic commitments ensure that one party cannot alter the terms of the exchange once committed, while zero-knowledge proofs allow one party to prove the validity of a transaction or asset without revealing sensitive information. Smart contracts, deployed on blockchain platforms, can autonomously enforce the conditions of the exchange, ensuring that both parties adhere to the agreed terms.

Using these technologies in fair exchange protocols provides a robust and decentralized solution to the simultaneity problem. These protocols ensure that if one party fails to fulfill their part of the transaction, the other party is not disadvantaged, maintaining trust and fairness without requiring intermediaries.

Although blockchain can be trusted for correctness and availability, it does not inherently guarantee the privacy of transacting parties or transaction data [67]. This necessitates a tailored approach to ensure both privacy and fair exchange in such protocols.

Chapter 3

Cryptographic Building Blocks

This chapter introduces key cryptographic building blocks used as the foundation of various proposed protocols of this thesis. We begin by exploring digital signatures, next, we discuss the Diffie-Hellman key exchange protocol. We then cover symmetric key encryption. Moving beyond traditional cryptographic schemes, we introduce rerandomizable digital signatures (RDS). Additionally, we present traceable attribute-based signatures (TABS). The chapter also delves into simulation-sound extractable non-interactive zero-knowledge proofs (SE-NIZK). Finally, we discuss ciphertext-policy attribute-based encryption (CP-ABE).

Throughout this thesis, we use $i \in \mathbb{I}$ to denote an identity i from the identity universe $\mathbb{I}$ and $\mathbb{S} \subseteq \mathbb{U}$ denote an attribute set $\mathbb{S}$ from the attribute universe $\mathbb{U}$. We use $\lambda \in \mathbb{N}$ to denote our security parameter, then a function $\epsilon(\lambda) : \mathbb{N} \to [0, 1]$ denotes the negligible function if for any $c \in \mathbb{N}$, $c > 0$ there exists $\lambda_c \in \mathbb{N}$ s.t. $\epsilon(\lambda) < \lambda^{-c}$ for all $\lambda > \lambda_c$.

3.1 Digital Signature (DS)

A digital signature scheme is a tuple of six polynomial-time algorithms, Sig $= \{$ppGenSig, KeyGenSig, SignSig, VerifySig$\}$, which are defined as follows.

- ppGenSig. This algorithm outputs the system's public parameters, $pp_{Sig} \leftarrow \mathsf{ppGenSig}(1^\lambda)$ which becomes an implicit input for all other algorithms.

- KeyGenSig. This algorithm outputs the signer public secret key pair, $(pk_{Sig}, sk_{Sig}) \leftarrow \mathsf{KeyGenSig}(pp_{Sig})$.

- SignSig. On input of a message m, this algorithm outputs a signature over m using sk_{Sig}, $\sigma_{Sig} \leftarrow \mathsf{SignSig}(sk_{Sig}, m)$.

- VerifySig. This algorithm verifies σ_{Sig} over m using pk_{Sig}, $\{\top, \perp\} \leftarrow$ VerifySig (pk_{Sig}, m, σ_{Sig}).

The standard security notion of a digital signature scheme is EUF-CMA [1] is defined in Appendix A.1.

3.2 Diffie-Hellman key exchange protocol

The Diffie-Hellman (DH) key exchange protocol enables two parties to establish a shared secret key over an insecure communication channel. A DH exchange protocol consists of three polynomial-time algorithms, DH $= \{$Setup, KeyGen, KeyEx$\}$, defined as follows:

- Setup: This algorithm outputs the public parameters of the protocol, pp, obtained from $\mathsf{Setup}(1^\lambda)$. Here, pp becomes an implicit parameter for all the subsequent algorithms.

- KeyGen: This algorithm is run independently by each of the two parties (Alice and Bob) intending to establish a shared secret key. It computes their corresponding public/private key pairs: $(pk_{DH}^{A}, sk_{DH}^{A})$ and $(pk_{DH}^{B}, sk_{DH}^{B})$, obtained from $\mathsf{KeyGen}(pp)$.

- KeyEx: Upon receiving the other party's public key, each party computes the shared secret key using its own private key and the received public key as follows:

$$ssk_{B}^{A} \leftarrow \mathsf{KeyEx}(pk_{DH}^{B}, sk_{DH}^{A})$$
$$ssk_{A}^{B} \leftarrow \mathsf{KeyEx}(pk_{DH}^{A}, sk_{DH}^{B})$$

  The correctness property of the protocol ensures that ssk_{B}^{A} is indeed equal to ssk_{B}^{A}.

3.3 Symmetric key encryption

Symmetric key encryption is a cryptographic technique that uses the same secret key for both encryption and decryption of data. Asymmetric key encryption scheme typically consists of three algorithms, $\mathsf{Sym} = \{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$, defined as follows:

- KeyGen: This algorithm generates a secret key K used by both the sender and the receiver. The key generation process is denoted as $\mathsf{KeyGen}(1^{\lambda}) \rightarrow K$, where λ represents the security parameter.

- Enc: The encryption algorithm takes a plaintext message m and the secret key K as inputs and produces a ciphertext c. It is represented as $\mathsf{Enc}(m) \rightarrow ct$.

- Dec: The decryption algorithm takes a ciphertext ct and the secret key K as inputs and produces the original plaintext message m. It is represented as $\mathsf{Dec}(ct) \rightarrow m$.

The semantic security of a symmetric encryption scheme informally ensures that only negligible information about the plaintext can be feasibly extracted from the ciphertext.

By combining DH key exchange with symmetric encryption, parties can symmetrically encrypt the plaintext using the established shared secret key, theoretically eliminating the need for the KeyGen function.

3.4 Rerandomizable Digital Signature (RDS)

RDS schemes are digital signature algorithms that allow rerandomizing a signature such that the rerandomized version of the signature is still verifiable under the verification key of the signer [34,76,77,100]. An important property of RDS schemes is that the rerandomized signatures produced using the same signing key on the same message are indistinguishable from a freshly signed one [76].

An RDS scheme is a tuple of five polynomial-time algorithms, RDS = {ppGenRDS, KeyGenRDS, SignRDS, RandomizeRDS, VerifyRDS} which are defined as follows.

- ppGenRDS. This algorithm outputs the public parameters of the scheme, $pp_{RDS} \leftarrow \mathsf{ppGenRDS}(1^{\lambda})$.

- KeyGenRDS. This algorithm generates the signer's secret and public key pair, (sk_{RDS}, pk_{RDS}) $\leftarrow$KeyGenRDS(pp_{RDS}).

- SignRDS. This algorithm generates a digital signature σ_{RDS} on a message m, $\sigma_{RDS} \leftarrow \mathsf{SignRDS}(sk_{RDS}, m)$.

- VerifyRDS. This algorithm verifies the (rerandomized) signature σ_{RDS} over m, $\{\top, \bot\} \leftarrow \mathsf{VerifyRDS}(pk_{RDS}, m, \sigma_{RDS})$.

- RandomizeRDS. This algorithm randomizes the digital signature σ_{RDS} and outputs

$\sigma'_{RDS}, \sigma'_{RDS} \leftarrow \mathsf{RandomizeRDS}(\sigma_{RDS})$.

Some RDS schemes include a $\sigma_{RDS} \leftarrow \mathsf{SignComRDS}(sk_{RDS}, C)$ procedure that enables the signing of a commitment C of a hidden message m such that the resulting σ_{RDS} is verifiable for m [33, 76].

RDS schemes ensure existential unforgeability under chosen message attacks (EUF-CMA) and unlinkability where it is infeasible for adversaries to link a rereandomized version of a signature to its original one. RDS unlinakability also implies the indistinguishability of rereandomized signatures. The formal definitions of both security notions are given in Appendix A.1 below [76, 100].

3.5 Traceable Attribute-Based Signatures (TABS)

Attribute-Based signature (ABS) schemes are probabilistic digital signature schemes in which the produced signature attests a specific claim predicate (Υ) regarding the attributes that the signer possesses rather than the identity of the signer [73]. ABS schemes ensure privacy where the signer's identity is anonymous among all the users who possess a set of attributes that satisfy the claim predicate specified in the signature. Such schemes utilize a trusted entity called the Attribute Authority (AA) to authenticate users' identities and issue their corresponding attributes. Traceable ABS (TABS) schemes are a variant of ABS schemes where tracing a signature to its original signer is supported [47]. In such schemes, tracing could be performed by AA or another tracing authority (TA) [47, 48]. A TABS scheme is a tuple of eight polynomial-time algorithms, $\mathsf{TABS} = \{\mathsf{ppGenTABS}, \mathsf{TAKeyGenTABS}, \mathsf{AAKeyGenTABS}, \mathsf{SignerKeyGenTABS}, \mathsf{SignTABS}, \mathsf{VerifyTABS}, \mathsf{TraceTABS}, \mathsf{JudgeTABS}\}$ which are specified as follows.

- ppGenTABS. This algorithm outputs the public parameters of the scheme pp_{TABS} which also defines both the identity universe $\mathbb{I}$ and the attribute universe $\mathbb{U}$, $pp_{TABS} \leftarrow \mathsf{ppGenTABS}(1^{\lambda})$.

- TAKeyGenTABS. This algorithm is run by the TA and outputs a tracing key tsk^{TA}_{TABS} for the tracing authority, $tsk^{TA}_{TABS} \leftarrow \mathsf{TAKeyGenTABS}(pp_{TABS})$.

- AAKeyGenTABS. This algorithm is run by the AA to generate its public key and master secret key pair, $(pk_{TABS}, msk^{AA}_{TABS}) \leftarrow \mathsf{AAKeyGenTABS}(pp_{TABS})$.

- SignerKeyGenTABS. This algorithm is run by the AA, on the attribute set $\mathbb{S}_i \subset \mathbb{U}$ and a user identity $i \in \mathbb{I}$ for a specific user and the AA master secret key msk^{AA}_{TABS}. It outputs the user's secret key, $sk^{User,i}_{TABS} \leftarrow \mathsf{SignerKeyGenTABS}(pp_{TABS}, msk^{AA}_{TABS}, i, \mathbb{S}_i)$.

- SignTABS. This algorithm is run by the signer on a message $m \in \{0,1\}^*$ for a claim predicate Υ where the user possesses a set of attributes $\mathbb{S}'_i \subseteq \mathbb{S}_i$ satisfying the claim predicate Υ, i.e., $\Upsilon(\mathbb{S}'_i) = 1$. It outputs a signature, $\sigma_{TABS} \leftarrow \mathsf{SignTABS}(pp_{TABS}, sk^{User,i}_{TABS}, m, \Upsilon)$.

- VerifyTABS. This algorithm verifies the signature σ_{TABS} over m with respect to a claim predicate Υ, $\{\top, \bot\} \leftarrow \mathsf{VerifyTABS}(pp_{TABS}, pk_{TABS}, m, \sigma_{TABS}, \Upsilon)$.

- TraceTABS. The TA runs this algorithm to trace a signature tuple $(m, \sigma_{TABS}, \Upsilon)$ to its actual signer. It outputs the identity of the signer along with NIZK proof π, attesting to this claim, $(i, \pi) \leftarrow \mathsf{TraceTABS}(tsk^{TA}_{TABS}, m, \sigma_{TABS}, \Upsilon)$.

- JudgeTABS. This algorithm outputs true if it verifies that π proves that i is the identity of the signer who produced σ_{TABS} on m,
$\{\top, \bot\} \leftarrow \mathsf{JudgeTABS}(pp_{TABS}, pk_{TABS}, m, \sigma_{TABS}, \Upsilon, i, \pi)$.

The security definitions for unforgeability, privacy, traceability, and non-frameability are defined in Appendix A.1 and in [53].

3.6 Simulation-sound Extractable NIZK (SE-NIZK)

A SE-NIZK system enables a prover with a witness w to prove non-interactively the truthfulness of a statement x to a verifier without conveying why [57]. For x in an $\mathbb{NP}$-language $\mathcal{L}$ such that (x,w) in a relation $\mathbb{R}$ associated with $\mathcal{L}$, a SE-NIZK is a tuple of six polynomial-time algorithms, $\mathsf{NIZK} = \{\mathsf{SetupNIZK}, \mathsf{SimSetupNIZK}, \mathsf{ProveNIZK}, \mathsf{SimProveNIZK}, \mathsf{VerifyNIZK}, \mathsf{ExtrNIZK}\}$, which are defined as follows.

- SetupNIZK. This algorithm outputs the common reference string of the system, $crs \leftarrow \mathsf{SetupNIZK}(1^\lambda)$.

- SimSetupNIZK. This algorithm outputs a simulated crs and a trapdoor, $(crs, tr_{NIZK}) \leftarrow \mathsf{SimSetupNIZK}(1^\lambda)$.

-ProveNIZK. On input of crs, $x \in \{0,1\}^*$ and $w \in \{0,1\}^*$ for the relation $\mathbb{R}$, this algorithm outputs a proof π_{NIZK}, $\pi_{NIZK} \leftarrow \mathsf{ProveNIZK}(crs, x, w)$.

- SimProveNIZK. On input of crs, $x \in \{0,1\}^*$ and the trapdoor tr_{NIZK}, this algorithm outputs a proof π_{NIZK}, $\pi_{NIZK} \leftarrow \mathsf{SimProveNIZK}(crs, x, tr_{NIZK})$.

- VerifyNIZK. This algorithm verifies the proof π_{NIZK} on x. $\{\top, \bot\} \leftarrow \mathsf{VerifyNIZK}(crs,$ $x, \pi_{NIZK})$.

- ExtrNIZK. This procedure extracts w from a π_{NIZK} using tr_{NIZK}, $w \leftarrow \mathsf{ExtrNIZK}\ (crs,$ $x, \pi_{NIZK})$.

An SE-NIZK scheme ensures zero-knowledge which ensures a negligible success of an adversary that can distinguish between a proof for a statement x using a witness w from a

simulated one. A SE-NIZK scheme also provides simulation-extractability which implies that it is hard for an adversary to output a verifiable proof for a statement x using a witness w such that $\mathbb{R}(x, w) = 0$. The formal definitions of such security notions are given in Appendix A.1 and in [12].

3.7 Ciphertext-Policy Attribute-Based Encryption (CP-ABE)

Compared to the all-or-nothing approach of public-key encryption, attribute-based encryption (ABE) offers a fine-grained level of control over encrypted data [87]. In ciphertext-policy ABE (CP-ABE) schemes [55], access policies are linked to ciphertexts, while keys are associated with sets of attributes that the user possesses. For a key to retrieve a plaintext, the set of attributes must satisfy the access policy included in the ciphertext. CP-ABE is a tuple of four polynomial-time algorithms CP-ABE = {SetupABE, EncryptABE, KeyGenABE, DecryptABE} which are defined as follows.

- SetupABE. On the input of a security parameter λ, This algorithm outputs the public key and master secret key of the scheme (pk_{ABE}, msk_{ABE}), and it also defines the attribute universe $\mathbb{U}$, $(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^{\lambda})$.

- EncryptABE. On input of the public key pk_{ABE}, an access structure $\mathbb{A}$, and a message m, it outputs a ciphertext C, $C \leftarrow \mathsf{EncryptABE}(pk_{ABE}, \mathbb{A}, m)$.

- KeyGenABE. This algorithm is run by the entity holding msk_{ABE} which on the input of a set of attributes $\mathbb{S}$, outputs a secret key $sk_{ABE}^{\mathbb{S}}$ associated with such a set of attributes, $sk_{ABE}^{\mathbb{S}} \leftarrow \mathsf{KeyGenABE}(msk_{ABE}, \mathbb{S})$.

- DecryptABE. On input of the public key pk_{ABE}, a ciphertext C, and a secret key $sk_{ABE}^{\mathbb{S}}$, it outputs a message m if $\mathbb{A}(\mathbb{S}) = 1$. Otherwise, it outputs $\perp$, $m \leftarrow$ DecryptABE (pk_{ABE},

$sk^{\mathbb{S}}_{ABE}, C$).

IND-CCA2 is a crucial security property of the CP-ABE scheme. The formal definition of such a security notion, its associated experiment, and security oracles are given in Appendix A.1.

3.8 Hashchain Micropayments Scheme

Rivest and Shamir introduced a micropayment scheme based on hash chains [83]. The core concept involves the payer generating a sequence of “paywords" (hash values) using a one-way hash function for the payee. Subsequently, the payer progressively discloses each payword to the payee to facilitate micropayments. Although this method offers flexibility to accommodate larger value transactions, in the settlement process, a trusted third party, typically a broker, should be involved. The broker assumes responsibility for transferring micropayments to the payee and returning any remaining funds to the payer. In the typical protocol, the buyer creates the payword chain in reverse order by selecting the last payword h_N randomly and then computing $h_i = \mathsf{H}(h_{i+1})$ for $i = N-1, N-2, \ldots, 0$, where h_0 is the root of the payword chain and is a payword itself. The buyer sends a digitally signed h_0 to the seller. After authenticating the buyer, the seller gradually provides the service to the buyer, who gradually releases (h_i, i) to the seller. At the end of the trading session, the seller reports the highest indexed payment (h_l, l) to the trusted third-party broker along with the digitally signed h_0, who charges the buyer in accordance with (h_l, l).

Chapter 4

Unlinkable Policy-Based Sanitizable Signatures

In this chapter, we propose an Unlinkable Policy-Based Sanitizable Signature Scheme (UP3S), a novel approach that enhances privacy and security in Policy-Based Sanitizable Signatures Schemes (PB3S). This work is published in CT-RSA 2023 [5].

UP3S enables a signer to grant sanitization rights for a specific document to sanitizers who satisfy a predefined policy. A key feature of UP3S is unlinkability, ensuring that different sanitized versions of the same document cannot be associated with one another or traced back to the original document. Furthermore, UP3S is designed for efficiency such that for a given sanitization policy, the system setup is executed once. Additionally, the signature size remains fixed, irrespective of the number of sanitizations performed.

The black-box construction of UP3S (Section 4.1) builds upon two cryptographic primitives:

- A rerandomizable digital signature scheme (Section 3.4).

- A traceable attribute-based signature (TABS) scheme (Section 3.5)

Compared to P3S [89], the only other policy-based sanitizable signature scheme in the literature, UP3S introduces unlinkability, eliminates the need to share new secrets with future sanitizers before each sanitization, and maintains a fixed signature size for a given sanitization policy.

We formally define and prove the security properties of UP3S in Sections 4.2 and 4.4, covering key notions such as unlinkability, unforgeability, immutability, transparency, privacy, and accountability. In Section 4.5, we present an instantiation of UP3S based on the Pointcheval-Sanders rerandomizable signature scheme [76] and the DTABS traceable attribute-based signature scheme [53]. We also analyze the efficiency of this instantiation in the same section. Finally, Section 4.6 compares UP3S with P3S in terms of features, scalability, and security models.

Throughout this chapter we use m_i to denotes a message block in a message $m = (m_1, m_2, \ldots, m_l) \in \mathbb{M}^l$, and the variable $adm = (\{A \subseteq \{1, \ldots, l\}\}, l)$ specifies the set of indices A of the modifiable blocks over m which contains l blocks each of size n bits. We use $\mathsf{Adm}(m) = 1$ if adm is valid with respect to m, i.e., it contains a subset A of $\{0, ..., l\}$ and m contains exactly l blocks. We use m_{adm} to denote the list of blocks in m which are admissible with respect to adm. We denote the list of blocks in m which are not admissible under adm with $m_{!adm}$. The function $m' \leftarrow \mathsf{MoD}(m, adm, mod)$ is used to modify the message m by applying the modifications mod on the admissible block(s) adm and outputs the modified message m', where mod is a set that contains the tuple (i, m_i, m'_i) for $i \in adm$. Furthermore, we write $\mathsf{CheckMoD}(m, adm) = 1$, if adm is valid with respect to m, i.e., $\mathsf{Adm}(m) = 1$ and the blocks indices to be modified in a message m are contained in adm as admissible. For each message m, there is one signer and one or more sanitizer(s) who

can sanitize m by running $m' \leftarrow \mathsf{MoD}(m, adm, mod)$ and generating a valid sanitized signature on m' depending on their attributes. Finally, to denote that an attribute set $\mathbb{S}$ satisfies a monotone access structure predicate Υ (see Def. 1), we use $\Upsilon(\mathbb{S}) = 1$.

Definition 1. *(Access Structure [17]). Let $\mathbb{U}$ denote the universe of attributes. A collection $\mathbb{A} \in 2^{\mathbb{U}} \setminus \{0\}$ of a non-empty set is an access structure on $\mathbb{U}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets. The collection $\mathbb{A}$ is called monotone if $\forall B, C \in \mathbb{U}$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$.*

4.1 UP3S Black-box Construction

The main idea behind the proposed construction is that after signers and sanitizers acquire their respective secret keys, the signer of a given message defines a policy Υ that controls the sanitization rights of such a message, i.e., any sanitizer who possesses an attribute-set satisfying Υ is able to generate a sanitized version of such a message without interaction with the AA or the signer. Formally, UP3S scheme is a tuple of nine polynomial-time algorithms, UP3S = {ParGenUP3S, SetupUP3S, KGenSignUP3S, KGenSanUP3S, SignUP3S, SanitizeUP3S, VerifyUP3S, ProveUP3S, JudgeUP3S}. The specifications of the aforementioned algorithms are as follows:

- ParGenUP3S. This algorithm returns the scheme's public parameters which become implicit input for all UP3S algorithms. It also defines the identity universe $\mathbb{I}$ and the attribute universe $\mathbb{U}$. $pp_{UP3S} \leftarrow \mathsf{ParGenUP3S}(1^{\lambda})$

- SetupUP3S. This algorithm outputs the global public key pk_{UP3S} and the master secret key sk_{UP3S} of the scheme. $(pk_{UP3S}, sk_{UP3S}) \leftarrow \mathsf{SetupUP3S}(pp_{UP3S})$

- KGenSignUP3S. This algorithm generates the public-secret key pairs of a signer with identity $i_{Sign} \in \mathcal{I}$ who holds an attribute-set $\mathbb{S}_{Sign} \subset \mathbb{U}$.

$(pk_{UP3S}^{Sign}, sk_{UP3S}^{Sign}) \leftarrow \mathsf{KGenSignUP3S}(sk_{UP3S}, i_{Sign}, \mathbb{S}_{Sign})$

- KGenSanUP3S. This algorithm generates the secret key of a sanitizer with identity $i_{San} \in \mathcal{I}$ who holds an attribute-set $\mathbb{S}_{San} \subset \mathbb{U}$.

$sk_{UP3S}^{San} \leftarrow \mathsf{KGenSanUP3S}(sk_{UP3S}, i_{San}, \mathbb{S}_{San,i})$

- SignUP3S. This algorithm generates a sanitizable signature σ_m using the signer's key sk_{UP3S}^{Sign} on a message m, given the set of indices of the modifiable blocks adm, a predicate Υ, and the attribute-set of possible future sanitizer(s) $\mathbb{S}_{PSan} \subseteq \mathbb{U}$. $(m, \sigma_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}(pk_{UP3S}, sk_{UP3S}^{Sign}, m, adm, \mathbb{S}_{PSan})$

- VerifyUP3S. This algorithm verifies a signature σ_m on a message m, a set of indices of the modifiable blocks adm and a predicate Υ, using the scheme's public key pk_{UP3S} and the signer's public key pk_{UP3S}^{Sign}.

$\{\top, \bot\} \leftarrow \mathsf{VerifyUP3S}(pk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon)$

- SanitizeUP3S. This algorithm generates a sanitized signature σ'_m using the sanitizer secret key sk_{UP3S}^{San} on a signature σ_m, the original message m which is modified to $m' \leftarrow MoD(m, adm, mod)$, the set of indices of the modifiable blocks adm, a predicate Υ, the scheme public key pk_{UP3S}, and the signer public key pk_{UP3S}^{Sign}.

$(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SanitizeUP3S}(pk_{UP3S}, pk_{UP3S}^{Sign}, sk_{UP3S}^{San}, m, \sigma_m, adm, mod, \Upsilon)$

- ProveUP3S. This algorithm outputs the identity i of the signer (resp. sanitizer) of a specific message signature tuple $(pk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon)$ along with a NIZK proof π which proves that i is the signer who generated σ_m on m.

$\{i, \pi\} \leftarrow \mathsf{ProveUP3S}\ (pk_{UP3S}, sk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon)$

- JudgeUP3S. This algorithm verifies the proof π on a specific message signature tuple $(m, \sigma_m, adm, \Upsilon)$ and an identity i.

$\{\top, \bot\} \leftarrow \mathsf{JudgeUP3S}(pk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon, i, \pi)$

UP3S Correctness. For the correctness of UP3S, we require that, for all $\lambda \in \mathbb{N}$, for all $pp_{UP3S} \leftarrow \mathsf{ParGenUP3S}(1^\lambda)$, for all $(pk_{UP3S}, sk_{UP3S}) \leftarrow \mathsf{SetupUP3S}\ (pp_{UP3S})$, for all $i_{Sign} \in \mathbb{I}$, for all $\mathbb{S}_{Sign} \subseteq \mathbb{U}$, for all $(sk_{UP3S}^{Sign}) \leftarrow \mathsf{KGenSignUP3S}\ (sk_{UP3S}, i_{Sign}, \mathbb{S}_{Sign})$, for all $l \in \mathbb{N}$, for all $m \in \mathbb{M}^l$, for all $\mathbb{S}_{PSan} \in \mathbb{U}$, for all $\Upsilon \in 2^{\mathbb{U}} \mid \Upsilon(\mathbb{S}_{Sign}) = 1$, for all $adm = (\{A \subseteq \{1, \ldots, l\}\}, l)$ such that $\mathsf{Adm}(m) = 1$, for all $(m, \sigma_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}$ $(pk_{UP3S}, sk_{UP3S}^{Sign}, m, adm, \mathbb{S}_{PSan})$, for all $i_{San} \in \mathbb{I}$, for all $\mathbb{S}_{San} \subseteq \mathbb{U} \mid \Upsilon(\mathbb{S}_{San}) = 1$, for all $sk_{UP3S}^{San} \leftarrow \mathsf{KGenSanUP3S}\ (sk_{UP3S}, i_{San}, \mathbb{S}_{San,i})$, for all $mod = \{mod_i\} \mid mod_i = (i, m_i, m_i') \forall i \in adm$, for all $m' \leftarrow \mathsf{MoD}(m, adm, mod) \mid \mathsf{CheckMoD}(m, adm) = 1$, and for all $(m', \sigma_m', adm, \Upsilon) \leftarrow \mathsf{SanitizeUP3S}\ (pk_{UP3S}, pk_{UP3S}^{Sign}, sk_{UP3S}^{San}, m, \sigma_m, adm, mod, \Upsilon)$, we have $\top = \mathsf{VerifyUP3S}\ (pk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon)$ and $\top = \mathsf{VerifyUP3S}(pk_{UP3S}, pk_{UP3S}^{Sign}, m', \sigma_m', adm, \Upsilon)$.

Furthermore, for all $\{i, \pi\} \leftarrow \mathsf{ProveUP3S}(pk_{UP3S}, sk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon)$, and $\{i', \pi'\}$ $\leftarrow \mathsf{ProveUP3S}\ (pk_{UP3S}, sk_{UP3S}, pk_{UP3S}^{Sign}, m', \sigma_m', adm, \Upsilon)$, we have $\top = \mathsf{JudgeUP3S}$ $(pk_{UP3S}, pk_{UP3S}^{Sign},$

$m, \sigma_m, adm, \Upsilon, i, \pi)$ and $\top = \mathsf{JudgeUP3S}(pk_{UP3S}, pk_{UP3S}^{Sign}, m', \sigma_m', adm, \Upsilon, i', \pi')$.

4.2 UP3S Security Definitions

In what follows, we define the required security notion of UP3S. We use the same notations as in [26, 28, 89] for ease of readability. The oracles used in the security experiments are

defined in Fig. 4.1. All the security experiments are initialized by running the following setup and key generation procedures.

$$pp_{UP3S} \leftarrow \mathsf{ParGenUP3S}(1^{\lambda})$$
$$(pk_{UP3S}, sk_{UP3S}) \leftarrow \mathsf{SetupUP3S}()$$
$$(sk^{Sign}_{UP3S}, pk^{Sign}_{UP3S}) \leftarrow \mathsf{KGenSignUP3S}(sk_{UP3S}, i_{Sign}, \mathbb{S}_{Sign,i})$$
$$(sk^{San}_{UP3S}) \leftarrow \mathsf{KGenSanUP3S}(sk_{UP3S}, i_{San}, \mathbb{S}_{San,i})$$

The $\mathcal{O}$SignUP3S, $\mathcal{O}$SanitUP3S, $\mathcal{O}$ProveUP3S, $\mathcal{O}$LoRSanitUP3S, $\mathcal{O}$LoRSignSanitUP3S, and $\mathcal{O}$Sign-or-SanitUP3S oracles are implicitly initialized with the secrets sk^{Sign}_{UP3S}, sk^{San}_{UP3S}, sk_{UP3S}, sk^{San}_{UP3S}, (sk^{Sign}_{UP3S}, sk^{San}_{UP3S}), and (sk^{Sign}_{UP3S}, sk^{San}_{UP3S}), respectively.

Moreover, $\mathcal{O}$LoRSanitUP3S, $\mathcal{O}$LoRSignSanitUP3S, and $\mathcal{O}$Sign-or-SanitUP3S oracles are further initialized with a secret bit b that is randomly chosen in the experiments, thus we pass it as a secret input after it gets selected. Note that the attribute sets $\mathbb{S}_{Sign,i}$ and $\mathbb{S}_{San,i}$ used in the experiments initialization are selected such that $\Upsilon(\mathbb{S}_{Sign,i}) = 1$ and $\Upsilon(\mathbb{S}_{San,i}) = 1$ for those oracles queried by the adversary with any Υ.

Unlinkability. Unlinkability is defined using the experiment in Fig 4.2, where the adversary has access to left-or-right sanitization oracle $\mathcal{O}$LoRSanitUP3S (see Fig. 4.1) among other oracles. The adversary inputs two sanitizable messages-signature pairs $\{(m_0, \sigma_{m0}),(m_1, \sigma_{m1})\}$ along with their modifications to $\mathcal{O}$LoRSanitUP3S, the oracle is initialized with a secret random bit '$b \in \{0, 1\}$'. Depending on 'b', the oracle outputs a sanitized signature of either the left or right input message signature pair. The adversary wins if it could determine which pair is used in the sanitization process with probability better than the random guess. The adversary is restricted to inputting two messages such that their modified outputs are the same $m'_0 = m'_1$ to prevent linking a sanitized message to its original source. To achieve such a re-

$\mathcal{O}\mathsf{SignUP3S}(pk'_{UP3S}, m, adm, \mathbb{S}_{PSan})$

if $pk'_{UP3S} = pk_{UP3S}$
  $(m, \sigma_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}(pk_{UP3S}, sk^{Sign}_{UP3S}, m, adm, \mathbb{S}_{PSan})$
  $\mathcal{M} = \mathcal{M} \cup \{m, adm, \sigma_m, \Upsilon\}$
  $\mathcal{S} = \mathcal{S} \cup \{\mathbb{S}_{PSan}\}$
  return $(m, \sigma_m, adm, \Upsilon)$
return 0

$\mathcal{O}\mathsf{SanitUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, mod, \Upsilon)$

$(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SanitizeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, sk^{San}_{UP3S}, m, \sigma_m, adm, mod, \Upsilon)$
$\mathcal{L} = \mathcal{L} \cup \{m', \sigma'_m, adm, \Upsilon\}$
return $(m', \sigma'_m, adm, \Upsilon)$

$\mathcal{O}\mathsf{ProveUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon)$

if $(m, \sigma_m) \notin \mathcal{T}$
  return $(i, \pi) \leftarrow \mathsf{ProveUP3S}(pk_{UP3S}, sk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon)$
return 0

$\mathcal{O}\mathsf{LoRSanitUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m_0, \sigma_{m0}, adm_0, \Upsilon_0, mod_0, m_1, \sigma_{m1}, adm_1, \Upsilon_1, mod_1)$

if $\mathsf{MoD}(m_0, adm_0, mod_0) = \mathsf{MoD}(m_1, adm_1, mod_1) \wedge \Upsilon_0 = \Upsilon_1 \wedge adm_0 = adm_1$
  if $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m_0, \sigma_{m0}, adm_0, \Upsilon_0) \wedge \mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m_1, \sigma_{m1}, adm_1, \Upsilon_1)$
    return $(m'_b, \sigma'_{mb}, adm_b, \Upsilon_b) \leftarrow \mathsf{SanitizeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, sk^{San}_{UP3S}, m_b, \sigma_{mb}, adm_b, mod_b, \Upsilon_b)$
return 0

$\mathcal{O}\mathsf{LoRSignSanitUP3S}(m_0, adm_0, \Upsilon_0, mod_0, \mathbb{S}_{PSan,0}, m_1, adm_1, \Upsilon_1, mod_1, \mathbb{S}_{PSan,1})$

if $\mathsf{MoD}(m_0, adm_0, mod_0) = \mathsf{MoD}(m_1, adm_1, mod_1) \wedge \Upsilon_0 = \Upsilon_1 \wedge adm_0 = adm_1 \wedge \mathbb{S}_{PSan,0} = \mathbb{S}_{PSan,1}$
  $(m_b, \sigma_{mb}, adm_b, \Upsilon_b) \leftarrow \mathsf{SignUP3S}(pk_{UP3S}, sk^{Sign}_{UP3S}, m_b, adm_b, \mathbb{S}_{PSan,b})$
  return $(m'_b, \sigma'_{mb}, adm_b, \Upsilon_b) \leftarrow \mathsf{SanitizeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, sk^{San}_{UP3S}, m_b, \sigma_{mb}, adm_b, mod_b, \Upsilon_b)$
return 0

$\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitUP3S}(m, adm, \mathbb{S}_{PSan}, mod)$

if $b = 0$
  $m' \leftarrow \mathsf{MoD}(m, mod, adm)$
  $(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}(pk_{UP3S}, sk^{Sign}_{UP3S}, m', adm, \mathbb{S}_{PSan})$
else $(m, \sigma_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}(pk_{UP3S}, sk^{Sign}_{UP3S}, m, adm, \mathbb{S}_{PSan})$
  $(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SanitizeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, sk^{San}_{UP3S}, m, \sigma_m, adm, mod, \Upsilon)$
$\mathcal{T} = \mathcal{T} \cup \{m'_b, \sigma'_{mb}\}$
return $(m', \sigma'_m, adm, \Upsilon)$

Figure 4.1: UP3S security experiments oracles.

striction, the adversary must input two messages with identical fixed parts, $m_{0_{!adm}} = m_{1_{!adm}}$ and the two messages' admissible blocks indices must be the same, i.e., $adm_0 = adm_1$. To match UP3S's policy-based expressiveness, we further restrict the adversary to input two messages that could be sanitized under the same predicate, i.e., $\Upsilon_1 = \Upsilon_2$. Note that, unlike group signature schemes where unlinkability is defined as the infeasibility to link two messages and their signatures to the same signer [13], in sanitizable signature, unlinkability is

defined as the infeasibility to link signatures of two or more sanitized versions of a message to the same source message [28].

$\mathbf{Exp}^{Unlinkability}_{\mathcal{A},UP3S}(\lambda)$

$b \xleftarrow{\$} \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignUP3S}(.),\mathcal{O}\mathsf{SanitUP3S}(.),\mathcal{O}\mathsf{ProveUP3S}(.),\mathcal{O}\mathsf{LoRSanitUP3S}(.,b)}(pk_{UP3S}, pk^{Sign}_{UP3S})$
if $a = b$
  return 1
return 0

Figure 4.2: UP3S unlinkability experiment.

Definition 2. *(Unlinkability) UP3S scheme is unlinkable if for any PPT adversary $\mathcal{A}$,*

$$\left| \Pr[\boldsymbol{Exp}^{Unlinkability}_{\mathcal{A},UP3S}(\lambda) = 1] - \frac{1}{2} \right| \leq \epsilon(\lambda)$$

, where the unlinkability experiment is described in Fig. 4.2.

Transparency. This notion requires that no adversary can distinguish between sanitizable signatures created by the signer or the sanitizer. Transparency is modeled by the experiment in Fig. 4.3 in which adversary $\mathcal{A}$ has access to $\mathcal{O}$SignUP3S, $\mathcal{O}$SanitUP3S , and $\mathcal{O}$ProveUP3S. At the end, $\mathcal{A}$ queries $\mathcal{O}$Sign-or-SanitUP3S with a message m, a modification mod, possible sanitizers attribute set $\mathbb{S}_{PSan}$ and the set of indices of the modifiable blocks adm. $\mathcal{O}$Sign-or-SanitUP3S which is initialized by a secret random bit b, outputs the signature tuple $(m', \sigma'_m, adm, \Upsilon)$ as follows.

- For $b = 0$, $m' \leftarrow \mathsf{MoD}(m, adm, mod)$, $\mathcal{O}$Sign-or-SanitUP3S runs the signing algorithm to create $(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}(\, pk_{UP3S}\, , sk^{Sign}_{UP3S}, m', adm, \mathbb{S}_{PSan})$ and outputs the message signature pair $(m', \sigma'_m, adm, \Upsilon)$.
- For $b = 1$, $\mathcal{O}$Sign-or-SanitUP3S runs the signing algorithm to create $(m, \sigma_m, adm, \Upsilon) \leftarrow$

$\mathsf{SignUP3S}(pk_{UP3S}, sk_{UP3S}^{Sign}, m, adm, \mathbb{S}_{PSan})$, further sanitizes the message $m' \leftarrow \mathsf{MoD}$ (m, adm, mod) and returns $(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SanitizeUP3S}$ $(pk_{UP3S}$,pk_{UP3S}^{Sign},sk_{UP3S}^{San} ,m,σ_m,adm,mod,$\Upsilon)$.

$\mathcal{A}$ wins if it can guess b with probability better than the random guess. Note that access to $\mathcal{O}\mathsf{ProveUP3S}$ oracle is restricted to (m, σ_m) pairs that have never been queried to $\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitUP3S}$ oracle.

Definition 3. *(Transparency) UP3S is transparent if for any PPT adversary $\mathcal{A}$,*

$$\left| \mathsf{Pr}[\boldsymbol{Exp}_{\mathcal{A},UP3S}^{Transparency}(\lambda) = 1] - \frac{1}{2} \right| \leq \epsilon(\lambda)$$

where the transparency experiment is defined in Fig 4.3

$\mathbf{Exp}_{\mathcal{A},UP3S}^{Transparency}(\lambda)$

$b \xleftarrow{\$} \{0,1\}, \mathcal{T} = \{\}$
$a \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignUP3S}(.),\mathsf{OSanitUP3S}(.),\mathcal{O}\mathsf{ProveUP3S}(.),\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitUP3S}(.,b)}(pk_{UP3S}, pk_{UP3S}^{Sign})$
if $a = b$
  return 1
else return 0

Figure 4.3: UP3S transparency experiment.

Immutability. This security notion implies that no adversary with no access to the signer's secret key sk_{UP3S}^{Sign} can alter inadmissible blocks. In UP3S, we extend the immutability definition to capture adversarial changes in the predefined signing predicate Υ, i.e., no adversary can change the signing predicate defined by the original signer of a message. Immutability is modeled by the security experiment defined in Fig. 4.4 in which adversary $\mathcal{A}$ has access to $\mathcal{O}\mathsf{SignUP3S}$, and $\mathcal{O}\mathsf{SanitUP3S}$ oracles. The signing oracle $\mathcal{O}\mathsf{SignUP3S}$ is initialized with sk_{UP3S}^{Sign} for the attribute set $\mathbb{S}_{Sign}$. $\mathcal{A}$ queries $\mathcal{O}\mathsf{SignUP3S}$ by $m_i, adm_i, \mathbb{S}_{PSan,i}$ for $i = 1, 2, ..., q$, the signing oracle outputs the signature tuple $(m_i, \sigma_{mi}, adm_i, \Upsilon_i)$ where

the predicate Υ_i is satisfied by $\mathbb{S}'_{Sign} \subseteq \mathbb{S}_{Sign}$ and by $\mathbb{S}_{PSan,i}$. On the other hand, The sanitization oracle $\mathcal{O}$SanitUP3S is initialized with sk^{San}_{UP3S} for the attribute set $\mathbb{S}_{San}$. $\mathcal{A}$ queries $\mathcal{O}$SanitUP3S by $(m_j, \sigma_{mj}, adm_j, mod_j, \Upsilon_j)$ for $j = 1, 2, ..., p$, the sanitization oracle outputs the signature tuple $(m'_j, \sigma'_{mj}, adm_j, \Upsilon_j)$. The adversary wins if it could generate a verifiable $(\sigma^*_m, m^*, adm^*, \Upsilon^*)$ such that for all $i = 1, 2, ..., q$ (resp. $j = 1, 2, ..., p$), m^* is not valid a modification of any m_i (resp. m_j) under adm_i (resp. adm_j) where $\mathsf{CheckMoD}(m_i, adm_i) = 1$ (resp. $\mathsf{CheckMoD}(m_j, adm_j) = 1$), or m^* is a valid a modification of any m_i (resp. m_j) under adm_i (resp. adm_j) where $\mathsf{CheckMoD}(m_i, adm_i) = 1$ (resp. $\mathsf{CheckMoD}(m_j, adm_j) = 1$) and $\Upsilon^* \neq \Upsilon_i$ (resp. $\Upsilon^* \neq \Upsilon_j$). Note that $\mathcal{A}$ is allowed to query $\mathcal{O}$SanitUP3S oracle to simulate multiple sanitization cases where a sanitized message could be further sanitized by a different sanitizer. The definition considers adversaries who are valid sanitizers trying to alter inadmissible blocks thus the adversary may access some sanitization key $sk^{San,\mathcal{A}}_{UP3S}$ for a predefined attribute set.

Definition 4. *(Immutability) UP3S is an immutable sanitizable signature scheme if for any PPT adversary $\mathcal{A}$,*

$$\mathsf{Pr}[\boldsymbol{Exp}^{Immutability}_{\mathcal{A},UP3S}(\lambda) = 1] \leq \epsilon(\lambda)$$

where the immutability experiment is defined in Fig. 4.4.

Accountability. This security notion implies that if a signer (resp. sanitizer) did not sign (resp. sanitize) a message, then a malicious sanitizer (resp. signer) should not be able to convince the judge to accuse the signer (resp. sanitizer). Accountability is modeled by the security experiment defined in Fig. 4.5, in which adversary $\mathcal{A}$ has access to either sk^{San}_{UP3S} (resp. sk^{Sign}_{UP3S}), in addition to two oracles $\mathcal{O}$SanitUP3S (resp. $\mathcal{O}$SignUP3S) and $\mathcal{O}$ProveUP3S. $\mathcal{A}$ can query $\mathcal{O}$SanitUP3S (resp. $\mathcal{O}$SignUP3S) with $(m_i, \sigma_{mi}, adm_i, mod_i, \Upsilon_i)$ (resp. m_i)

$\mathbf{Exp}^{Immutability}_{\mathcal{A},UP3S}(\lambda)$

$\mathcal{M} = \mathcal{L} = \{\}$
  $(m^*, \sigma^*_m, adm^*, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignUP3S}(.),\mathcal{O}\mathsf{SanitUP3S}(.)}(pk_{UP3S}, pk^{Sign}_{UP3S})$
  if $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*)$
    $(\forall\{m_i, adm_i, \Upsilon_i\} \in \mathcal{M} \wedge \forall\{m_j, adm_j, \Upsilon_j\} \in \mathcal{L})$
      if $(m^* \notin \{\mathsf{MoD}(m_i, adm_i, .)|\mathsf{CheckMoD}(m_i, adm_i) = 1\} \wedge$
        $m^* \notin \{\mathsf{MoD}(m_j, adm_j, .)|\mathsf{CheckMoD}(m_j, adm_j) = 1\})$
      return 1
      elseif $(m^* \in \{\mathsf{MoD}(m_i, adm_i, .)|\mathsf{CheckMoD}(m_i, adm_i) = 1\} \wedge \Upsilon^* \neq \Upsilon_i) \vee$
        $(m^* \in \{\mathsf{MoD}(m_j, adm_j, .)|\mathsf{CheckMoD}(m_j, adm_j) = 1\} \wedge \Upsilon^* \neq \Upsilon_j))$
      return 1
return 0

Figure 4.4: UP3S immutability experiment.

to get $(m'_i, \sigma'_{mi}, adm_i, \Upsilon_i)$ (resp. $(m_i, \sigma_{mi}, adm_i, \Upsilon_i)$) for $i = \{1, 2, ..., q\}$. The adversary wins if it outputs a verifiable message signature pair $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$ where $m^* \notin \{m_1, \ldots, m_q\}$ and the output $\mathcal{O}\mathsf{ProveUP3S}$ oracle on the input of (pk^{Sign}_{UP3S}, m^*, σ^*_m, adm^*, Υ^*) is falsely traced back to i_{Sign} if $\mathcal{A}$ has access to sk^{San}_{UP3S}), or to i_{San} if $\mathcal{A}$ has access to sk^{Sign}_{UP3S}, and such a result is verified by the $\mathsf{JudgeUP3S}$ algorithm.

Definition 5. *(Accountability) UP3S ensures accountability if for any PPT adversary $\mathcal{A}$,*

$$\mathsf{Pr}[\boldsymbol{Exp}^{Accountability}_{\mathcal{A},UP3S}(\lambda) = 1] \leq \epsilon(\lambda)$$

where the accountability experiment is defined in Fig. 4.5

Privacy. This notion implies that it is infeasible to use sanitized signatures to recover information about the sanitized parts of the message. Privacy is defined using an experiment where the adversary inputs two message-modifications tuples (m_0, adm_0, mod_0) and (m_1, adm_1, mod_1) to $\mathcal{O}\mathsf{LoRSignSanitUP3S}$ oracle which is initialized with a secret random bit 'b'. Depending on 'b', the oracle outputs a sanitized signature of either the left or right input message modification tuple. The adversary wins if it could determine which pair

$\mathbf{Exp}^{Accountability}_{\mathcal{A},UP3S}(\lambda)$

$\mathcal{M} = 0, \mathcal{L} = 0$
if $\mathcal{A}$ has sk^{Sign}_{UP3S}
  $(m^*, \sigma^*_m, adm^*, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SanitUP3S}(.),\mathcal{O}\mathsf{ProveUP3S}(.)}(pk_{UP3S}, pk^{Sign}_{UP3S})$
  $(i^*, \pi^*) \leftarrow \mathsf{ProveUP3S}(pk_{UP3S}, sk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*)$
  if $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*) \wedge i^* \neq i_{Sign} \wedge (m^*, \sigma^*_m) \notin (\mathcal{M} \cup \mathcal{L}) \wedge$
    $\top \leftarrow \mathsf{JudgeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*, i^*, \pi^*)$
    return 1
  else return 0
if $\mathcal{A}$ has sk^{San}_{UP3S}
  $(m^*, \sigma^*_m, adm^*, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignUP3S}(.),\mathcal{O}\mathsf{ProveUP3S}(.)}(pk_{UP3S}, pk^{Sign}_{UP3S})$
  $(i^*, \pi^*) \leftarrow \mathsf{ProveUP3S}(pk_{UP3S}, sk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*)$
  if $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*) \wedge i^* \neq i_{San} \wedge (m^*, \sigma^*_m) \notin (\mathcal{M} \cup \mathcal{L}) \wedge$
    $\top \leftarrow \mathsf{JudgeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*, i^*, \pi^*)$
  return 1
return 0

Figure 4.5: UP3S accountability experiment.

is used in the sanitization process with probability better than the random guess. Similar to $\mathcal{O}$LoRSanitUP3S, the adversary must input two messages with identical fixed parts, $m_{0_{!adm}} = m_{1_{!adm}}$, the two messages' admissible policies must be the same, i.e., $adm_0 = adm_1$, and the two messages have to be signed under the same attribute-set of possible future sanitizers, $\mathbb{S}_{PSan,0} = \mathbb{S}_{PSan,1}$.

Definition 6. *(Privacy) UP3S scheme is private if for any PPT adversary $\mathcal{A}$,*

$$|\mathsf{Pr}[\boldsymbol{Exp}^{Privacy}_{\mathcal{A},UP3S}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$$

where the privacy experiment is defined in Fig. 4.6.

Unforgeability This notion implies that an adversary with no access to either the signer or the sanitizer secret keys cannot generate a verifiable signature under honestly generated keys. This also includes the case where the adversary does not possess the required attribute set

$\mathbf{Exp}^{Privacy}_{\mathcal{A},UP3S}(\lambda)$

$b \xleftarrow{\$} \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignUP3S}(.),\mathcal{O}\mathsf{SanitUP3S}(.),\mathcal{O}\mathsf{ProveUP3S}(.),\mathcal{O}\mathsf{LoRSignSanitUP3S}(.,b)}(pk_{UP3S}, pk^{Sign}_{UP3S})$
if $a = b$
  return 1
return 0

Figure 4.6: UP3S privacy experiment.

by the claim predicate to generate such signatures. This must hold even if the adversary has access to additional message signature pairs and the public keys. Unforgeability is modeled by the experiment depicted in Fig. 4.7 in which adversary $\mathcal{A}$ has access to three oracles $\mathcal{O}$SignUP3S, $\mathcal{O}$SanitUP3S , $\mathcal{O}$ProveUP3S and possesses a set of attributes $\mathbb{S}_{\mathcal{A}}$. $\mathcal{A}$ wins if it outputs a verifiable tuple $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$ that has never been queried to $\mathcal{O}$SignUP3S nor $\mathcal{O}$SanitUP3S oracles and the claim predicate Υ^* is not satisfied by $\mathbb{S}_{\mathcal{A}}$.

Definition 7. *(Unforgeability) UP3S scheme is unforgeability if for any PPT adversary $\mathcal{A}$,*

$$|\mathsf{Pr}[\boldsymbol{Exp}^{Unforgeability}_{\mathcal{A},UP3S}(\lambda) = 1]| \leq \epsilon(\lambda)$$

where the unforgeability experiment is defined in Fig. 4.7.

$\mathbf{Exp}^{Unforgeability}_{\mathcal{A},UP3S}(\lambda)$

$\mathcal{M} = \mathcal{L} = \{\}$
$(m^*, \sigma^*_m, adm^*, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignUP3S}(.),\mathcal{O}\mathsf{SanitUP3S}(.),\mathcal{O}\mathsf{ProveUP3S}(.)}(pk_{UP3S}, pk^{Sign}_{UP3S})$
if $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m^*, \sigma^*_m, adm^*, \Upsilon^*) \wedge (m^*, \sigma^*_m) \notin \mathcal{M} \cup \mathcal{L} \wedge \Upsilon^*(\mathbb{S}_{\mathcal{A}}) \neq 1$
  return 1
return 0

Figure 4.7: UP3S unforgeability experiment.

4.3 UP3S Generic Construction

In the generic construction for UP3S, we utilize two main building blocks, a TABS scheme, and an RDS scheme. The generic construction of UP3S scheme is depicted in Fig. 4.8. Once UP3S is initialized, signers and sanitizers can generate their keys using KGenSignUP3S and KGenSanUP3S algorithms. To construct the sanitization policy for a given message, the signer uses their own selective set of attributes ($\mathbb{S}'_{Sign} \subseteq \mathbb{S}_{Sign}$), and that of possible future sanitizers $\mathbb{S}_{PSan}$ to construct a monotone access structure (predicate) Υ. The produced predicate Υ must be satisfied by some of the signer attributes ($\mathbb{S}'_{Sign}$) and should be satisfied by the selected attribute sets of future possible sanitizers $\mathbb{S}_{PSan}$ as well, i.e., ($\Upsilon(\mathbb{S}'_{Sign}) = 1$ and $\Upsilon(\mathbb{S}_{PSan}) = 1$). Thus any scheme user who holds an attribute set $\mathbb{S}''$ that satisfies the claim predicate Υ, i.e, $\Upsilon(\mathbb{S}'') = 1$, can sanitize such a message.

Signing. To sign a given message, the signer uses the SignUP3S algorithm, in which a hash function H is applied on the access structure Υ along with the inadmissible part of the message $m_{\setminus adm}$, and the set of indices of the modifiable blocks adm. The output of H is signed using the RDS scheme with the signer key sk^{Sign}_{RDS} to output a signature σ_{fix}. Next, the full message m is anonymously signed using the TABS scheme under the signer key sk^{Sign}_{TABS} to output $(\sigma_{full}, \Upsilon)$, where σ_{full} attests that the message signer possesses a set of attributes satisfying the sanitization policy, i.e., $\Upsilon(\mathbb{S}'_{Sign}) = 1$. Finally, the signer outputs (σ_m, Υ) as the sanitizable signature over m, where $\sigma_m = (\sigma_{fix}, \sigma_{full})$.

Sanitizing. A sanitizer who holds a set of attributes ($\mathbb{S}'_{PSan} \subseteq \mathbb{S}_{PSan}$) that satisfy the message signature claim predicate i.e $\Upsilon(\mathbb{S}'_{PSan}) = 1$, is authorized to sanitize the admissible part(s) of the message m_{adm} according to adm. The sanitizer first applies the set of modification mod over m to generate the modified version of the message m' such that

$m' = \mathsf{MoD}(m, , adm, mod)$. Then the sanitizer signs m' anonymously using their TABS scheme sanitizer key sk^{San}_{TABS} under the same claim predicate Υ where $\Upsilon(\mathbb{S}'_{PSan}) = 1$ to evaluate σ'_{full}. Finally, the sanitizer rerandomizes the original signature σ_{fix} to produce σ'_{fix}, and outputs (σ'_m, Υ) as the sanitized signature version, where $\sigma'_m = (\sigma'_{fix}, \sigma'_{full})$.

Verifying and tracing. Verifying a message signature pair is straightforward, where σ'_{fix} and σ'_{full} are separately verified with respect to their corresponding verification keys using the VerifyUP3S algorithm. To trace a message signature pair to its original signer, the tracing function of the underlying TABS scheme is utilized in the ProveUP3S algorithm and then the JudgeUP3S algorithm attests whether the output of ProveUP3S is valid or not.

4.4 UP3S Security

It has been proven in [28] that unlinkable sanitizable signature schemes are private. More precisely, Brzuska *et al.* have shown how to convert an adversary against privacy into an adversary against unlinkability. Accordingly, in what follows we prove that UP3S is unlinkable (implies private), accountable, immutable, transparent, and unforgeable sanitizable signature scheme.

Theorem 1. *Given an unlinkable RDS scheme, then the sanitizable signature scheme in Fig. 4.8 is unlinkable.*

Proof. In the UP3S unlinkability experiment in Fig. 4.2, the adversary inputs to $\mathcal{O}$LoRSanitUP3S oracle two valid signature tuples $(m_0, \sigma_{m0}, adm_0, \Upsilon_0, mod_0)$, and $(m_1, \sigma_{m1}, adm_1, \Upsilon_1, mod_1)$ where $adm_0 = adm_1$, $\mathsf{MoD}(m_0, adm_0, mod_0) = \mathsf{MoD}(m_1, adm_1, mod_1)$ and $\Upsilon_0 = \Upsilon_1$. $\mathcal{O}$LoRSanitUP3S oracle outputs $(m'_b, \sigma'_{mb}) \leftarrow$ SanitizeUP3S (pk_{UP3S}, pk^{Sign}_{UP3S}, sk^{San}_{UP3S}, m_b, σ_{mb}, adm_b, mod_b, Υ_b) for $b \xleftarrow{\$} \{0, 1\}$. Recall that $\sigma'_{mb} = (\sigma'_{fix,b}, \sigma'_{full,b})$ where $\sigma'_{fix,b}$

ppGenUP3S. Given a collision-resistant hash function $H : \{0,1\}^* \rightarrow Z_p^*$, run $pp_{TABS} \leftarrow \mathsf{ppGenTABS}(1^\lambda)$, $pp_{RDS} \leftarrow \mathsf{ppGenRDS}(1^\lambda)$. Set $pp_{UP3S} = \{H, pp_{TABS}, pp_{RDS}\}$, where pp_{UP3S} becomes an implicit input for all UP3S algorithms.

$$pp_{UP3S} \leftarrow \mathsf{ppGenUP3S}(1^\lambda)$$

SetupUP3S. Initialize the TABS scheme trusted entities and generate their corresponding keys, $tsk_{TABS}^{TA} \leftarrow \mathsf{TAKeyGenTABS}(pp_{UP3S})$ and $(pk_{TABS}, msk_{TABS}) \leftarrow \mathsf{AAKeyGenTABS}(pp_{UP3S})$. Output UP3S public-secret key pair $(pk_{UP3S}, sk_{UP3S}) = (pk_{TABS}, (msk_{TABS}, tsk_{TABS}^{TA}))$

$$(pk_{UP3S}, sk_{UP3S}) \leftarrow \mathsf{SetupUP3S}(pp_{UP3S})$$

KGenSignUP3S. Let $(sk_{RDS}^{Sign}, pk_{RDS}^{Sign}) \leftarrow \mathsf{keyGenRDS}(pp_{RDS})$ and $sk_{TABS}^{Sign,i} \leftarrow \mathsf{SignerKeyGenTABS}(pp_{TABS}, msk_{TABS}, i_{Sign}, \mathbb{S}_{Sign,i})$. Output $(sk_{UP3S}^{Sign}, pk_{UP3S}^{Sign}) = ((sk_{RDS}^{Sign}, sk_{TABS}^{Sign,i}), pk_{RDS}^{Sign})$.

$$(sk_{UP3S}^{Sign}, pk_{UP3S}^{Sign}) \leftarrow \mathsf{KGenSignUP3S}(sk_{UP3S}, i_{Sign}, \mathbb{S}_{Sign,i})$$

KGenSanUP3S. Let $sk_{TABS}^{San,i} \leftarrow \mathsf{SignerKeyGenTABS}(pp_{TABS}, msk_{TABS}, i_{San}, \mathbb{S}_{San,i})$. Output $sk_{UP3S}^{San} = sk_{TABS}^{San,i}$.

$$sk_{UP3S}^{San} \leftarrow \mathsf{KGenSanUP3S}(sk_{UP3S}, i_{San}, \mathbb{S}_{San,i})$$

SignUP3S. Generate the signing predicate Υ s.t. $\mathbb{S}'_{Sign,i} \subseteq \mathbb{S}_{Sign,i}$ and $\mathbb{S}_{PSan}$, where $\Upsilon(\mathbb{S}'_{Sign,i}) = 1$ and $\Upsilon(\mathbb{S}_{PSan}) = 1$. Generate $\sigma_{fix} \leftarrow \mathsf{SignRDS}(sk_{RDS}^{Sign}, H(pk_{UP3S}||m_{!adm}||adm||\Upsilon))$ and $\sigma_{full} \leftarrow \mathsf{SignTABS}(pp_{TABS}, sk_{TABS}^{Sign,i}, m, \Upsilon)$. Let $\sigma_m = (\sigma_{fix}, \sigma_{full})$, return $(m, \sigma_m, adm, \Upsilon)$.

$$(m, \sigma_m, adm, \Upsilon) \leftarrow \mathsf{SignUP3S}(pk_{UP3S}, sk_{UP3S}^{Sign}, m, adm, \mathbb{S}_{PSan})$$

VerifyUP3S. Check if $\mathsf{Adm}(m) = 1$ and $m_{!adm} \in m$ at the correct positions, otherwise return $\perp$. Let $(\sigma_{fix}, \sigma_{full}) \leftarrow \sigma_m$, if $\mathsf{VerifyRDS}(pk_{RDS}^{Sign}, H(pk_{UP3S}||m_{!adm}||adm||\Upsilon, \sigma_{fix})) \wedge \mathsf{VerifyTABS}(pp_{TABS}, pk_{TABS}, m, \sigma_{full}, \Upsilon)$ return $\top$. Otherwise, return $\perp$.

$$\{\top, \perp\} \leftarrow \mathsf{VerifyUP3S}(pk_{UP3S}, pk_{UP3S}^{Sign}, m, \sigma_m, adm, \Upsilon)$$

SanitizeUP3S. If $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon) = \perp \vee \mathsf{CheckMoD}(m, adm) \neq 1$ return $\perp$. Otherwise, let $(\sigma_{fix}, \sigma_{full}) \leftarrow \sigma_m$, generate $\sigma'_{fix} \leftarrow \mathsf{RerandomizeRDS}(m, \sigma_{fix})$, $m' \leftarrow \mathsf{MoD}(m, adm, mod)$, $\sigma'_{full} \leftarrow \mathsf{SignTABS}(pp_{TABS}, sk^{San,i}_{TABS}, m', \Upsilon)$. Let $\sigma'_m \leftarrow (\sigma'_{fix}, \sigma'_{full})$, return $(m', \sigma'_m, adm, \Upsilon)$.

$$(m', \sigma'_m, adm, \Upsilon) \leftarrow \mathsf{SanitizeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, sk^{San}_{UP3S}, m, \sigma_m, adm, mod, \Upsilon)$$

ProveUP3S. If $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon) = \perp$ return $\perp$. Otherwise, parse σ_{full} from σ_m. Return $(i, \pi) \leftarrow \mathsf{TraceTABS}(tsk^{TA}_{TABS}, m, \sigma_{full}, \Upsilon)$.

$$\{i, \pi\} \leftarrow \mathsf{ProveUP3S}(pk_{UP3S}, sk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon)$$

JudgeUP3S. If $\mathsf{VerifyUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon) = \perp$, return $\perp$. Otherwise, parse σ_{full} from σ_m. Return $\{\top, \perp\} \leftarrow \mathsf{JudgeTABS}(pp_{TABS}, pk_{UP3S}, m, \sigma_{full}, \Upsilon, i, \pi)$.

$$\{\top, \perp\} \leftarrow \mathsf{JudgeUP3S}(pk_{UP3S}, pk^{Sign}_{UP3S}, m, \sigma_m, adm, \Upsilon, i, \pi)$$

Figure 4.8: UP3S generic construction.

is a randomized version of the signer's RDS signature on $H(pk_{UP3S}||m_{!adm,b}||adm_b||\Upsilon_b)$ and $\sigma'_{full,b}$ is the sanitizer's TABS signature on the modified message $m'_b = MoD(m_b, adm_b, mod_b)$. By contradiction, we let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}^{Unlinkability}_{\mathcal{A},UP3S}$, we then show that we can build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the unlinkability of the underlying RDS scheme and win in $\mathbf{Exp}^{Unlinkability}_{\mathcal{B},RDS}$ in Fig. A.2. $\mathcal{B}$ first generates $(sk^B_{TABS}, pk^B_{TABS})$ for attribute set $\mathbb{S}_\mathcal{B} = \mathbb{U}$. To simulate $\mathcal{A}$'s oracles calls, $\mathcal{B}$ answers $\mathcal{A}$'s calls to $\mathcal{O}\mathsf{SignUP3S}$ by constructing the claim predicate Υ such that $\Upsilon(\mathbb{S}_\mathcal{B}) = 1$, calculating $H(pk_{UP3S}||m_{!adm}||adm||\Upsilon)$, and passes $H(pk_{UP3S}||m_{!adm}||adm||\Upsilon)$ to $\mathcal{O}\mathsf{SignRDS}(.)$ to get σ_{fix}, then signs (m) using sk^B_{TABS} to get σ_{full}. To answer $\mathcal{A}$'s calls to $\mathcal{O}\mathsf{SanitizeUP3S}$, $\mathcal{B}$ evaluates σ'_{fix} by rerandomizing σ_{fix}, and calculates $m' = \mathsf{MoD}(m, adm, mod)$, then

signs (m') using sk^B_{TABS} (where $\Upsilon(\mathbb{S}_\mathcal{B} = 1)$) to get σ'_{full}. For $\mathcal{A}$'s calls to $\mathcal{O}$ProveUP3S, $\mathcal{B}$ simply replies with its own identity for all queries where $\Upsilon(\mathbb{S}_\mathcal{B} = 1)$. When $\mathcal{A}$ inputs $(m_0, \sigma_{m0}, adm_0, \Upsilon_0, mod_0)$, and $(m_1, \sigma_{m1}, adm_1, \Upsilon_1, mod_1)$ to $\mathcal{O}$LoRSanitUP3S, $\mathcal{B}$ forwards $(H(pk_{UP3S}||m_{!adm,0}||adm_0||\Upsilon_0), \sigma_{fix,0})$ and $(H(pk_{UP3S}||m_{!adm,1}||adm_1||\Upsilon_1), \sigma_{fix,1})$ to $\mathcal{O}$LoRRDS to obtain the challenge $\sigma'_{fix,b}$. Then $\mathcal{B}$ evaluates $m' = m'_0 \leftarrow \mathsf{MoD}\ (m_0,$ $adm_0, mod_0) = m'_1 \leftarrow \mathsf{MoD}\ (m_1, adm_1, mod_1)$ and then signs (m') using sk^B_{TABS} where $\Upsilon(\mathbb{S}_\mathcal{B}) = \Upsilon_0(\mathbb{S}_\mathcal{B}) = \Upsilon_1(\mathbb{S}_\mathcal{B}) = 1$ to obtain σ'_{full}. $\mathcal{B}$ returns $(m', (\sigma'_{fix,b}, \sigma'_{full}), adm, \Upsilon)$ where $adm = adm_0 = adm_1$ to $\mathcal{A}$ as the sanitizer's signature over m_b under adm_b ,and Υ_b. At the end, $\mathcal{A}$ outputs a bit 'a' which $\mathcal{B}$ relays as its answer to its $\mathcal{O}$LoRRDS oracle. Note that both messages m_0 and m_1 have the same modified message m' and since $\mathcal{B}$ signs the same m' for either m_0, or m_1, i.e., $m' = m'_0 \leftarrow \mathsf{MoD}(m_0, adm_0, mod_0) = m'_1 \leftarrow \mathsf{MoD}(m_1, adm_1, mod_1)$ from scratch using the TABS scheme to generate σ'_{full}, $\mathcal{A}$ cannot link the signature σ'_{full} to either m_0 or m_1 (since $m' \neq m_0$ and $m' \neq m_1$). Even if $\mathcal{A}$ is a successful adversary against the privacy of the underlying TABS scheme (see Fig. A.6), it could only deduce the identity of the TABS signer and\or the attributes used in signing m' but it is not able to link the signature over m' to either m_0 or m_1. Hence, the success of $\mathcal{A}$ in $\mathbf{Exp}^{Unlinkability}_{\mathcal{A},UP3S}$ implies the success of $\mathcal{B}$ in $\mathbf{Exp}^{Unlinkability}_{\mathcal{B},RDS}$. □

Theorem 2. *Given a private TABS scheme, then the sanitizable signature scheme in Fig. 4.8 is transparent.*

Proof. The privacy of the TABS scheme ensures that the generated signature reveals no information about the signer other than the fact that the signer possesses a set of attributes that satisfies a claim predicate. Hence, such a signature hides both the original signer identity and the attributes used to satisfy the predicate Υ as well. Therefore, by contradiction, we assume that the UP3S scheme is not a transparent sanitizable signature scheme. We then

show that the privacy of the underlying TABS scheme cannot hold. Let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}^{Transparency}_{\mathcal{A},UP3S}$, we show how to build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the privacy of the underlying TABS scheme and win in $\mathbf{Exp}^{Privacy}_{\mathcal{B},TABS}$ in Fig. A.6. $\mathcal{B}$ simulates $\mathcal{A}$'s UP3S oracles calls as follows; $\mathcal{B}$ first generates the keys (sk_{RDS}, pk_{RDS}) for the RDS scheme so that it can compute σ_{fix} on $H(pk_{UP3S}||m_{!adm}||adm||\Upsilon)$. $\mathcal{B}$ answers $\mathcal{A}$'s calls to $\mathcal{O}$SignUP3S by constructing the claim predicate Υ, calculating $H(pk_{UP3S}||m_{!adm}||adm||\Upsilon)$, signs the output using sk_{RDS} to get σ_{fix}, then forwards (m, Υ) to $\mathcal{O}$SignTABS oracle to get σ_{full}. To answer $\mathcal{A}$'s calls to $\mathcal{O}$SanitizeUP3S, $\mathcal{B}$ obtains σ'_{fix} by rerandomizing σ_{fix}, then it calculates $m' = \mathsf{MoD}(m, adm, mod)$ and forwards (m', Υ) to $\mathcal{O}$SignTABS oracle to get σ'_{full}. For $\mathcal{A}$ calls to $\mathcal{O}$ProveUP3S, $\mathcal{B}$ simply forwards $(m, \sigma_{full}, \Upsilon)$ to $\mathcal{O}$ProveTABS. When $\mathcal{A}$ inputs (m, mod, adm, Υ) to $\mathcal{O}$Sign-or-SanitUP3S,using its RDS keys $\mathcal{B}$ signs the message $H(pk_{UP3S}\|m_{!adm}\|adm\|\Upsilon)$, producing σ_{fix}. Then, $\mathcal{B}$ evaluates $m' \leftarrow \mathsf{MoD}(m, adm, mod)$ and passes (m', Υ) to the $\mathcal{O}$LoRSignTABS oracle, which responds with a challenge TABS signature σ_{full}. Finally, $\mathcal{B}$ returns $(m', (\sigma_{fix}, \sigma_{full}), adm, \Upsilon)$ to $\mathcal{A}$ as either the signer or sanitizer signature over m'. At the end, $\mathcal{A}$ outputs a bit 'a', which $\mathcal{B}$ forwards as its answer to the $\mathcal{O}$LoRSignTABS oracle.

□

Theorem 3. *Given an unforgeable RDS, and a collision-resistant hash function, the sanitizable signature scheme in Fig. 4.8 is immutable.*

Proof. Recall that for an adversary $\mathcal{A}$ against UP3S immutability to succeed in $\mathbf{Exp}^{Immutability}_{\mathcal{A},UP3S}$, it has to output a verifiable $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$ such that $m^* \notin \{\mathsf{MoD}\ (m_i,\ adm_i,\ .)\ |\ \mathsf{CheckMoD}\ (m_i, adm_i) = 1\ \}\ \forall\ i$ queries to $\mathcal{O}SignUP3S$ and $m^* \notin \{\mathsf{MoD}\ (m_j, adm_j, .)\ |\ \mathsf{CheckMoD}\ (m_j, adm_j) = 1\}\ \forall\ j$ queries to $\mathcal{O}SanitUP3S$ or $(m^* \in \{\mathsf{MoD}\ (m_i, adm_i, .)\ |\ \mathsf{CheckMoD}\ (m_i, adm_i) = 1\} \wedge \Upsilon^* \neq \Upsilon_i)\ \forall\ i$ queries to $\mathcal{O}SignUP3S$ or $(m^* \in \{\mathsf{MoD}$

$(m_j, adm_j, .) \mid \mathsf{CheckMoD}\ (m_j, adm_j) = 1\} \wedge \Upsilon^* \neq \Upsilon_j)) \ \forall\ j$ queries to $\mathcal{O}SanitUP3S$. Given a collision-resistant hash function H, by contradiction, we assume that the UP3S scheme is not immutable. We show that if we have a successful adversary $\mathcal{A}$ in $\mathbf{Exp}^{Immutability}_{\mathcal{A},UP3S}$, we can build an adversary $\mathcal{B}$ that wins the unforgeability of the underlying RDS signature scheme in $\mathbf{Exp}^{EUF-CMA}_{\mathcal{B},RDS}$ in Fig. A.3. $\mathcal{B}$ simulates $\mathcal{A}$'s environment with the help of the RDS signing oracle $\mathcal{O}\mathsf{SignRDS}$ as follows, $\mathcal{B}$ receives a public key pk^{Sign}_{RDS} from its experiment, initializes the TABS scheme, then generates a secret key of the TABS scheme $sk^{\mathcal{B}}_{TABS}$. It then passes to $\mathcal{A}$ both public keys and answers $\mathcal{A}$'s oracle queries as follows. $\mathcal{B}$ answers $\mathcal{A}$'s calls to $\mathcal{O}\mathsf{SignUP3S}$ by constructing the claim predicate Υ, calculating $H(pk_{UP3S}||m_{!adm}||adm||\Upsilon)$, then passes $H(pk_{UP3S}||m_{!adm}||adm||\Upsilon)$ to $\mathcal{O}\mathsf{SignRDS}$ to obtain σ_{fix}, and signs (pk_{UP3S}, m, Υ) using $sk^{\mathcal{B}}_{TABS}$ to generate σ_full. To answer $\mathcal{A}$'s calls to $\mathcal{O}\mathsf{SanitizeUP3S}$, $\mathcal{B}$ obtains σ'_{fix} by rerandomizing σ_{fix}, calculates $m' = \mathsf{MoD}(m, adm, mod)$, then signs $(pk_{UP3S}, m', \Upsilon)$ using its generated $sk^{\mathcal{B}}_{TABS}$ to evaluate σ'_{full}. When $\mathcal{A}$ eventually outputs $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$, $\mathcal{B}$ returns to its RDS unforgeability challenger in $\mathbf{Exp}^{EUF-CMA}_{\mathcal{B},RDS}$ the message $(H(m^*_{!adm}||adm^*||\Upsilon^*))$ and the forgery attempt σ^*_{fix} which is the forged RDS signature on the output of H on the input of $(m^*_{!adm}||adm^*||\Upsilon^*)$. Note that $\mathcal{A}$ succeeds if it outputs a verifiable σ_{fix} under the original signer's public key where $m^* \notin (\{\mathsf{MoD}\ (m_i, adm_i, .) \mid \mathsf{CheckMoD}\ (m_i, adm_i) = 1\} \wedge m^* \notin \{\mathsf{MoD}\ (m_j, adm_j, .) \mid \mathsf{CheckMoD}\ (m_j, adm_j) = 1\})$ *or* $(m^* \in \{\mathsf{MoD}\ (m_i, adm_i, .) \mid \mathsf{CheckMoD}(m_i, adm_i) = 1\} \wedge \Upsilon^* \neq \Upsilon_i) \vee (m^* \in \{\mathsf{MoD}\ (m_j, adm_j, .) \mid \mathsf{CheckMoD}\ (m_j, adm_j) = 1\} \wedge \Upsilon^* \neq \Upsilon_j))$, hence $H(m^*_{!adm}||adm^*||\Upsilon^*)$ was not queried by $\mathcal{B}$ to its RDS signing oracle before in either case which implies a valid forgery by $\mathcal{B}$. □

Theorem 4. *Given a non-frameable and traceable TABS scheme, then the sanitizable signature scheme in Fig. 4.8 achieves accountability.*

Proof. Recall that the non-frameability security property of a TABS scheme ensures that even if all authorities and users of the scheme collude, they cannot produce a signature that traces to an honest user whose secret key has not been learned by the adversary. In other words, any generated signature must be traced back to the entity that holds the secret key used in signing such a message. Moreover, the traceability security property of a TABS scheme ensures that every message signature pair generated could be traced. By contradiction, we let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}^{Accountability}_{\mathcal{A},UP3S}$ and show that we can build an adversary $\mathcal{B}$ (resp. $\mathcal{B}'$) which can break the non-frameability (resp. traceability) of the underlying TABS scheme and win in $\mathbf{Exp}^{Non-frameability}_{\mathcal{B},TABS}$ in Fig. A.7 (resp. $\mathbf{Exp}^{Traceability}_{\mathcal{B}',TABS}$ in Fig. A.8). $\mathcal{B}$ simulates $\mathcal{A}$'s oracles as follows. $\mathcal{B}$ first generates keys (sk_{RDS}, pk_{RDS}) for the underlying RDS scheme so $\mathcal{B}$ can compute σ_{fix} on $H(m_{!adm}||adm||\Upsilon)$. When $\mathcal{A}$ queries $\mathcal{O}$SignUP3S with $(m_i, adm_i, \mathbb{S}_{PSan})$, $\mathcal{B}$ constructs the claim predicate Υ and uses the RDS key pairs to compute $\sigma_{fix,i}$ on $H(m_{!adm,i}, adm_i, \Upsilon_i)$ and forwards (m_i, Υ_i) to the TABS signing oracle $\mathcal{O}$SignTABS to get $\sigma_{full,i}$ on m_i and then forwards the tuple $((\sigma_{fix,i}, \sigma_{full,i}), adm_i, \Upsilon_i)$ to $\mathcal{A}$. When $\mathcal{A}$ queries $\mathcal{O}$SanitUP3S with m_j, $\sigma_{m,j}$, adm_j, mod_j, Υ_j, $\mathcal{B}$ rerandomize $\sigma_{fix,j}$ to get $\sigma'_{fix,j}$, then calculates $m'_j \leftarrow \mathsf{MoD}(m_j, adm_j, mod_j)$ and forwards (m'_j, Υ_j) to the TABS signing oracle $\mathcal{O}$SignTABS to get $\sigma'_{full,j}$ on m_j and then forwards the tuple $((\sigma'_{fix,j}, \sigma'_{full,j}), adm_j, \Upsilon_j)$ to $\mathcal{A}$. For $\mathcal{O}$ProveUP3S queries by $\mathcal{A}$, $\mathcal{B}$ forwards the queries directly to the Prove oracle of the TABS scheme $\mathcal{O}$ProveTABS and relays back the output. At the end $\mathcal{A}$ outputs a tuple $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$, $\mathcal{B}$ forwards $(m^*, \sigma^*_{full}, \Upsilon^*)$ to its non-frameability challenger in $\mathbf{Exp}^{Non-frameability}_{\mathcal{B},TABS}$ experiment in Fig. A.7. On the other hand, $\mathcal{B}'$ could be constructed in a similar way to $\mathcal{B}$. However, when $\mathcal{A}$ outputs a tuple $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$, $\mathcal{B}'$ forwards $(m^*, \sigma^*_{full}, \Upsilon^*)$ to its traceability challenger in $\mathbf{Exp}^{Traceability}_{\mathcal{B}',TABS}$ in Fig. A.8. Therefore, the success of $\mathcal{A}$ in $\mathbf{Exp}^{Accountability}_{\mathcal{A},UP3S}$ implies the

success of $\mathcal{B}$ and $\mathcal{B}'$ in $\mathbf{Exp}^{Non-frameability}_{\mathcal{B},TABS}$ and $\mathbf{Exp}^{Traceability}_{\mathcal{B}',TABS}$, respectively. □

Theorem 5. *Given an unforgeable RDS scheme, an unforgeable TABS scheme, and a collision-resistant hash function, the sanitizable signature scheme in Fig. 4.8 is unforgeable.*

Proof. Recall that the unforgeability security property of a TABS scheme ensures that an adversary cannot generate a valid signature under a predicate where it does not possess the corresponding set of attributes that satisfy such a predicate. Moreover, the unforgeability security property of an RDS scheme ensures that it is infeasible for an adversary who does not have access to the signing keys to output a valid message signature pair. Given a collision-resistant hash function H, by contradiction, we let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}^{Unforgeability}_{\mathcal{A},UP3S}(\lambda)$, then we show that we can build an adversary $\mathcal{B}$ (resp. $\mathcal{B}'$) which can break the unforgeability of the underlying TABS scheme (resp. RDS scheme) and win in $\mathbf{Exp}^{Unforgeability}_{\mathcal{B},TABS}(\lambda)$ in Fig. A.5 (resp. $\mathbf{Exp}^{EUF-CMA}_{\mathcal{B}',RDS}(\lambda)$ in Fig. A.3). $\mathcal{B}$ simulates $\mathcal{A}$'s oracles as follows. $\mathcal{B}$ first generates the keys (sk_{RDS}, pk_{RDS}) for the underlying RDS scheme. When $\mathcal{A}$ queries $\mathcal{O}\mathsf{SignUP3S}$ with $(m_i, adm_i, \mathbb{S}_{PSan})$, $\mathcal{B}$ constructs the claim predicate Υ_i and uses the RDS secret key to compute $\sigma_{fix,i}$ on $H(m_{!adm,i}, adm_i, \Upsilon_i)$ and forwards (m_i, Υ_i) to the TABS signing oracle $\mathcal{O}\mathsf{SignTABS}$ to get $\sigma_{full,i}$ on m_i and then answers $\mathcal{A}$ with the tuple $((\sigma_{fix,i}, \sigma_{full,i}), adm_i, \Upsilon_i)$. When $\mathcal{A}$ queries $\mathcal{O}\mathsf{SanitUP3S}$ with $(m_j, \sigma_{m,j}, adm_j, mod_j, \Upsilon_j)$ where $\sigma_{m,j} = (\sigma_{fix,j}, \sigma_{full,j})$, $\mathcal{B}$ rerandomize $\sigma_{fix,j}$ to get $\sigma'_{fix,j}$, then calculates $m'_j \leftarrow \mathsf{MoD}(m_j, adm_j, mod_j), \Upsilon_j)$ and forwards (m'_j, Υ_j) to the TABS signing oracle $\mathcal{O}\mathsf{SignTABS}$ to get $\sigma'_{full,j}$ on m_j and then forwards the tuple $((\sigma'_{fix,j}, \sigma'_{full,j}), adm_j, \Upsilon_j)$ to $\mathcal{A}$. For $\mathcal{O}\mathsf{ProveUP3S}$ queries by $\mathcal{A}$, $\mathcal{B}$ forwards the queries directly to the Prove oracle of the TABS scheme $\mathcal{O}\mathsf{ProveTABS}$ and relays the output back to $\mathcal{A}$. At the end of $\mathbf{Exp}^{Unforgeability}_{\mathcal{A},UP3S}(\lambda)$, $\mathcal{A}$ outputs a tuple (m^*,

$sigma^*_m$, adm^*, Υ^*) where σ^*_m = (σ^*_{fix}, σ^*_{full}), and $\mathcal{B}$ forwards (m^*, σ^*_{full}, Υ^*) to its unforgeability challenger in $\mathbf{Exp}^{Unforgeability}_{\mathcal{B},TABS}(\lambda)$. On the other hand, an RDS unforgeability adversary $\mathcal{B}'$ is constructed as follows. To simulate $\mathcal{A}$'s oracles, $\mathcal{B}'$ initializes the TABS scheme and generates the secret key $sk^{B'}_{TABS}$ for some identity $i_{B'}$ and a set of attributes ($\mathbb{S}_{B'}$) s.t $\Upsilon(\mathbb{S}_{B'}) = 1$ for any Υ, hence $\mathcal{B}'$ can compute σ_{full} on m for any predicate. When $\mathcal{A}$ queries $\mathcal{O}$SignUP3S with $(m_i, adm_i, \mathbb{S}_{PSan})$, $\mathcal{B}'$ constructs the claim predicate Υ_i such that $\Upsilon(\mathbb{S}_{B'}) = 1$ and $\Upsilon(\mathbb{S}_{PSan}) = 1$, computes $H(m_{!adm,i}, adm_i, \Upsilon_i)$ and forwards it to the RDS signing oracle $\mathcal{O}$SignRDS to get $\sigma_{fix,i}$ on m_i. Then $\mathcal{B}'$ uses its own TABS secret key to compute $\sigma_{full,i}$ on m_i, and then forwards the tuple $((\sigma_{fix,i}, \sigma_{full,i}), adm_i, \Upsilon_i)$ to $\mathcal{A}$. When $\mathcal{A}$ queries $\mathcal{O}$SanitUP3S with $m_j, \sigma_{m,j}, adm_j, mod_j, \Upsilon_j$, $\mathcal{B}'$ rerandomize $\sigma_{fix,j}$ to get $\sigma'_{fix,j}$, then calculates $m'_j \leftarrow \mathsf{MoD}(m_j, adm_j, mod_j)$ and signs $\sigma'_{full,j}$ using $sk^{B',i_{B'}}_{TABS}$ and then forwards the tuple $((\sigma'_{fix,j}, \sigma'_{full,j}), adm_j, \Upsilon_j)$ to $\mathcal{A}$. For $\mathcal{O}$ProveUP3S queries by $\mathcal{A}$, $\mathcal{B}'$ returns its own identity and a valid proof for all queries. At the end, $\mathcal{A}$ outputs a tuple $(m^*, \sigma^*_m, adm^*, \Upsilon^*)$ where σ^*_m = (σ^*_{fix},σ^*_{full}), and $\mathcal{B}'$ forwards (m^*, σ^*_{fix}) to its unforgeability challenger in $\mathbf{Exp}^{EUF-CMA}_{\mathcal{B}',RDS}(\lambda)$. Therefore, the success of $\mathcal{A}$ in $\mathbf{Exp}^{Unforgeability}_{\mathcal{A},UP3S}$ implies the success of $\mathcal{B}$ and $\mathcal{B}'$ in $\mathbf{Exp}^{Unforgeability}_{\mathcal{B},TABS}(\lambda)$ and $\mathbf{Exp}^{EUF-CMA}_{\mathcal{B}',RDS}(\lambda)$, respectively. □

4.5 Instantiation and Efficiency

We instantiate UP3S with Pointcheval-Sanders (PS) RDS Scheme [76] because of its short signature size and low signing and verification costs. For the TABS scheme, we utilize the DTABS scheme in [53] because in addition to providing all the security properties required by UP3S, DTABS offers minimal trust in the attribute authorities by defining a stronger definition for non-frameability, i.e., when all authorities and users collude, they can not

frame an honest user. This stronger notion of non-frameability overcomes the shortcomings in standard ABS schemes where the attribute keys are generated by the attribute authority for the scheme's users (signers and sanitizers in UP3S) and hence, the attribute authority has to be fully trusted. Another advantage of using DTABS is the ability to add multiple attribute authorities to the scheme dynamically, which further supports UP3S's scalability. Both PS and DTABS are instantiated in a type-3 bilinear group setting. We use instantiation 1 of DTABS for its shorter signature size [53]. The hash function H should be chosen such that its output is mapped to $\mathbb{Z}_P^*$, thus the PS scheme is used in a single message signature setting where $m \in \mathbb{Z}_p$ to produce σ_{fix}.

Efficiency. To sign a message, the signer needs to generate a hash, an RDS signature on the output of the hash function, and a TABS signature on the whole message. To sanitize a message, the sanitizer has to modify the message, rerandomize the RDS signature and generate a TABS signature for the modified message. To verify a message signature pair, the verifier verifies both the RDS and TABS signatures. Tracing a signature to its origin requires verifying the sanitizable signature, and running the tracing algorithm of the underlying TABS scheme. Finally, to verify the output of the tracing algorithm, the judge procedure verifies both the sanitizable signature and the proof generated by the tracing algorithm of the TABS scheme. The computation and communication complexities of the instantiated UP3S are as follows. Let $l \times t$ be the size of DTAB's claim-predicate monotone span program [53]. The proposed instantiation produces a total signature (σ_{fix}, σ_{full}) size of $(27.l + 21)$ elements in $\mathbb{G}_1$ + $(22.l + 15)$ elements in $\mathbb{G}_2$ + $(t + 3)$ elements in $\mathbb{Z}_p$, where σ_{fix} is a PS signature of size 2 elements in $\mathbb{G}_1$ and requires two modular exponentiations in G_1 [76], and σ_{full} is a DTABS signature of size $(27.l + 19)$ elements in $\mathbb{G}_1$ + $(22.l + 15)$ elements $\mathbb{G}_2$ + $(t + 3)$ elements in $\mathbb{Z}_p$ [53] and, costs approximately $(27l + 32)$ modular exponentiation in G_1 +

$(38l+34)$ modular exponentiation in G_2 to produce. Note that the aforementioned signature size and computational cost apply to both signing and sanitizing a given message. Verifying a given UP3S message signature pair costs a total of $(32l + 80)$ pairing operations + 1 modular exponentiation in G_1 + 2 modular exponentiation in G_2[1]. On the other hand, to trace a signature to its origin, the tracing authority produces 2 elements in $\mathbb{G}_2$ and performs 2 modular exponentiation in $\mathbb{G}_2$ in addition to the cost of UP3S signature verification. The judge procedure performs 4 pairing operations to verify the proof of the tracing procedure.

4.6 Comparing UP3S to P3S

In what follows, we provide a comparison between UP3S and P3S with respect to their features and security models. The reader is referred to [89] for the formal definition of P3S and its security notions. Note that comparing the efficiency of the UP3S and P3S schemes is not possible because P3S does not provide an efficiency evaluation for its suggested instantiation. Also, both generic schemes have different building blocks and there are no standard metrics for the associated complexities of the generic building blocks, i.e., PCH and a group signature scheme in P3S compared to TABS and RDS in UP3S.

Features comparison We compare UP3S with P3S in terms of the roles of the scheme's entities, features of its procedures, and scalability. *Signing*. In UP3S, the signer's responsibility is limited only to signature generation and sanitization policy definition, and no interaction is needed from the signer to reveal the identity of the actual signer or sanitizer of a given message signature pair. In P3S, signers act as group managers, where they add new sanitizers to the system in addition to acting as openers for the group signature on the

[1]The verification cost of DTABS could be enhanced using batch verification [21] of the underlying Groth-Sahai proof of knowledge [58]

message. In P3S, the signer should know the identity/public key of at least one sanitizer prior to signature generation in order to be able to create the group signature using a NIZK OR proof. However, in UP3S, the signer defines a sanitization policy (signing predicate) which determines possible future sanitizers based on their attributes only, and no need to know the identity/key of any of them prior to signature generation.

Sanitizing. Unlike P3S which uses a policy-based chameleon hash as its core building block, UP3S uses a TABS scheme. Thus, it is not required to share any trapdoor information with every sanitizer before sanitizable signature generation as in the case of P3S. In P3S, Υ is only used as an input to the signing algorithm and could not be verified during the signature verification. In UP3S, Υ is an input to all its subsequent algorithms, hence any of UP3S algorithms can verify that a message signature pair is generated by a signer\sanitizer who possesses a set of attributes satisfying Υ. Furthermore, in UP3S, the sanitization rights of a given message are solely controlled by the attribute set possessed by any scheme user. Hence, UP3S neither requires a group manager role nor defines an AddSan procedure (Def. 6 in [89]) as in P3S, which is used by the group manager to grant the sanitization rights of a given message to a specific sanitizer.

Scalability. In P3S, the signature size should grow linearly with the number of group members (possible future sanitizers) which is required to achieve transparency in a linkable signature scheme. More precisely, like in group signature schemes, P3S generates a NIZK OR proof that proves that the encrypted public key (identity) of the signature generator for a given message is either the original signer OR a sanitizer. However, since P3S is linkable, assuming a given timeline for signature generation, an observer can link two signatures originating from two different sanitizers to their original message. Thus, using the description of the NIZK in construction 1 in [89] where the anonymity set is always equal to two (the

signer identity is always in the set), an adversary can determine with more than the negligible probability if the second message is sanitized or not which contradicts the transparency requirement. In UP3S, every message has a specific sanitization policy with no sanitizers identities included in the signature, and whatever the number of sanitizers who are authorized to sanitize a given message, the signature size is fixed per the associated sanitization policy. In P3S all system-wide parameters including secret-public key pairs are initialized from scratch for each message, i.e., a new chameleon hash instance, which may limit the system's scalability. UP3S on the other hand is based on an ABS scheme where once initialized, signers (resp. sanitizers) can sign (resp. sanitize) any message, and sanitization rights are controlled by a predicate defined by the signer only.

Table 4.1 summarizes the features comparison between UP3S and P3S in terms of the building blocks, if the scheme requires knowing future sanitizers or not, sanitization technique, how sanitization rights are granted, signature size, and if a group manager is needed.

Table 4.1: Comparison between UP3S and P3S.

	UP3S (this work)	P3S [89]
Building blocks	TABS and RDS	PCH and GSS
Unlinkability	yes	no
Future sanitizers	no	at least one
Sanitization technique	ABS	secret key sharing
Sanitization rights	set prior to sig. gen.	granted after sig. gen.
Signature size	fixed*	variable**
Group manager	no	yes

GSS: Group signature scheme

* Per message sanitization policy

** To achieve transparency, the signature should grow linearly with the number of group members (possible future sanitizers of a certain message)

Security models comparison Our security definitions introduce some modifications to the definitions which are proposed in P3S to capture the roles and features of the un-

derlying building blocks in UP3S. P3S defines nine security properties, namely unforgeability, immutability, privacy, transparency, pseudonymity, signer-accountability, sanitizer-accountability, proof-soundness, and traceability [89]. Besides unlinkability which is not offered by P3S, UP3S defines unforgeability, immutability, privacy, transparency, and accountability as its required notion of security. In what follows, we compare the definitions of the security properties of both schemes.

Unforgeability. Unlike UP3S, P3S uses the concept of groups and defines unforgeability in a way to capture the various cases that arise where groups are used, such as secret signing keys can be re-used across multiple groups and sanitization between different groups. On the other hand, UP3S does not use groups, accordingly, the unforgeability experiment (see Fig. 4.7) is defined with no consideration for forgery cases associated with groups as in P3S.

Immutability. Both P3S and UP3S definitions follow the original definition in [26]. However, in UP3S's immutability experiment (see Fig. 4.4), we give the adversary access to the sanitization oracle to consider double sanitization cases where a sanitized message could be further sanitized by a different sanitizer who fulfills the sanitization policy.

Privacy. P3S defines a stronger notion of privacy, to capture secret key leakage and bad randomness in key generation use cases. However, since UP3S provides unlinkability and it has been proven in [28] that unlinkability implies privacy, UP3S follows the definition in [28].

Transparency. Both schemes follow Brzuska *et. al* definition of transparency [26]. However, both schemes designed the experiment with different inputs to the oracles due to the difference in the used building blocks.

Pseudonymity. P3S defines pseudonymity as the infeasibility that an adversary can decide which sanitizer actually is responsible for a given signature. P3S modeled such property

by an experiment where the adversary input a message signature pair, some modifications, and two possible sanitizers' secret keys to the left-or-right sanitization oracle. The adversary wins if it can decide which sanitizer secret key is used by the left-or-right sanitization oracle (see Fig.8 in [89]). To prove the independence of pseudonymity, the authors assume that the sanitizer's identity is encoded such that it can only be recovered if both the sanitized and the original signatures are available to the adversary. We find the latter assumption counter-intuitive to the transparency requirement because such an adversary can decide with certainty which of the signatures is freshly signed and which is sanitized. UP3S provides a stronger notion of pseudonymity since it defines unlinkability (see Theorem. 1), where such an assumption can not hold while preserving unlinkability.

Accountability. P3S uses the signer secret key to open a signature and trace it to the identity (the public key) of the signer/sanitizer of a given message. Hence, it defines two types of accountability, signer-accountability, and sanitizer-accountability. Moreover, P3S defines traceability to capture the case when the opening algorithm returns $\perp$. On the other hand, UP3S uses a separate tracing authority to trace a signature back to its actual signer and does not use the signer keys in the tracing process. Hence, UP3S defined one security property, accountability in Fig. 4.5, which captures the cases of signer-accountability, sanitizer-accountability, and traceability in P3S.

Proof-Soundness. P3S constructs a dynamic-group-signature-like scheme, hence, it introduces proof-soundness to resist signature hijacking in group signatures where an adversary can generate a valid NIZK for an already signed message that traces back to another user [88]. In UP3S, traceability is provided by the underlying TABS scheme, where its traceability-soundness notion (see tracing soundness in [53]) serves the same goal.

Chapter 5

Traceable Policy-Based Signatures with Delegation

In this chapter, we propose a Traceable Policy-Based Signature (TPBS) scheme, which extends the functionality of policy-based signatures (PBS) by incorporating a tracing mechanism to enforce accountability while maintaining the essential properties of PBS schemes. TPBS achieves traceability without compromising delegatability by introducing identity keys, which are used alongside policy keys for signing. Additionally, TPBS ensures non-frameability, even under the assumption of a corrupted tracing authority. This work is published in CANS 2023 [4]

To construct TPBS, we employ a rerandomizable signature scheme, a digital signature scheme, and a zero-knowledge proof system as its building blocks. Unlike prior traceable PBS approaches, such as the scheme proposed by Xu et al. [96], where user identities are directly attached to policies, TPBS decouples the identity key from the policy key. This design choice preserves the original intent of PBS schemes by allowing users with access to a policy key to sign messages without binding their identities to a specific policy. Con-

sequently, TPBS supports policy key delegation while preventing key misuse and ensuring accountability.

We define and formally prove the extractability, simulatability, non-frameability, and traceability security notions for the generic TPBS scheme. Additionally, we propose a concrete instantiation of TPBS utilizing the Pointcheval-Sanders rerandomizable signature scheme, Abe et al.'s structure-preserving signature scheme, and the Groth-Sahai non-interactive zero-knowledge (NIZK) proof system. We analyze the efficiency of this instantiation and compare its performance with existing approaches.

5.1 Traceable Policy-Based Signatures (TPBS)

We build on PBS and present a Traceable Policy-Based Signatures (TPBS) scheme. The main idea of our scheme is that in addition to the PBS issuer's policy key, we require the use of an identity key for signing a message that satisfies the policy defined by the issuer in the policy key. Hence, we introduce a Tracing Authority (TA) where every scheme user registers with to generate an identity key. The user then uses the identity key in addition to the policy key to sign a message that conforms to the policy set by the issuer. The produced signature allows the TA to trace it to the registration information acquired from the user during identity key generation. Note that contrary to the issuer's policy key, which could be shared among users allowed by the issuer to sign a specific message, the identity key is generated by individual users and is not shared with any other entity in the system.

Throughout this chapter, we use $x \xleftarrow{\$} \mathbb{Z}_p$ to denote sampling x uniformly at random from $\mathbb{Z}_q$. We denote by i an identity from the identity universe $\mathbb{I}$, $i \in \mathbb{I}$. Let $\lambda \in \mathbb{N}$ denotes our security parameter, then a function $\epsilon(\lambda) : \mathbb{N} \rightarrow [0, 1]$ denotes the negligible function if for

any $c \in \mathbb{N}$, $c > 0$ there exists $\lambda_c \in \mathbb{N}$ s.t. $\epsilon(\lambda) < \lambda^{-c}$ for all $\lambda > \lambda_c$. We use $f(.)$ to denote a one-way function with a domain denoted by $\mathcal{F}$, and we use $\mathsf{PoK}(x : C = f(x))$ to denote an interactive perfect zero-knowledge proof of knowledge of x such that $C = f(x)$ [41]. Let a policy checker (PC) denote an NP-relation $\mathsf{PC} : \{0,1\}^*\{0,1\}^* \leftarrow \{0,1\}$, where the first input is a pair (p, m) representing a policy $p \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$, while the second input is a witness $w_p \in \{0,1\}^*$. The signing of m is permitted under policy p if (p, m, w_p) is PC-valid such that $\mathsf{PC}((p, m), w_p) = 1$ [12].

Definition 8. *(Policy Checker [12]). Let a policy checker (*PC*) denote an NP-relation* $\mathsf{PC} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$*, where the first input is a pair* (p, m) *representing a policy* $p \in \{0,1\}^*$ *and a message* $m \in \{0,1\}^*$*, while the second input is a witness* $w_p \in \{0,1\}^*$*. The signing of* m *is permitted under policy* p *if* (p, m, w_p) *is* PC*-valid such that* $\mathsf{PC}((p, m), w_p) = 1$.

In what follows, we give the black box definitions of the proposed construction.

TPBS is a tuple of ten polynomial-time algorithms, TPBS = {ppGen, TASetup, IssuerSetup, UserKeyGen, IDKeyGen, PolicyKeyGen, Sign, Verify, Trace, Judge} which are defined as follows.

- ppGen. This algorithm outputs the public parameters of the scheme, which become an implicit input to all the other algorithms.

$$pp_{TPBS} \leftarrow \mathsf{ppGen}(1^\lambda)$$

- TASetup. This algorithm generates the TA's public secret key pair $(pk^{TA}_{TPBS}, sk^{TA}_{TPBS})$,

and initializes a private empty registry Reg at the TA.

$$(pk^{TA}_{TPBS}, sk^{TA}_{TPBS}, Reg) \leftarrow \mathsf{TASetup}(pp_{TPBS})$$

- IssuerSetup. This algorithm generates the issuer's public key secret key pair.

$$(pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS})$$

- UserKeyGen. For user identity $i \in \mathbb{I}$, this algorithm generates the user's secret public key pair (sk_i, pk_i). We assume that pk_i is authentically associated with i in a public registry $\mathcal{D}$ such that $\mathcal{D}[i] = pk_i$, a PKI system may be used for such a purpose. Moreover, this algorithm outputs the registration information ID_i generated from sk_i using a one-way function.

$$(pk_i, sk_i, ID_i) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS}, i)$$

- IDKeyGen. This two-party interactive procedure runs between a scheme user and the TA to generate the user's identity key. The inputs of the user's routine are $(i, (sk_i))$, and the inputs to the TA's routine are $((sk^{TA}_{TPBS}), i, ID_i)$, where i and ID_i are sent to the TA by the user. At the end of the interaction, the user obtains the TA's signature σ^i_{ID} over their hidden secret sk_i. Finally, the user sets $sk^i_{TPBS} = (sk_i, \sigma^i_{ID})$ whereas the TA obtains some registration information $Reg[i] = ID_i$.

$$((Reg[i]), (sk^i_{TPBS})) \leftarrow \mathsf{IDKeyGen}((sk^{TA}_{TPBS}) \xleftarrow[\sigma^i_{ID}]{(i,ID_i)} (sk_i)),$$

where the first (resp. second) $(.)$ in the input and output of IDKeyGen contains values that are only known to the TA (resp. user).

- PolicyKeyGen. The issuer runs this procedure to generate a secret key for a specific policy $p \in \{0,1\}^*$.

$$sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p)$$

- Sign. On input of a message m, a witness $w_p \in \{0,1\}^*$ that m conforms to a specific policy p, the secret signing key sk^p_{TPBS}, the user identity key sk^i_{TPBS} , this procedure generates a signature σ_m.

$$\sigma_m \leftarrow \mathsf{Sign}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^p_{TPBS}, sk^i_{TPBS}, m, p, w_p)$$

- Verify. This algorithm verifies the signature σ_m over m using the issuer's and TA's public keys.

$$\{\top, \bot\} \leftarrow \mathsf{Verify}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m, \sigma_m)$$

- Trace. This algorithm is run by the TA to trace a signature σ_m over m to its original signer and returns the signer identity along with proof confirming such a claim.

$$(i, \pi_{Trace}) \leftarrow \mathsf{Trace}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, Reg, m, \sigma_m)$$

- Judge. This algorithm verifies the output of the tracing algorithm.

$$\{\top, \bot\} \leftarrow \mathsf{Judge}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m, \sigma_m, i, \pi_{Trace})$$

TPBS Correctness for the correctness of TPBS, we require that for all $\lambda \in \mathbb{N}$, all $pp_{TPBS} \leftarrow$ ppGen (1^λ), for all $(pk^{TA}_{TPBS}$,$(sk^{TA}_{TPBS}$,$Reg))$ $\leftarrow$ TASetup(pp_{TPBS}), for all $(pk^{Issuer}_{TPBS}$,$sk^{Issuer}_{TPBS})$ $\leftarrow$ IssuerSetup (pp_{TPBS}), for all $i \in \mathbb{I}$, for all $(pk_i, sk_i, ID_i) \leftarrow$ UserKeyGen (pp_{TPBS}), for

all $((Reg[i]),(sk^i_{TPBS})) \leftarrow \mathsf{IDKeyGen}\ ((sk^{TA}_{TPBS}) \xleftrightarrow[\sigma^i_{ID}]{(i,ID_i)} (sk_i))$, for all $sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}\ (sk^{Issuer}_{TPBS}, p)$, and for all $(m, p, w_p) \in \{0,1\}^*$ s.t $\mathsf{PC}((p, m), w_p) = 1$, we have $\sigma_m \leftarrow \mathsf{Sign}\ (pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^p_{TPBS}, sk^i_{TPBS}, m, p, w_p)$ such that $\top \leftarrow \mathsf{Verify}\ (pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m, \sigma_m)$. Moreover, we have $(i, \pi_{Trace}) \leftarrow \mathsf{Trace}\ (pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, Reg, m, \sigma_m)$ such that $\top \leftarrow \mathsf{Judge}\ (pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m, \sigma_m, i, \pi_{Trace})$.

To prevent a misbehaving TA or any party who has access to the policy key sk^p_{TPBS} from framing a user, we ensure that sk^i_{TPBS} contains sk_i which is generated by individual users and not shared with any entity in the scheme. Moreover, Since our scheme segregates the identity keys from the policy keys, the delegatability of policy keys becomes a natural extension for our scheme and could be achieved seamlessly by applying the same technique of Bellare and Fuchsbauer [12]. Moreover, segregating the issuer and TA rules make our scheme a perfect fit for decentralized environments where multiple issuers may coexist.

5.2 TPBS Security Definitions

The security notions of PBS are privacy (policy-indistinguishability) and unforgeability [12]. Privacy of the policy ensures that a signature reveals neither the policy associated with the policy key nor the witness that was used in creating such a signature. Unforgeability is defined as the infeasibility of creating a valid signature for a message m without holding a policy key for some policy p and a witness w_p such that $\mathsf{PC}((p, m), w_p) = 1$. In the same context, Bellare and Fuchsbauer have defined simulatability and extractability as stronger versions of the aforementioned security notions [12]. The main reason behind introducing such stronger notions is that the traditional notions of policy privacy and unforgeability are insufficient for all applications. For instance, a PBS scheme with a policy checker PC

such that for every message m, there is only one policy p where $\mathsf{PC}((p, m_i), w_i) = 1$ for $i \in \{0, \ldots, n\}$, such a scheme does not hide the policy, yet still satisfies indistinguishability.

Since TPBS signing requires the user's identity key and the produced signatures are traceable by the TA, we extend the definition of privacy to include user anonymity in addition to policy-privacy. Moreover, we define non-frameability and traceability to capture the newly introduced traceability feature. We also define simulatability and extractability as the stronger notions of privacy and unforgeability. Note that our definition of simulatability and extractability differs from those in PBS in that they include the newly introduced signer identity and tracing feature. In what follows, we give the formal definitions of the TPBS security notions. The oracles used in the security experiments are defined in Fig. 5.1.

Note that $\mathcal{O}\mathsf{KeyGen}$ is set up to generate the signer identity key from scratch and return it to the adversary along with the policy key. Such a setup allows the adversary to corrupt as many users as it wants without engaging with the oracle interactively. ***Privacy*** TPBS ensures privacy if it guarantees signer anonymity and policy-privacy, which are defined as follows.

Signer anonymity. Anonymity is modeled by the indistinguishability experiment in Fig. 5.2, where the adversary has access to $\mathcal{O}\mathsf{KeyGen}(.)$, $\mathcal{O}\mathsf{USign}(.)$, $\mathcal{O}IdLoRSign$, and $\mathcal{O}\mathsf{Trace}(.)$ oracles. The challenge oracle $\mathcal{O}IdLoRSign$ is initialized with a random bit $b \in \{0, 1\}$. The adversary inputs to $\mathcal{O}IdLoRSign$ are (i_0, i_1, m, p, w_p) where the adversary chooses i_0, i_1 from a predefined list of users $\mathcal{U}$ that it has no access to their signing keys. After verifying that $\mathsf{PC}((p, m), w_p) = 1$ and $i_0, i_1 \in \mathcal{U}$, the oracle generates σ_{m_b} for the message m using (sk^p_{TPBS},$sk^{i_b}_{TPBS}$). Finally, the oracle returns the tuple $(\sigma_{m_b}, sk^p_{TPBS})$. The adversary wins if it can determine the bit b with more than the negligible probability. The adversary has access to $\mathcal{O}\mathsf{USign}(.)$ oracle, which on input $(i \in \mathcal{U}, m, p, w_p)$, it obtains a signature on mas-

$\mathcal{O}\mathsf{KeyGen}(i, p)$

if $i \in \mathcal{U}$ **return** $\perp$
$(pk_i, sk_i, ID_i) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS})$
$((Reg[i]), (sk^i_{TPBS})) \leftarrow \mathsf{IDKeyGen}((sk^{TA}_{TPBS}) \xleftrightarrow[\sigma^i_{ID}]{(i,ID_i)} (sk_i))$
$sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p)$
$\mathcal{T} = \mathcal{T} \cup \{i, sk_i\}; \quad \mathcal{L} = \mathcal{L} \cup \{p\}$
return $(sk^i_{TPBS}, sk^p_{TPBS})$

$\mathcal{O}\mathsf{USign}(i_j, m, p, w_p)$

if $\mathsf{PC}((p, m), w_p) = 0 \vee i_j \notin \mathcal{U}$
  return $\perp$
$(sk^{i_j}_{TPBS}) \leftarrow \mathcal{Q}_i[i_j]$
$sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p)$
$\sigma_m \leftarrow \mathsf{Sign}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^p_{TPBS}, sk^{i_j}_{TPBS}, m, p, w_p)$
$\mathcal{M} = \mathcal{M} \cup \sigma_m$
return $(\sigma_m, \sigma^i_{ID}, sk^p_{TPBS})$

$\mathcal{O}\mathsf{IdLoRSign}(i_{j_0}, i_{j_1}, m, p, w_p)$

if $\mathsf{PC}((p, m), w_p) = 0 \vee i_{j_0}, i_{j_1} \notin \mathcal{U}$
  return $\perp$
$(sk^{i_0}_{TPBS}) \leftarrow \mathcal{Q}_i[j_0][1]; \quad (sk^{i_1}_{TPBS}) \leftarrow \mathcal{Q}_i[j_1][1]$
$sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p)$
$\sigma_{m_b} \leftarrow \mathsf{Sign}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^p_{TPBS}, sk^{i_b}_{TPBS}, m, p, w_p)$
$\mathcal{M}' = \mathcal{M}' \cup (m, \sigma_{m_b})$
return $(\sigma_{m_b}, sk^p_{TPBS})$

$\mathcal{O}\mathsf{Sign}(m, i, p, w_p)$

if $i \in \mathcal{T} \wedge p \in \mathcal{L}$
  return $\perp$
if $i \in \mathcal{Q}_i;\ sk^i_{TPBS} = \mathcal{Q}_i[i]$
elseif $p \in \mathcal{Q}_p;\ sk^p_{TPBS} = \mathcal{Q}_p[p]$
else
  $(pk_i, sk_i, ID_i) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS})$
  $((Reg[i]), (sk^i_{TPBS})) \leftarrow \mathsf{IDKeyGen}((sk^{TA}_{TPBS}) \xleftrightarrow[\sigma^i_{ID}]{(i,ID_i)} (sk_i))$
  $sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p)$
  $\mathcal{Q}_i[i] = sk^i_{TPBS};\ \mathcal{Q}_p[p] = sk^p_{TPBS}$
$\sigma_m \leftarrow \mathsf{Sign}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^p_{TPBS}, sk^i_{TPBS}, m, p, w_p)$
$\mathcal{M} = \mathcal{M} \cup (m, \sigma_m)$
return σ_m

$\mathcal{O}\mathsf{PLoRSign}(i, m, p_0, w_{p_0}, p_1, w_{p_1})$

if $\mathsf{PC}((p_0, m), w_{p_0}) = 0 \vee \mathsf{PC}((p_1, m), w_{p_1}) = 0$
  return $\perp$
$(pk_i, sk_i, ID_i) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS})$
$((Reg[i]), (sk^i_{TPBS})) \leftarrow \mathsf{IDKeyGen}((sk^{TA}_{TPBS}) \xleftrightarrow[\sigma^i_{ID}]{(i,ID_i)} (sk_i))$
$sk^{p_0}_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p_0)$
$sk^{p_1}_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p_1)$
$\sigma_{m_b} \leftarrow \mathsf{Sign}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^{p_b}_{TPBS}, sk^i_{TPBS}, m, p_b, w_{p_b})$
$\mathcal{M} = \mathcal{M} \cup \{m, \sigma_m\}$
return $(\sigma_{m_b}, sk^{p_b}_{TPBS}, sk^i_{TPBS})$

$\mathcal{O}\mathsf{Sim\text{-}or\text{-}Sign}(i_j, p, m, w_p)$

if $\mathsf{PC}((p, m), w_p) = 1$
  $(sk^{i_j}_{TPBS}) \leftarrow \mathcal{Q}_i[i_j]$
  $sk^p_{TPBS} \leftarrow \mathsf{PolicyKeyGen}(sk^{Issuer}_{TPBS}, p)$
  $\sigma_{m_0} \leftarrow \mathsf{Sign}(pk^{TA}_{TPBS_0}, pk^{Issuer}_{TPBS_0}, sk^p_{TPBS}, sk^{i_j}_{TPBS}, m, w_p)$
  $\sigma_{m_1} \leftarrow \mathsf{SimSign}(tr_{NIZK}, pk^{TA}_{TPBS_1}, pk^{Issuer}_{TPBS_1}, m)$
  $\mathcal{M}' = \mathcal{M}' \cup \{m, \sigma_{m_b}\}$
  return $(\sigma_{m_b}, sk^p_{TPBS})$
return $\perp$

$\mathcal{O}\mathsf{Trace}(m, \sigma_m)$

if $\sigma_m \in \mathcal{M}'$ **return** $\perp$
$(i, \pi_{Trace}) \leftarrow \mathsf{Trace}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, Reg, m, \sigma_m)$
return (i, π_{Trace})

Figure 5.1: TPBS security oracles.

sage m under the identity key of $i \in \mathcal{U}$ and any policy of its choice. Furthermore, $\mathcal{O}\mathsf{USign}(.)$ returns the TA signature σ^i_{ID} of the user i to simulate the case where σ^i_{ID} is leaked without the knowledge of sk_i. Furthermore, we give the adversary access to sk^{Issuer}_{TPBS} to simulate the case of a corrupt issuer. Note, to prevent trivial attacks, the queries to $\mathcal{O}\mathsf{KeyGen}(.)$ are limited to users' identities not in $\mathcal{U}$ which models the set of honest users. Also, the adversary cannot query the $\mathcal{O}Trace$ with the output of $\mathcal{O}IdLoRSign$.

Anonymity is defined in a selfless setting where we do not provide the adversary with access to the identity keys of the two signers, $sk_{TPBS}^{i_0}$ and $sk_{TPBS}^{i_1}$, involved in the query to $\mathcal{O}IdLoRSign$ [22]. This models the case where an internal adversary should not be able to distinguish between two signatures generated under two identities different than its own, even if both signatures are generated using the same policy key. Such a restriction is essential to construct a significantly more efficient scheme [19].

Definition 9. *(TPBS Anonymity) The TPBS scheme is anonymous if for any PPT adversary* $\mathcal{A}$, $|\mathsf{Pr}[\boldsymbol{Exp}_{\mathcal{A},TPBS}^{Anonymity}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$, *where* $\boldsymbol{Exp}_{\mathcal{A},TPBS}^{Anonymity}$ *is defined in Fig. 5.2.*

$\mathbf{Exp}_{\mathcal{A},TPBS}^{Anonymity}(\lambda)$

$b \xleftarrow{\$} \{0,1\}, \mathcal{U} = \{0,\ldots,n\}, \mathcal{M}' = \{\}, \mathcal{Q}_i = [\,], pp_{TPBS} \leftarrow \mathsf{ppGen}(1^\lambda)$
$(pk_{TPBS}^{TA}, sk_{TPBS}^{TA}) \leftarrow \mathsf{TASetup}(pp_{TPBS})$
$(pk_{TPBS}^{Issuer}, sk_{TPBS}^{Issuer}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS})$
foreach $i_j \in \mathcal{U}$
  $(pk_{i_j}, sk_{i_j}, ID_{i_j}) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS})$
  $((Reg[i_j]), (sk_{TPBS}^{i_j})) \leftarrow \mathsf{IDKeyGen}((sk_{TPBS}^{TA}) \xleftarrow[\sigma_{ID}^{i_j}]{(i, ID_{i_j})} (sk_{i_j}))$
  $\mathcal{Q}_i[i_j] = sk_{TPBS}^{i_j}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(.),\mathcal{O}\mathsf{USign}(.),\mathcal{O}\mathsf{Trace}(.),\mathcal{O}\mathsf{IdLoRSign}(.,b)}(\mathcal{U}, pp_{TPBS}, pk_{TPBS}^{TA}, pk_{TPBS}^{Issuer}, sk_{TPBS}^{Issuer})$
if $b = b'$
  return $\top$
return $\bot$

Figure 5.2: TPBS anonymity experiment.

Policy-privacy. Policy-privacy is modeled by the indistinguishability experiment in Fig. 5.3, where the adversary has access to $\mathcal{O}\mathsf{KeyGen}(.)$ and $\mathcal{O}PLoRSign$ oracles. The challenge oracle $\mathcal{O}PLoRSign$ is initialized with a random bit $b \in \{0,1\}$. The adversary inputs to $\mathcal{O}PLoRSign$ oracle are $(i, m, p_0, w_{p_0}, p_1, w_{p_1})$. After verifying that $\mathsf{PC}((p_0, m), w_{p_0}) = 1$, and $\mathsf{PC}((p_1, m), w_{p_1}) = 1$, the oracle generates sk_{TPBS}^{i} and $sk_{TPBS}^{p_b}$ for $b \in \{0,1\}$. It then signs m using ($sk_{TPBS}^{p_b}$, sk_{TPBS}^{i}) and returns σ_{m_b} along with $sk_{TPBS}^{p_b}$ and sk_{TPBS}^{i}. The adversary wins if it can determine the bit b with a probability better than the random guess.

Note that we give the adversary access to sk_{TPBS}^{TA} and sk_{TPBS}^{Issuer} to simulate the case of a corrupt TA and\or issuer.

Definition 10. *(TPBS Policy-privacy) The TPBS scheme is policy-private if for any PPT adversary $\mathcal{A}$, $|\mathsf{Pr}[\boldsymbol{Exp}_{\mathcal{A},TPBS}^{Policy-privacy}(\lambda)=1]-\frac{1}{2}|\leq \epsilon(\lambda)$, where $\boldsymbol{Exp}_{\mathcal{A},TPBS}^{Policy-privacy}$ is defined in Fig. 5.3.*

$\mathbf{Exp}_{\mathcal{A},TPBS}^{Policy-privacy}(\lambda)$

$b \xleftarrow{\$} \{0,1\}, pp_{TPBS} \leftarrow \mathsf{ppGen}(1^{\lambda})$
$(pk_{TPBS}^{TA}, sk_{TPBS}^{TA}) \leftarrow \mathsf{TASetup}(pp_{TPBS})$
$(pk_{TPBS}^{Issuer}, sk_{TPBS}^{Issuer}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS})$
$(b') \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(.),\mathcal{O}\mathsf{PLoRSign}(.,.,b)}(pp_{TPBS}, pk_{TPBS}^{TA}, pk_{TPBS}^{Issuer}, sk_{TPBS}^{TA}, sk_{TPBS}^{Issuer})$
if $b = b'$
  return $\top$
return $\bot$

Figure 5.3: TPBS policy-privacy experiment.

Consider a PBS scheme where for every message m there is only one policy p such that $\mathsf{PC}((p,m),w_p)=1$; then the aforementioned policy-privacy definition can not hide the associated policy. It has been proven that simulatability is a stronger notion of policy-privacy that remedies the aforementioned limitation [12]. Since the same limitation is inherited in TPBS, thus, we also define simulatability, and we prove that our definition implies the privacy of TPBS, which is defined as both anonymity and policy-privacy.

Simulatability. This security notion requires the existence of a simulator that can create simulated signatures without having access to any of the users' signing keys or witnesses. Yet, such signatures are indistinguishable from real signatures. Thus, we assume that for every TPBS procedure, there exists a simulated procedure whose output is indistinguishable from the non-simulated one. We denote such a procedure with the Sim prefix. More precisely, we require the following algorithms, SimppGen, SimTASetup, SimIssuerSetup, SimUserKeyGenTPBS, SimIDKeyGen, SimPolicyKeyGen, SimSign, and

SimTraceTPBS. Note that SimppGen, SimTASetup, and SimIssuerSetup also output the trapdoor information tr_{NIZK}, tr_{TA}, and tr_{Issuer}, respectively. Such trapdoor outputs are used as inputs to the other relevant simulated procedures instead of the secret inputs. We give the definitions of the simulated procedures in Fig 5.9 after we present the generic construction.

We formally define simulatability in a selfless setting by the experiment in Fig. 5.4, in which the adversary has access to $\mathcal{O}$KeyGen(.), $\mathcal{O}$USign(.), $\mathcal{O}\mathsf{Trace}(.)$, and $\mathcal{O}$Sim-or-Sign(.) oracles. $\mathcal{O}\mathsf{Sim\text{-}or\text{-}Sign}(.)$ is its challenge oracle which on the input of some i_j from a predefined list of honest users identities $\mathcal{U}$, a message m, a policy p, and a witness w_p that m conforms to p, the oracle outputs a signature σ_m. The adversary wins if it can determine whether σ_m is generated using i_j identity key and p policy key or it is a simulated signature. To prevent trivial attacks, the adversary cannot query the $\mathcal{O}\mathsf{Trace}(.)$ with the signatures generated by the challenging oracle.

Definition 11. *(TPBS Simulatability) The TPBS scheme is simulatable if for any PPT adversary $\mathcal{A}$, $|\mathsf{Pr}[\boldsymbol{Exp}^{SIM}_{\mathcal{A},TPBS}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$, where the $\boldsymbol{Exp}^{SIM}_{\mathcal{A},TPBS}$ is defined in Fig. 5.4.*

Unforgeability Intuitively unforgeability is the infeasibility of creating a valid signature on a message m without holding the policy key for policy p to which m conforms.

To model users' corruption and collusion attacks where users could combine their policy keys to sign messages non of them is authorized to, Bellare and Fuchsbauer have defined the unforgeability of the PBS scheme by an experiment where the adversary is allowed to query a key generation oracle to generate user keys and gain access to some of them. However, in their definition, it becomes hard to efficiently determine if an adversary has won the unforgeability experiment by producing a valid signature such that $\mathsf{PC}((p, m), w_p) = 1$ using a queried policy key or not since policy-privacy requires hiding the policy and witness

$\mathbf{Exp}^{SIM}_{\mathcal{A},TPBS}(\lambda)$

$b \xleftarrow{\$} \{0,1\}, \mathcal{U} = \{0,\ldots,n\}, \mathcal{M}' = \{\}, \mathcal{Q}_i = [\,]$

$pp_{TPBS_0} \leftarrow \mathsf{ppGen}(1^\lambda), (pp_{TPBS_1}, tr_{NIZK}) \leftarrow \mathsf{SimppGen}(1^\lambda)$

$(pk^{TA}_{TPBS_0}, sk^{TA}_{TPBS_0}) \leftarrow \mathsf{TASetup}(pp_{TPBS_0})$

$(pk^{Issuer}_{TPBS_0}, sk^{Issuer}_{TPBS_0}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS_0})$

$(pk^{TA}_{TPBS_1}, sk^{TA}_{TPBS_1}, tr_{TA}) \leftarrow \mathsf{SimTASetup}(pp_{TPBS_1})$

$(pk^{Issuer}_{TPBS_1}, sk^{Issuer}_{TPBS_1}, tr_{Issuer}) \leftarrow \mathsf{SimIssuerSetup}(pp_{TPBS_1})$

foreach $i_j \in \mathcal{U}$

$(pk_{i_j}, sk_{i_j}, ID_{i_j}) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS})$

$((Reg[i_j]), (sk^{i_j}_{TPBS})) \leftarrow \mathsf{IDKeyGen}((sk^{TA}_{TPBS}) \xleftrightarrow[\sigma^{i_j}_{ID}]{(i,ID_{i_j})} (sk_{i_j}))$

$\mathcal{Q}_i[i_j] = sk^{i_j}_{TPBS}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(.),\mathcal{O}\mathsf{USign}(.),\mathcal{O}\mathsf{Trace}(.),\mathcal{O}\mathsf{Sim\text{-}or\text{-}Sign}(.)}(\mathcal{U}, pp_{TPBS_b}, pk^{TA}_{TPBS_b}, sk^{TA}_{TPBS_b}, pk^{Issuer}_{TPBS_b}, sk^{Issuer}_{TPBS_b})$

if $b = b'$ **return** $\top$

return $\bot$

Figure 5.4: TPBS simulatability experiment.

used in generating a specific signature. To overcome the aforementioned limitation, they defined extractability as a strengthened version of unforgeability and proved that extractability implies unforgeability [12]. Since TPBS privacy requires hiding the policy, witness, and signer's identity used in generating signatures over m, we define extractability and adapt it to imply the unforgeability for TPBS.

Extractability. We formally define TPBS extractability by the experiment in Fig. 5.5, where we assume the existence of an extractor algorithm Extr which upon inputting a valid message signature pair (m,σ_m) in addition to trapdoor information tr_{NIZK}, it outputs the tuple $(p, sk_i, sk^p_{TPBS}, w_p)$. An adversary $\mathcal{A}$ who has access to $\mathcal{O}$KeyGen and $\mathcal{O}$Sign oracles (Fig. 5.1) wins $\mathbf{Exp}^{Ext}_{\mathcal{A},TPBS}$ if it outputs a valid message signature pair (m^*, σ_{m^*}) such that either i) it does not hold some sk^{i*}_{TPBS} that is obtained from $\mathcal{O}$KeyGen oracle or for all p, it obtained sk^p_{TPBS} by querying $\mathcal{O}$KeyGen oracle, ii) it does not hold an sk^{p*}_{TPBS} corresponds to p^* such that $\mathsf{PC}((p^*, m^*), w^*_p) = 1$ or iii) $\mathsf{PC}((p^*, m^*), w^*_p) = 0$. Note that since tr_{NIZK} is required by Extr algorithm, the extractability experiment is initialized using $\mathsf{SimppGen}(1^\lambda)$

algorithm rather than $\mathsf{ppGen}(1^\lambda)$, and all other algorithms are kept the same.

Definition 12. *(TPBS Extractability) a TPBS scheme is extractable if for any PPT adversary* $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{Ext}_{\mathcal{A},TPBS}(\lambda) = 1] \leq \epsilon(\lambda)$, *where* $\boldsymbol{Exp}^{Ext}_{\mathcal{A},TPBS}$ *is defined in Fig. 5.5.*

$\mathbf{Exp}^{Ext}_{\mathcal{A},TPBS}(\lambda)$

$(pp_{TPBS}, tr_{NIZK}) \leftarrow \mathsf{SimppGen}(1^\lambda)$
$(pk^{TA}_{TPBS}, sk^{TA}_{TPBS}, Reg) \leftarrow \mathsf{TASetup}(pp_{TPBS})$
$(pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS})$
$\mathcal{Q}_i = \mathcal{Q}_p = [\,]$
$\mathcal{T} = \mathcal{L} = \mathcal{M} = \{\}$
$(m^*, \sigma_{m^*}) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(.),\mathcal{O}\mathsf{Sign}(.)}(pp_{TPBS}, pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS})$
if $(m^*, \sigma_{m^*}) \in \mathcal{M} \ \vee \ \mathsf{Verify}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m^*, \sigma_{m^*}) = \perp$
  return $\perp$
$(p^*, sk^*_i, sk^p_{TPBS}, w_{p^*}) \leftarrow \mathsf{Extr}(tr_{NIZK}, m^*, \sigma_{m^*})$
if $sk^*_i \notin \mathcal{T} \vee p^* \notin \mathcal{L} \vee \mathsf{PC}((p^*, m^*), w^*_p) = 0$
  return $\top$
return $\perp$

Figure 5.5: TPBS extractability experiment.

Non-frameability This property ensures that even if the tracing authority, issuer, and all corrupt users in the scheme collude together, they cannot produce a valid signature that is traced back to an honest user. TPBS non-frameability is modeled by the experiment defined in Fig. 5.6, in which the adversary has access to both TA and issuer secret keys $(sk^{TA}_{TPBS}, sk^{Issuer}_{TPBS})$, in addition to $\mathcal{O}\mathsf{KeyGen}$, $\mathcal{O}\mathsf{USign}$, and $\mathcal{O}\mathsf{Trace}$ oracles. The adversary wins if it outputs a verifiable (m^*, σ_{m^*}) that has not been queried to $\mathcal{O}\mathsf{USign}$ and when (m^*, σ_{m^*}) is traced back to its signer, the tracing algorithm outputs an identity of one of the honest users in $\mathcal{U}$. Additionally, the output of $\mathcal{O}\mathsf{Trace}$ oracle should be verifiable using the Judge algorithm.

Definition 13. *(TPBS Non-frameability) a TPBS scheme is non-frameable if for any PPT adversary* $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{Non-frameability}_{\mathcal{A},TPBS}(\lambda) = 1] \leq \epsilon(\lambda)$, *where the non-frameability experiment is defined in Fig. 5.6.*

$\mathbf{Exp}^{Non-frameability}_{\mathcal{A},TPBS}(\lambda)$

$\mathcal{U} = \{0, \ldots, n\}, \mathcal{M} = \{\}, \mathcal{Q}_i = [\,], pp_{TPBS} \leftarrow \mathsf{ppGen}(1^\lambda)$
$(pk^{TA}_{TPBS}, sk^{TA}_{TPBS}) \leftarrow \mathsf{TASetup}(pp_{TPBS}), (pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS})$
foreach $i_j \in \mathcal{U}$
  $(pk_{i_j}, sk_{i_j}, ID_{i_j}) \leftarrow \mathsf{UserKeyGen}(pp_{TPBS})$
  $((Reg[i_j]), (sk^{i_j}_{TPBS})) \leftarrow \mathsf{IDKeyGen}((sk^{TA}_{TPBS}) \xleftarrow[\sigma^{ij}_{ID}]{(i, ID_{i_j})} (sk_{i_j}))$
  $\mathcal{Q}_i[i_j] = sk^{i_j}_{TPBS}$
$(m^*, \sigma_{m^*}) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(.), \mathcal{O}\mathsf{USign}(.), \mathcal{O}\mathsf{Trace}(.)}(\mathcal{U}, pp_{TPBS}, pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}, sk^{TA}_{TPBS})$
if $(m^*, \sigma_{m^*}) \in \mathcal{M} \vee \mathsf{Verify}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m^*, \sigma_{m^*}) = \perp$
  return $\perp$
$(i^*, \pi^*_{Trace}) \leftarrow \mathsf{Trace}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, Reg, m, \sigma_m)$
if $i^* \notin \mathcal{U}$
  return $\perp$
return $\mathsf{Judge}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m^*, \sigma_{m^*}, i^*, \pi^*_{Trace})$

Figure 5.6: TPBS non-frameability experiment.

Traceability Traceability requires that even if all scheme users collude together, they cannot produce a signature that cannot be traced. We require the tracing authority to be honest, as knowing the secret key of the tracing authority would allow the adversary to sign a dummy sk_i under the tracing authority's secret key resulting in an untraceable signature. TPBS traceability is modeled by the experiment defined in Fig. 5.7, in which the adversary has access to $\mathcal{O}$KeyGen and $\mathcal{O}$Trace procedures. We omit the adversarial access to $\mathcal{O}$Sign oracle since the adversary could corrupt as many users as it wants and get access to their keys. Hence it could use the signing algorithm directly Sign(.) to produce signatures. The Adversary wins if it outputs a verifiable (m^*, σ_{m^*}), which when traced, the tracing algorithm Trace outputs $\perp$.

Definition 14. *(TPBS Traceability) a TPBS scheme is traceable if for any PPT adversary $\mathcal{A}$,* $\mathsf{Pr}[\boldsymbol{Exp}^{Traceability}_{\mathcal{A},TPBS}(\lambda) = 1] \leq \epsilon(\lambda)$*, where the traceability experiment is defined in Fig. 5.7.*

$\mathbf{Exp}^{Traceability}_{\mathcal{A},TPBS}(\lambda)$

$(pp_{TPBS}) \leftarrow \mathsf{ppGen}(1^\lambda), (pk^{TA}_{TPBS}, sk^{TA}_{TPBS}) \leftarrow \mathsf{TASetup}(pp_{TPBS})$
$(pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}) \leftarrow \mathsf{IssuerSetup}(pp_{TPBS})$
$(m^*, \sigma_{m^*}) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(.),\mathcal{O}\mathsf{Trace}(.)}(pp_{TPBS}, pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS})$
if $\mathsf{Verify}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m^*, \sigma_{m^*})$
  $(i^*, \pi^*_{Trace}) \leftarrow \mathsf{Trace}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, Reg, m^*, \sigma_{m^*})$
  if $i = \perp$
    return $\top$
return $\perp$

Figure 5.7: TPBS traceability experiment.

5.3 TPBS Generic Construction

The main building blocks of the new construction are a EUF-CMA RDS scheme capable of signing a commitment on a secret message, a SE-NIZK proof system, and a digital signature scheme.

Setup. The general idea of the new scheme is that in addition to the policy key sk^p_{TPBS} that is generated by the issuer using PolicyKeyGen and shared with any user who is allowed to sign a message m conforming to p, each user has to run an interactive algorithm IDKeyGen with the TA to obtain an identity key sk^i_{TPBS}. In IDKeyGen, the user gets the TA's RDS signature σ^i_{ID} on a user-chosen secret value sk_i, where sk_i is generated using UserKeyGen. σ^i_{ID} along with sk_i are the user's identity key sk^i_{TPBS}. The TA keeps track of users' registration information ID_i in a secret registry Reg. Users generate their own registration information ID_i using a one-way function over sk_i, where it contains a tracing trapdoor that allows the TA to trace the generated TPBS signature to its original signer. More precisely, ID_i contains $C_i = (c_i, \tilde{c}_i) = f(sk_i)$ and the user's digital signature τ_i over c_i where (c_i, τ_i) could be publicly mapped to such a user's identity and $\tilde{c}_i$ is held secretly by the TA as the tracing trapdoor information. To ensure non-framability, an RDS signing algorithm is used

to sign the output of a one-way function c_i rather than sk_i itself, yet the produced signature is verifiable over sk_i.

Signing. To sign a message m, the user generates a rerandomized version of the TA signature ${\sigma'}^i_{ID}$ along with a SE-NIZK proof π_m for the relation $\mathbb{R}'_{\mathbb{NP}}$ that is given by

$$((pk^{TA}_{TPBS}, {\sigma'}^i_{ID}, pk^{Issuer}_{TPBS}, m), (sk_i, p, sk^p_{TPBS}, w_p)) \in \mathbb{R}'_{\mathbb{NP}} \Leftrightarrow$$

$$\mathsf{VerifyRDS}(pk^{TA}_{TPBS}, sk_i, {\sigma'}^i_{ID}) = 1 \tag{5.1a}$$

$$\wedge\ \mathsf{VerifySig}(pk^{Issuer}_{TPBS}, p, sk^p_{TPBS}) = 1 \tag{5.1b}$$

$$\wedge\ \mathsf{PC}((p, m), w) = 1, \tag{5.1c}$$

whose statements $X = (pk^{TA}_{TPBS}, {\sigma'}^i_{ID}, pk^{Issuer}_{TPBS}, m)$ with witnesses $W = (sk_i, p, sk^p_{TPBS}, w_p)$. Intuitively, π_m proves that a) ${\sigma'}^i_{ID}$ is the TA signature over some signer-generated secret value sk_i, b) the user holds the issuer's signature over some policy p, and c) the message m conforms the policy p under some witness w_p, i.e. $\mathsf{PC}((p, m), w) = 1$.

Verifying and tracing. Signature verification is done by verifying π_m over the statements X. To trace a signature to its signer, the TA exhaustively searches Reg for a matching $\tilde{c}_i$ that verifies ${\sigma'}^i_{ID}$ in the signature; once a match is found, the TA outputs the resisted user's identity i along with (c_i, τ_i) in addition to a NIZK for the knowledge of $\tilde{c}_i$ such that c_i and $\tilde{c}_i$ are generated $f(sk_i)$ and ${\sigma'}^i_{ID}$ is verifiable over sk_i.

One advantage of using a sign-rerandomize-proof paradigm rather than a sign-encrypt-proof paradigm is that the former paradigm produces a significantly more efficient signature than the latter [19, 76]. On the other hand, the tracing algorithm becomes a linear operation in the number of scheme users, which is considered an affordable price since tracing is an infrequent operation and is run by a computationally powerful TA [19].

Figure 5.8 depicts the complete generic construction of TPBS. Note that we use two different instances of the digital signature scheme. The issuer uses one to sign a policy p in PolicyKeyGen, and the scheme users use the other to sign the output of the one-way function to generate ID_i in UserKeyGen. We label the latter with the subscript Σ.

Figure 5.8: TPBS generic construction.

In Fig. 5.9, we show how SimppGen(.), SimSign(.), Extr(.) are constructed in accordance with the concrete construction in Fig. 5.8. Since tr_{TA}, and tr_{Issuer} is equal to sk^{TA}_{TPBS} and sk^{Issuer}_{TPBS}, respectively, we omit the details of SimTASetup(.), SimIssuerSetup(.), SimUserKeyGenTPBS(.), SimIDKeyGen(.), SimPolicyKeyGen(.), and Sim-

Trace(.) which are defined in the same way as TASetup(.), IssuerSetup(.), IDKeyGen(.), PolicyKeyGen(.), and Trace(.), respectively, .

$\mathsf{SimppGen}(1^\lambda)$

$(crs, tr_{NIZK}) \leftarrow \mathsf{SimSetupNIZK}(1^\lambda), pp_{RDS} \leftarrow \mathsf{ppGenRDS}(1^\lambda)$
$pp_{Sig} \leftarrow \mathsf{ppGenSig}(1^\lambda), pp_{Sig_\Sigma} \leftarrow \mathsf{ppGenSig}(1^\lambda)$
return $pp_{TPBS} \leftarrow (crs, pp_{RDS}, pp_{Sig}, pp_{Sig_\Sigma}),\ tr_{NIZK}$

$\mathsf{SimSign}(tr_{NIZK}, (pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, m))$

$sk'_i \xleftarrow{\$} \mathbb{Z}^*_p, \sigma^i_{ID} \leftarrow \mathsf{SignRDS}(sk^{TA}_{TPBS}, sk'_i)$
$\sigma'^i_{ID} \leftarrow \mathsf{RandomizeRDS}(\sigma^i_{ID})$
$\pi_m \leftarrow \mathsf{SimProve}(crs, tr_{NIZK}, (pk^{Issuer}_{TPBS}, pk^{TA}_{TPBS}, \sigma'^i_{ID}, m))$
return $\sigma_m = (\sigma'^i_{ID}, \pi_m)$

$\mathsf{Extr}(tr_{NIZK}, m, \sigma_m)$

$(\sigma'^i_{ID}, \pi_m) = \sigma_m$
$(p, sk_i, sk^p_{TPBS}, w_p) \leftarrow \mathsf{Extr}_{\mathsf{NIZK}}(crs, tr_{NIZK}, m, \pi_m)$
return $(p, sk_i, sk^p_{TPBS}, w_p)$

Figure 5.9: TPBS simulated algorithms.

5.4 TPBS Security

The definition of extractability of TPBS (see Def. 12) implies its unforgeability. The privacy of TPBS includes policy privacy and anonymity. Accordingly, we first prove that simulatability implies both anonymity and policy-privacy. Then we present the security proofs for simulatability (implies privacy), extractability (implies unforgeability), non-frameability, and traceability.

Theorem 6. *Simulatability implies both anonymity and policy-privacy*

Proof. Assuming an adversary $\mathcal{A}$ against TPBS anonymity in $\mathbf{Exp}^{Anonymity}_{\mathcal{A},TPBS}$ in Fig. 5.2 (resp. policy-privacy in $\mathbf{Exp}^{Policy-privacy}_{\mathcal{A},TPBS}$ in Fig. 5.3), we can construct an adversary $\mathcal{B}$ (resp. $\mathcal{B}'$) against the simulatability of TPBS. $\mathcal{B}$ receives ($\mathcal{U}$, pp_{TPBS_b}, $pk^{TA}_{TPBS_b}$, $sk^{TA}_{TPBS_b}$, $pk^{Issuer}_{TPBS_b}$, $sk^{Issuer}_{TPBS_b}$) from its challenger in the $\mathbf{Exp}^{SIM}_{\mathcal{A},TPBS}$ in Fig. 5.4, chooses $d \xleftarrow{\$} \{0,1\}$, and runs $\mathcal{A}$ on ($\mathcal{U}$, pp_{TPBS_b},$pk^{TA}_{TPBS_b}$,$pk^{Issuer}_{TPBS_b}$,$sk^{Issuer}_{TPBS_b}$). Whenever $\mathcal{A}$ queries its challenging oracle $\mathcal{O}$IdLoRSign with $(i_{j_0}, i_{j_1}, m, p, w_p)$, if $\mathsf{PC}((p,m), w_p) = 0$ or $i_{j_0}, i_{j_1} \notin \mathcal{U}$, $\mathcal{B}$ returns $\perp$, otherwise it queries its challenger in the simulatability game with (i_{j_d}, m, p, w_p) and returns

$(\sigma_{m_b}, sk^p_{TPBS})$ to $\mathcal{A}$. When $\mathcal{A}$ outputs b', $\mathcal{B}$ outputs 0 if $(b' = d)$, indicating that $\mathcal{A}$ returned the identity $\mathcal{B}$ queried $\mathcal{O}$Sim-or-Sign with; thus σ_{m_b} is not a simulated signature. $\mathcal{B}$ outputs 1 otherwise. $\mathcal{B}'$ could be constructed similarly as follows. It receives $(\mathcal{U}, pp_{TPBS_b}, pk^{TA}_{TPBS_b}, sk^{TA}_{TPBS_b}, pk^{Issuer}_{TPBS_b}, sk^{Issuer}_{TPBS_b})$ its challenger in the simulatability game in Fig. 5.4, chooses $d \xleftarrow{\$} \{0,1\}$, and runs $\mathcal{A}$ on $(pp_{TPBS_b}, pk^{TA}_{TPBS_b}, sk^{TA}_{TPBS_b}, pk^{Issuer}_{TPBS_b}, sk^{Issuer}_{TPBS_b})$. Whenever $\mathcal{A}$ queries its challenge oracle $\mathcal{O}$PLoRSign with $(i, m, p_0, w_{p_0}, p_1, w_{p_1})$, if $\mathsf{PC}((p_0, m), w_{p_0}) = 0$ or $\mathsf{PC}((p_1, m), w_{p_1}) = 0$ or $i \notin \mathcal{U}$, $\mathcal{B}'$ returns $\perp$, otherwise it quires its challenger in the simulatability game with (i, m, p_d, w_{p_d}) and returns $(\sigma_{m_b}, sk^p_{TPBS})$ to $\mathcal{A}$. When $\mathcal{A}$ outputs b', $\mathcal{B}'$ outputs 0 if $(b' = d)$ and 1 otherwise. In either case, if in $\mathbf{Exp}^{SIM}_{\mathcal{B},TPBS}(\lambda)$ (resp. $\mathbf{Exp}^{SIM}_{\mathcal{B}',TPBS}(\lambda)$) the challenger's bit is 0 indicating a signed signature, then $\mathcal{B}$ (resp. $\mathcal{B}'$) perfectly simulates $\mathbf{Exp}^{Anonymity}_{\mathcal{A},TPBS}(\lambda)$ (resp. $\mathbf{Exp}^{Policy-privacy}_{\mathcal{A},TPBS}(\lambda)$) for $\mathcal{A}$. However, if the bit is 1 indicating a simulated signature, then the bit d chosen by $\mathcal{B}$ (resp. $\mathcal{B}'$) has no relation to $\mathcal{A}$'s response. Hence, $\mathcal{B}$ outputs 1 with probability $\frac{1}{2}$. Therefore, the success probability of $\mathcal{B}$ (resp. $\mathcal{B}'$) is half that of A in the anonymity (resp. policy-privacy) experiment. □

Theorem 7. *Given a zero-knowledge simulation-sound extractable NIZK system and an unlinkable RDS scheme, the traceable policy-based signature scheme in Fig. 5.8 is simulatable.*

Proof. (Sketch) Recall that for an adversary $\mathcal{A}$ to win the simulatability game $\mathbf{Exp}^{Sim}_{\mathcal{A},TPBS}$ in Fig. 5.4, it has to guess the bit 'b' that $\mathcal{O}$Sim-or-Sign$(.)$ is initialized with. In other words, $\mathcal{A}$ wins if it can determine whether the output signature $\sigma_m = (\sigma'^i_{ID}, \pi_m)$ of $\mathcal{O}$Sim-or-Sign$(.)$ is generated using the secret keys or it is a simulated signature. It follows that $\mathcal{A}$ succeeds if it can either distinguish π_m from a simulated one thus breaking the zero-knowledge of the utilized SE-NIZK or by distinguishing the rerandomized RDS signature σ'^i_{ID} through linking it to its original version σ^i_{ID}. Therefore, the simulatability of TPBS follows from a

zero-knowledge SE-NIZK system and an unlinkable RDS scheme.

□

Theorem 8. *Given an unforgeable RDS scheme, an unforgeable signature scheme, and a simulation-extractable NIZK system, the traceable policy-based signature scheme in Fig. 5.8 is extractable.*

Proof. (Sketch) Recall that for adversary $\mathcal{A}$ to win the extractability game $\mathbf{Exp}^{Ext}_{\mathcal{A},TPBS}$ in Fig. 5.5, it has to output a verifiable (m^*, σ_{m^*}) where m^* has never been queried to the $\mathcal{O}\mathsf{Sign}$ oracle and when the Extr algorithm is run over (m^*, σ_{m^*}) using the trapdoor information tr_{NIZK}, the returned $(p^*, sk_i^*, sk_{TPBS}^{p^*}, w_{p^*})$ satisfies any of the following conditions: i) $sk_i^* \notin \mathcal{T}$, which implies that the adversary has never obtained some sk_{TPBS}^i through $\mathcal{O}\mathsf{KeyGen}$ where $sk_i = sk_i^*$, or ii) $p^* \notin \mathcal{L}$, which implies that the adversary has not queried $\mathcal{O}\mathsf{KeyGen}$ with p^*, or iii) $\mathsf{PC}((p^*, m^*), w_p^*) = 0$. It follows that $\mathcal{A}$ succeeds if it can either i) forge the TA's RDS signature over sk_i^*, or ii) forge the issuer's digital signature over p^* or iii) generate a NIZK proof for a false statement thus breaking the simulation-extractability of the utilized SE-NIZK. Therefore, the extractability of TPBS follows from the EUF-CMA of the RDS scheme, the EUF-CMA of the underlying digital signature scheme, and the SE of the NIZK system.

□

Theorem 9. *Given a one-way function, an interactive perfect zero-knowledge proof of knowledge, an unforgeable digital signature scheme, and a zero-knowledge SE-NIZK system, the traceable policy-based signature scheme in Fig. 5.8 is non-frameable.*

Proof. (Sketch) Recall that for an adversary $\mathcal{A}$ to win $\mathbf{Exp}^{Non-Frameability}_{\mathcal{A},TPBS}$ in Fig. 5.6; it has to output (m^*, σ_{m^*}) when it is traced back, the tracing result (i^*, π^*_{Trace}) points to some honest signer whose signing keys were not shared with $\mathcal{A}$ and yet such a tracing result is accepted

by Judge algorithm. Without loss of generality, let an adversary $\mathcal{A}$ win $\mathbf{Exp}^{Non-Frameability}_{\mathcal{A},TPBS}$ for user identity $i_z \in \mathcal{U}$. Here we distinguish between two tracing results; i) $i^* = i_z$ and $ID_{i^*} = Reg[i_z]$, which implies that $\mathcal{A}$ has successfully revealed sk_{i_z} from an earlier signature by the user i_z, and ii) $i^* = i_z$ and $ID_{i^*} \neq Reg[i_z]$ that the honest user i_z has used in $\mathsf{IDKeyGen}(.)$ algorithm which implies $\mathcal{A}$ has manipulated Reg. It follows that $\mathcal{A}$ succeeds if it can either i) reveal sk_{i_z} by a) breaking the zero-knowledge property of the utilized NIZK system so that it can recover sk_{i_z} from an earlier signature by the user i_z b) breaking the one-wayness of $f(.)$ and recovering sk_{i_z} from C_i so that a corrupt TA could recover sk_{i_z} used in $\mathsf{IDKeyGen}(.)$, c) breaking the zero-knowledge of the underlying proof of knowledge $\mathsf{PoK}(sk_i : C_i = f(sk_i))$ so that a corrupt TA could recover sk_{i_z} used in $\mathsf{IDKeyGen}(.)$, or ii) forge the user's digital signature included in $Reg[i_z]$ so that a corrupt TA can produce a tracing result that identifies an honest user as the origin of a signature they did not produce. Therefore, given a one-way function $f(.)(.)$, and an interactive perfect zero-knowledge proof of knowledge PoK, a non-frameable TPBS follows from a zero-knowledge SE-NIZK system and a EUF-CMA digital signature scheme.

□

Theorem 10. *Given an unforgeable RDS scheme and a simulation-extractable NIZK system, the traceable policy-based signature scheme in Fig. 5.8 is traceable.*

Proof. (Sketch) Recall that for an adversary to win $\mathbf{Exp}^{Traceability}_{\mathcal{A},TPBS}$ in Fig. 5.7; it has to output (m^*, σ_{m^*}) where the produced signature cannot be traced to some signer whose signing keys are obtained through $\mathcal{O}\mathsf{KeyGen}(.)$ oracle. In other words, for adversary $\mathcal{A}$ to win, i) it should have access to a verifiable σ^i_{ID} under pk^{TA}_{TPBS} that has been obtained without calling the $\mathcal{O}\mathsf{KeyGen}(.)$ oracle or ii) adversary $\mathcal{A}$ has succeeded in generating NIZK proof for a false statement such that $\mathsf{VerifyRDS}(pk^{TA}_{TPBS}, sk^*_i, \sigma'^{i^*}_{ID}) = 0$. It follows that $\mathcal{A}$ succeeds if it can

either i) forge the TA's RDS signature to obtain σ^i_{ID}, or ii) break the simulation-extractability of the utilized SE-NIZK so it can generate a verifiable NIZK for a false statement. Therefore, a traceable TPBS follows from an EUF-CMA RDS scheme and a SE-NIZK system.

□

5.5 TPBS Instantiation and Performance

We instantiate TPBS with Pointcheval-Sanders (PS) RDS Scheme [76,77][1] in Appendix A.2 because of its short signature size and low signing cost in addition to its ability to sign a hiding commitment over a message using a special form of its signing algorithm. we consider the One-way function $f(.)$ over a type-3 bilinear group map defined by $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ where the SDH assumption holds to be simply the function $f(sk_i) = (g^{sk_i}, \tilde{g}^{sk_i})/$ for $(g, \tilde{g}) \in (\mathbb{G}, \tilde{\mathbb{G}})$ and $sk_i \in \mathbb{Z}^*_p$. We instantiate the issuer digital signature algorithm with the structure-preserving signature scheme in Appendix A.2 of Abe *et al.* [2]. we instantiate the SE-NIZK scheme with the Groth-Sahai proof system [58]. Moreover, any digital signature scheme can be utilized for the user's digital signature algorithm. However, we keep it as a black box since it is not utilized in TPBS signature generation or verification. Finally, we instantiate the PoK with the four-move perfect zero-knowledge protocol of Cramer *et al.* [41]. Furthermore, we keep the original definition of Bellare and Fuchsbauer for a policy p that defines a set of Pairing Product Equations (PPEs) $(E_1, \ldots, E_n)$, such that the policy checker $\mathsf{PC}((p, m), w_p) = 1$ iff $E_j((p, m), w_p) = 1$ for all $j \in [n]$.

ppGen. for a security parameter λ, let $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$ defines a type-3 bilinear group map that is generated by $(g, \tilde{g})$ that is used by all the scheme algorithms, Run $pp_{Sig} \leftarrow$

[1] PS scheme has two variants one is based interactive assumption to prove its security [76] and a slightly modified one [77] where its security is proved based on the SDH assumption both could be used to instantiate our scheme

$\mathsf{ppGenAbe}(1^\lambda)$, $pp_{Sig_\Sigma} \leftarrow \mathsf{ppGenSig}(1^\lambda)$, $pp_{RDS} \leftarrow \mathsf{ppGenPS}(1^\lambda)$, and $crs \leftarrow \mathsf{SetupGS}$. Set $pp_{TPBS} = \{crs, pp_{RDS}, pp_{Sig}, pp_{Sig_\Sigma}\}$, where pp_{TPBS} becomes an implicit input for all TPBS algorithms.

TASetup. $(pk^{TA}_{TPBS}, sk^{TA}_{TPBS}) \leftarrow \mathsf{KeyGenPS}(pp_{RDS})$ such that $pk^{TA}_{TPBS} = (g_1, \tilde{A}, \tilde{B})$, $sk^{TA}_{TPBS} = (a, b)$. Setup an empty $Reg = [\,]$.

IssuerSetup. $(pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}) \leftarrow \mathsf{KeyGenAbe}(pp_{Abe})$ such that $pk^{Issuer}_{TPBS} = (U, V, H, Z)$, and $sk^{Issuer}_{TPBS} = (u, v, h, z)$ for $U \in \mathbb{G}$, $(V, H, Z) \in \tilde{\mathbb{G}}$ and $(u, v, h, z) \in \mathbb{Z}^*_p$

UserKeyGen. Generates $(pk^i_{Sig_\Sigma}, sk^i_{Sig_\Sigma}) \leftarrow \mathsf{KeyGenSig}(pp_{Sig_\Sigma})$, sets $\mathcal{D}[i] = (pk^i_{Sig_\Sigma})$, picks $sk_i \xleftarrow{\$} \mathbb{Z}^*_p$, calculates $C_i = (c_i, \tilde{c}_i) = (g_1^{sk_i}, \tilde{B}^{sk_i})$, generates $\tau_i \leftarrow \mathsf{SignSig}(c_i, sk^i_{Sig_\Sigma})$, sets $ID_i = \{C_i, \tau_i\}$, finally return (pk_i, sk_i, ID_i).

IDKeyGen. The user sends (i, ID_i) to the TA, the TA parses ID_i as $\{(c_i, \tilde{c}_i), \tau_i\}$ and obtains an authentic copy of $pk^i_{Sig_\Sigma}$, if $Reg[i] = \emptyset \wedge \mathsf{VerifySig}(pk^i_{Sig_\Sigma}, c_i, \tau_i) \wedge e(c_i, \tilde{B}) = e(g_1, \tilde{c}_i)$, the TA engages with the user to start the interactive zero-knowledge protocol $\mathsf{PoK}(sk_i : c_i = g_1^{sk_i})$, if TA verifies that the user knows sk_i such that the relation of PoK holds, the TA generates $\sigma^i_{ID} \leftarrow \mathsf{SignComPS}(sk^{TA}_{TPBS}, c_i)$ as follows, the TA picks $r \xleftarrow{\$} \mathbb{Z}^*_p$ and generates $\sigma^i_{ID} = (\sigma^i_{ID_1}, \sigma^i_{ID_2}) \leftarrow (g_1^r, (g_1^a (c_i)^b)^r$, finally the TA sets $Reg[i] = ID_i$ and the user set his scheme identity key as $sk^i_{TPBS} = (sk_i, \sigma^i_{ID})$.

PolicyKeyGen. For policy $p \in \{0,1\}^*$, which is presented by a set of PPE equations $(E_1, \ldots, E_n)$ for a number of secret group elements $(M, \tilde{N}) \in \mathbb{G}^{k_M} \times \tilde{\mathbb{G}}^{k_N}$, the issuer generates $sk^p_{TPBS} \leftarrow \mathsf{SignAbe}(sk^{Issuer}_{TPBS}, (M, \tilde{N}))$ such that $sk^p_{TPBS} = (R, S, T)$.

Sign. To sign a message m, the signer first generates a rerandomized version of σ^i_{ID}, ${\sigma'}^i_{ID} \leftarrow \mathsf{RandomizePS}(\sigma^i_{ID})$, along with a SE-NIZK proof π_m for relation $\mathbb{R}'_{\mathbb{NP}}$ that is defined in 5.1

as follows

$$((pk^{TA}_{TPBS}, \sigma'^{i}_{ID}, pk^{Issuer}_{TPBS}, m), (sk_i, p, sk^{p}_{TPBS}, w_p)) \in \mathbb{R}'_{\mathbb{NP}} \Leftrightarrow$$

$$e(\sigma'_{ID1}, \tilde{A})e(\sigma'_{ID1}, \tilde{K}) = e(\sigma'_{ID2}, \tilde{g}) \wedge e(g, \tilde{K}) = e(g^{sk_i}, \tilde{B}) \tag{5.1a}$$

$$\wedge\, e(R, V)e(S, \tilde{g})e(M, H) = e(g, Z) \wedge e(R, T)e(U, N) = e(g, \tilde{g}) \tag{5.1b}$$

$$\wedge\, E_j(((M, \tilde{N}), m), (W_p, \tilde{W}_p)) = 1\ \forall j \in [n] \tag{5.1c}$$

Verify. To verify a message signature pair (m, σ_m), the verifier parses $(\sigma'^{i}_{ID}, \pi_m)$ from σ_m and runs $\mathsf{VerifyNIZK}(crs, (pk^{TA}_{TPBS}, \sigma'^{i}_{ID}, pk^{Issuer}_{TPBS}, m), \pi_m)$. Finally, the verifier outputs $\top$ in case of verification success and $\perp$ otherwise.

Trace. To trace a message signature pair (m, σ_m) to its original signer, the TA verifies such pair. If the verification succeeds, it parses $(\sigma'^{i}_{ID}, \pi_m)$ from σ_m and exhaustively searches Reg for a matching i as follows.

foreach $\tilde{c}_i \in Reg$

if $e(\sigma'_{ID_2}, \tilde{g})e(\sigma'_{ID_1}, \tilde{A})^{-1} = e(\sigma'_{ID_1}, \tilde{c}_i)$

return (i, ID_i)

$\pi \leftarrow \mathsf{ProveNIZK}(crs, (c_i, \sigma'^{i}_{ID}), \tilde{c}_i) \ni e(\sigma'_{ID_2}, \tilde{g})e(\sigma'_{ID_1}, \tilde{A})^{-1} = e(\sigma'_{ID_1}, \tilde{c}_i)$

$$\wedge\, e(c_i, \tilde{B}) = e(g_1, \tilde{c}_i)$$

$\pi_{Trace} \leftarrow (c_i, \tau_i, \pi)$

return (i, π_{Trace})

Judge. After verifying (m, σ_m), parses (c_i, τ_i, π) from π_{Trace} and outputs $\top$ if $\mathsf{VerifySig}$ $(pk^{i}_{Sig_\Sigma}, c_i, \tau_i) \wedge \mathsf{VerifyNIZK}$ $(crs, (c_i, \sigma'^{i}_{ID}), \pi))$ or $\perp$ otherwise.

Performance Analysis. To sign a message, the signer needs to perform a rerandomization for an RDS signature and calculate a SE-NIZK proof for relation 5.1. To verify a message

signature pair, the verifier verifies a SE-NIZK proof for relation 5.1. Tracing a signature to its origin requires verifying the message signature pair, exhaustively searching the registry for matching registration information, and calculating a SE-NIZK proof for the search result. Finally, to verify the output of the tracing algorithm, the Judge procedure verifies the message signature pair, the user's digital signature on the registration information, and the proof generated by the tracing algorithm. Concretely, let the TPBS be initialized with n users and the policy p be expressed in 1 PPE uniquely defined by $(M, \tilde{N}) \in \mathbb{G} \times \tilde{\mathbb{G}}$ group elements. To sign a message m that conforms to p, The proposed instantiation produces a total signature size of 14 elements in $\mathbb{G}$ + 16 elements in $\tilde{\mathbb{G}}$, where ${\sigma'}^{i}_{ID}$ is a PS signature of size 2 elements in $\mathbb{G}$, and π_m is a Groth-Sahai proof of knowledge of size 12 elements in $\mathbb{G}$ + 16 elements $\tilde{\mathbb{G}}$. Signing costs two exponentiations in $\mathbb{G}$ to generate ${\sigma'}^{i}_{ID}$ and approximately 40 exponentiations in $\mathbb{G}$ + 70 exponentiations in $\tilde{\mathbb{G}}$ to produce π_m. Verifying a given TPBS message signature pair costs approximately a total of 100 pairing operations to verify ${\pi_m}$[2]. For tracing a signature, the TA performs at most $3n$ pairing operations and produces a proof π of size 16 group elements in $\tilde{\mathbb{G}}$, which costs around 10 exponentiations in $\mathbb{G}$ and 20 exponentiations in $\tilde{\mathbb{G}}$. To verify the output of the tracing algorithm, the Judge performs around 40 pairing operations to verify π in addition to the verification cost of the TPBS signature and the verification cost of the signature τ_i of the user on the registration information.

5.6 Comparisson with PBS and Xu *et al.*'s Schemes

TPBS builds on PBS and further provides traceability and non-frameability. Accordingly, in addition to the issuer in PBS, TPBS has a TA that can trace signatures back to their signers.

[2]The verification cost of Groth-Sahai proofs could be enhanced using batch verification [21]

Non-frameability of TPBS holds under the assumption of a misbehaving TA. TPBS black-box construction has four new algorithms when compared to PBS. Namely, UserKeyGen, and IDKeyGen, where the latter is run interactively between each scheme user and the TA to generate such user's identity key, Furthermore, we introduce the Trace, and Judge algorithms. where Trace algorithm is used by the TA to trace a signature to its original signer and Judge algorithm is used to verify the output of the Trace algorithm. The security model of TPBS differs from that of PBS in that it includes formal definitions for traceability and non-frameability and, the definitions of simulatability and extractability capture the introduced notion of signer anonymity and identity features.

Xu *et al.* also builds on PBS by attaching the user's identity to the hidden policy and utilizing a sign-encrypt-proof paradigm to provide the traceability feature. On the other hand, TPBS utilizes sign-rerandomize-proof which produces more efficient signatures than the sign-encrypt-proof paradigm used in Xu *et al.*'s proposal. TPBS separates identity keys from policy keys, thus it supports the delegation of policy keys in the same way as PBS which is not applicable in Xu *et al.*'s proposal. The issuer Xu *et al.*'s scheme generates the signing keys of the user, thus, it does not ensure non-frameability. However, in TPBS the scheme users generate their own identity keys using an interactive protocol with the TA, hence TPBS provides non-frameability. Xu *et al.*'s proposal does not give a formal definition for traceability.

Table 5.1 summarizes the comparison between TPBS, PBS, and Xu *et al.*'s proposal. We consider the utilized building blocks and the availability of the traceability feature. If traceability is ensured by a scheme, then we contrast the schemes in terms of how the signer identity is utilized. We also consider the structure of the TA, whether a scheme enables the delegation of signing keys, and finally what security definitions are considered in the

scheme's security model.

Table 5.1: Comparison between TPBS, PBS and Xu *et al.*'s proposal.

	TPBS (this work)	PBS [12]	Xu *et al.* [96]
Building blocks	RDS SE-NIZK digital Sig.	SE-NIZK digital Sig.	encryption scheme SE-NIZK digital Sig.
Traceability	yes	no	yes
Identity	identity key	N/A	attached to the policy
Tracing Authority	standalone	N/A	issuer acts as the TA
Delegatability	yes	yes	no
Security definitions	simulatability extractability traceability non-frameability	simulatability extractability	simulatability extractability

N/A denotes an unavailable feature/entity.

Chapter 6

Extended Policy-Based Sanitizable Signatures

In this chapter, we propose an Extended Policy-Based Sanitizable Signature Scheme (EP3S), which enhances the signer's control over the modification policy by specifying who can sanitize a message, which parts of the message can be modified, and what specific modifications are allowed. Additionally, EP3S introduces multiple modification policies for the same message, enabling different sanitizers to perform distinct modifications. Furthermore, EP3S supports the delegation of sanitization rights, allowing authorized sanitizers to transfer their rights to others with additional restrictions, enhancing flexibility and practicality in real-world applications. This work is published in Inscrypt 2024 [6]

We present a generic construction of EP3S, utilizing a traceable policy-based signature scheme (TPBS, Chapter 5) and ciphertext-policy attribute-based encryption (CP-ABE) [55] scheme as its building blocks. We formally prove that the generic construction satisfies the essential security properties of policy-based sanitizable signature schemes (PB3S). Furthermore, we propose a concrete instantiation of EP3S, analyze its efficiency, and compare it

with existing PBSS schemes in terms of security, functionality, and performance.

Compared to P3S [89] and UP3S (Chapter 4), which allow sanitizers to modify specific message blocks but do not regulate how those blocks can be modified, EP3S introduces fine-grained modification control. It also eliminates the limitation of static sanitizer assignment, allowing for dynamic delegation of sanitization rights without requiring the signer to predict all potential sanitizers in advance.

Throughout this chapter, we use $m = (m_1, m_2, \ldots, m_l) \in \mathbb{M}^l$ to denote a message m containing l blocks each of size n bits where m_i denotes a message block. Let p_{San} denote the modification policy set by the signer for a message m. The function $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$ is used to modify the message m by applying the modifications mod in accordance to p_{San} and outputs the modified message m'. Furthermore, we write $\mathsf{CheckMoD}$ $(m, p_{San}) = 1$, if the message m conforms to the modification policy p_{San}, i.e., $\mathsf{PC}(p_{San}, m) = 1$ (see Def. 8). For each message m, there is one signer and one or more sanitizer(s) who can sanitize m by running $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$ and generating a valid sanitized signature on m' depending on their attributes. Finally, to denote that an attribute set $\mathbb{S}$ satisfies a monotone access structure $\mathbb{A}$ (see Def. 1), we use $\mathbb{A}(\mathbb{S}) = 1$.

6.1 Extended Policy-Based Sanitizable Signatures (EP3S)

In what follows, we give the generic construction of our proposal where we keep it in line with the black-box construction in (Chapter 4). Then, we show how a policy checker (Def. 8) is built to achieve the basic requirements of a sanitizable signature scheme and how we extend it to achieve fine-grained control over the modification policy. Finally, we show how $\mathsf{EP3S}$ could be extended to support both modification policy delegation and multiple

modification policy definitions for the same message.

6.2 EP3S Construction

In the generic construction for EP3S, we utilize two main building blocks, a TPBS scheme, and a CP-ABE scheme. The generic construction of EP3S scheme is depicted in Fig. 6.1. Once EP3S is initialized, signers and sanitizers can generate their keys using KGenSign and KGenSan algorithms where the signer's key is composed of a TPBS issuer key sk_{TPBS}^{Issuer} in addition to a TPBS signer key sk_{TPBS}^{Sign}. On the other hand, the sanitizer's key is composed of a TPBS signer key sk_{TPBS}^{San} plus an ABE key $sk_{ABE}^{\mathbb{S}_{San}}$ that corresponds to the attributes they possess where such a key is received from the ABE attribute authority (the entity holding the ABE master secret key msk_{ABE}).

Signing. To sign a given message, the signer first specifies the message m, the modification policy p_{San}, and a monotone access structure $\mathbb{A}_{San}$ based on the attributes that future sanitizers possess. Then using the Sign algorithm, the signer first generates the modification policy secret key $sk_{TPBS}^{p_{San}}$ using the TPBS issuer secret key, encrypts $p_{San}||sk_{TPBS}^{p_{San}}$ under $\mathbb{A}_{San}$ and publishes the encryption result $c_{p_{San}}$ along with the hashing result of the message $H(m)$ in an append-only public registry $C_{p_{San}}$ uniquely identified by ID=$H(m||p_{San})$ as an identifier. Finally, the signer generates a TPBS signature σ_m over m using $sk_{TPBS}^{p_{San}}$ and the TPBS signer secret key sk_{TPBS}^{Sign}.

While it might seem more direct to incorporate $C_{p_{San}}$ within the generated signature to distribute the modification policy keys, this approach would hinder the delegatability of the modification policy key (refer to section 6.2 for details). The reason is that to maintain the unforgeability of our scheme, the signer would need to sign $C_{p_{San}}$ before including it in

the signature, meaning any alteration to $C_{p_{San}}$ by the sanitizer would invalidate the entire signature. An alternative is to use a sign-crypt attribute-based encryption (ABE) scheme to sign and encrypt $p_{San}||sk_{TPBS}^{p_{San}}$, along with embedding the access structure $\mathbb{A}$ within the message itself as $\mathbb{A}||m$. However, this approach would not only introduce additional overheads to the scheme but would also necessitate the signer to identify all potential future sanitizers and their possible delegates. We choose to outsource $C_{p_{San}}$ to a public registry as a more straightforward strategy that offers unlinkability without impacting the unforgeability of the scheme. Additionally, identifying the entries in the register by $H(m||p_{San})$ enables signers to sign identical messages multiple times using the same modification policy or distinct modification policies (see section 6.2).

Sanitizing. A sanitizer who holds an ABE secret key $sk_{ABE}^{\mathbb{S}_{San}}$ that corresponds to a set of attributes $\mathbb{S}_{San}$ that satisfy $\mathbb{A}_{San}$ specified by the signer of a message m. Using the Sanitize algorithm, the sanitizer first looks up $H(m)$ in $C_{p_{San}}$ to retrieve $c_{p_{San}}$, decrypts $c_{p_{San}}$ to obtain $p_{San}||sk_{TPBS}^{p_{San}}$[1]. Then the sanitizer modifies the message m in accordance to p_{San} to obtain m'. Finally, the sanitizer generates a TPBS signature $\sigma_{m'}$ over m' using $sk_{TPBS}^{p_{San}}$ and his own TPBS signer secret key where $\sigma_{m'}$ could be verified under the signer issuer's public key.

Verifying and tracing. Verifying a message signature pair is straightforward, where σ_m and $\sigma_{m'}$ could be verified under the signer issuer's public key using the Verify algorithm. To trace a message signature pair to its original signer, the tracing function of the underlying TPBS scheme is utilized in the Prove algorithm and then the Judge algorithm attests whether the output of Prove is valid or not.

[1]In case $H(m)$ has multiple entries in the public registry, the sanitizer attempts to decrypt all the corresponding $c_{p_{San}}$, ignores all entries where DecryptABE(.) returns $\perp$ and only consider the entry(ies) that has been successfully decrypted in accordance to the attribute it possesses.

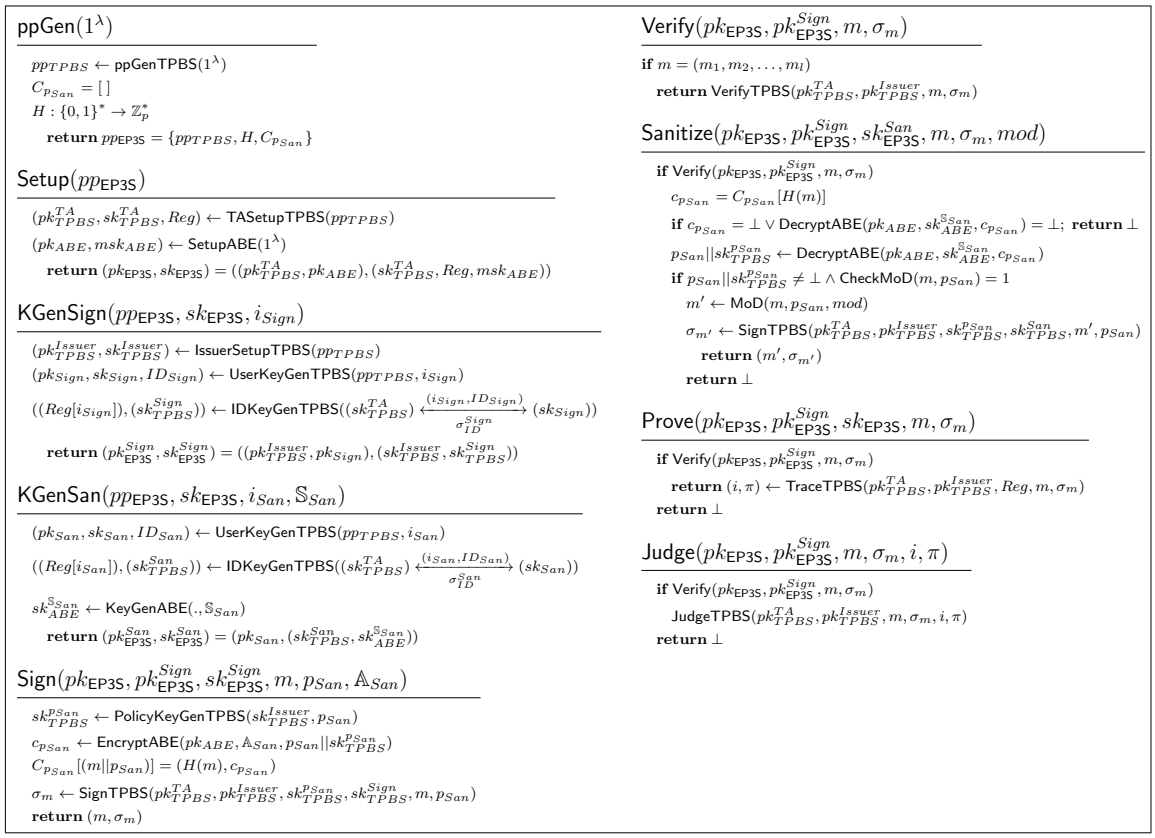

Figure 6.1: EP3S gneric construction.

Policy-checker for EP3S In what follows we show how a policy-checker PC could be constructed to the basic sanitization policy that allows an authorized sanitizer to sanitize a message m. We follow the original definition of the monotone policy p in [4, 12] which is defined as a set of Pairing Product Equations (PPEs) $(E_1, \ldots, E_n)$ [58], such that the policy checker $\mathsf{PC}((p, m), w_p) = 1$ iff $E_j((p, m), w_p) = 1$ for all $j \in [n]$. Let TPBS scheme be defined over a bilinear map $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$ that is generated by $(g, \tilde{g})$ and H is a collision-resistant hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$. Note that in what follows the group elements generated by the signer which defines the hidden policy p are marked by the superscript*. A modification policy p_{San} can be defined as follows:

$\mathsf{PC}(p_{San}, l||m) \Leftrightarrow$

$$e(g^{H(l)^*}, \tilde{g}) = e(g^{H(l)}, \tilde{g}) \wedge$$
$$e(g^{H(m_j)^*}, \tilde{g}) = e(g^{H(m'_j)}, \tilde{g}) \; for \; j \notin A$$

The above definition could be easily extended to any monotone modification policy required by the signer. Since the *LimitSet* restriction is deemed generally to be the most useful one [45], we give an example of how the signer can restrict future sanitizer(s) to modify a certain block of the message m_x following a predefined list of modifications (m'_{x_1}, m'_{x_2}), the above policy-checker PC could be adapted as follows:

$\mathsf{PC}(p, l||m) \Leftrightarrow$

$$e(g^{H(l)^*}, \tilde{g}) = e(g^{H(l)}, \tilde{g})$$
$$\wedge \, e(g^{H(m_j)^*}, \tilde{g}) = e(g^{H(m'_j)}, \tilde{g}) \; for \; j \notin A$$
$$\wedge \, \{e(g^{H(m'_{x_1})^*}, \tilde{g}) = e(g^{H(m'_x)}, \tilde{g})$$
$$\vee \, e(g^{H(m'_{x_2})^*}, \tilde{g}) = e(g^{H(m'_x)}, \tilde{g})\}$$

We refer the reader to [57, 89] on how to express the disjunction in the above example as the conjunction of sets of PPEs. Note that while our policy checker's definition, denoted as $\mathsf{PC}(p, l||m)$, typically involves concatenating the message number of blocks l to the message m in the format of $l||m$, in the generic construction Fig.(6.1), we use m to refer to the same format for the sake of simplicity.

Modification policy delegation A sanitizer who holds a secret key for some modification policy can delegate such a key to another sanitizer with possible restriction of the associated modification policy. For instance, the sanitizer can further restrict future sanitizers to modify a subset of the admissible blocks or add some restriction on the values to which

certain admissible blocks could be modified. The central concept is that sanitizers cannot override the modification policy established by the original message signer. However, they have the option to transfer their sanitization authority to others, either unrestricted or with additional conditions. To illustrate, if the initial signer granted sanitizer A the authority to alter a signed medical report concerning a specific patient by anonymizing personal details and/or amending the diagnosis section. Sanitizer A could delegate these rights to sanitizer B, potentially restricting B to modifying only the diagnosis section. Sanitizer B would then be held responsible solely for the sanitization attempts it undertakes.

This seamless process is facilitated by constructing the underlying TPBS within a delegatable framework, an approach initially introduced in [12] and further developed in [4] for a traceable context. The core concept of this approach allows the signer to use an append-only signature scheme [63] to generate the modification policy key $sk_{TPBS}^{p_1}$ associated with specific modification policy p_1. Any sanitizer who possesses $sk_{TPBS}^{p_1}$ can then compute $sk_{TPBS}^{p1||p_2}$, where $p1||p_2$ represents the fusion of the signer's original modification policy p_1 and the new restriction policy p_2 imposed by the sanitizer. With this newly derived modification policy key $sk_{TPBS}^{p_1||p_2}$ in hand, the subsequent sanitizer gains the ability to alter the same message. However, this ability is governed by the constraint that any modified message must adhere to both p_1 and p_2. In such a case, the new sanitizer assumes accountability for the signature it produces.

Multiple modifications policies for the same message An exciting feature of EP3S is that it can be easily adapted to support multiple modifications\sanitizers policies for the same message. More precisely, if a signer of a message m wants to specify n modifications policies p_{San_i} for $i \in \{0 \dots n\}$ where each policy p_{San_i} allows a different group of sanitizers identified by the access structure $\mathbb{A}_{San_i}$ to modify the message m in different ways.

The signing and sanitizing algorithms of EP3S can be adapted such that the signer generates a different TPBS modification policy key $sk_{TPBS}^{p_{San_i}}$ for each p_{San_i}, then encrypts each $p_{San_i}||sk_{TPBS}^{p_{San_i}}$ under $\mathbb{A}_{San_i}$ that defines the corresponding group of sanitizers, so that $c_{p_{San}}$ becomes an array of ABE encryptions $c_{p_{San_i}}$. A possible group of sanitizers who hold an ABE secret key that corresponds to a set of attributes $\mathbb{S}_{San_i}$ that satisfy some $\mathbb{A}_{San_i}$, can decrypt the corresponding ABE encryption in $c_{p_{San}}$ to obtain $p_{San_i}||sk_{TPBS}^{p_{San_i}}$ and then sanitizes the message in the same fashion of the original scheme.

6.3 EP3S Security Definitions

In what follows, we give the definitions of the required security notion of EP3S. We use the same notations as in [5, 26, 28, 89] for ease of readability. The oracles used in the security experiments are defined in Fig. 6.2.

Unforgeability This notion implies that an adversary with no access to either the signer secret key $sk_{\mathsf{EP3S}}^{Sign}$ or the scheme secret key sk_{EP3S} cannot generate a verifiable signature under honestly generated keys. This also includes cases where the adversary does not possess the required attribute set that satisfies the access structure specified by the signer. This must hold even if the adversary has access to additional message signature pairs and the public keys. Unforgeability is modeled by the experiment depicted in Fig. 6.3 in which adversary $\mathcal{A}$ has access to four oracles $\mathcal{O}$GetSanEP3S, $\mathcal{O}$SignEP3S, $\mathcal{O}$SanitEP3S and possesses a set of attributes $\mathbb{S}_{\mathcal{A}}$ tracked by list $\mathcal{S}$. The adversary gets signatures for messages (resp. sanitized messages) by querying the $\mathcal{O}$SignEP3S (resp. $\mathcal{O}$SanitEP3S) oracle tracked by $\mathcal{M}$ (resp. $\mathcal{L}$). $\mathcal{A}$ wins if it outputs a verifiable message signature pair (m^*, σ_m^*) that has never been queried to $\mathcal{O}$SignEP3S nor $\mathcal{O}$SanitEP3S oracles and the adversary does not hold an

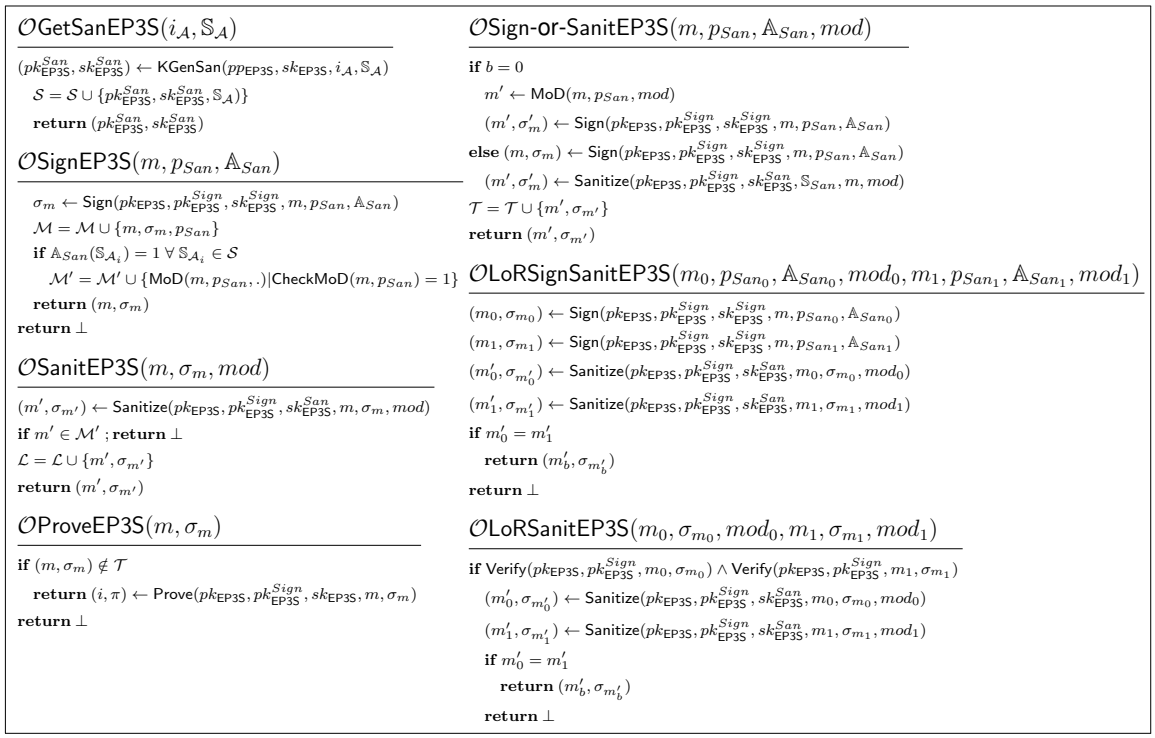

Figure 6.2: EP3S security experiments oracles.

attribute-set $\mathbb{S}_{\mathcal{A}}$ that satisfies any of possible sanitizers access structures specified in any of the adversary queries to $\mathcal{O}$SignEP3S. Note, We assume that both $sk^{Sign}_{\mathsf{EP3S}}$ and sk_{EP3S} are generated honestly. Furthermore, $\mathcal{M}'$ is used to track all the possible modifications over m that conform to the corresponding modification policy p_{San} if the adversary holds an attribute set that satisfies the supplied sanitizer access structure $\mathcal{A}_{San}$.

Definition 15. *(Unforgeability)* EP3S *scheme is unforgeability if for any PPT adversary* $\mathcal{A}$, $|\mathsf{Pr}[\boldsymbol{Exp}^{Unforgeability}_{\mathcal{A},\mathsf{EP3S}}(\lambda) = 1]| \leq \epsilon(\lambda)$, *where the unforgeability experiment is defined in Fig. 6.3.*

Immutability Originally, this security notion implies that no adversary with no access to the signer's secret key $sk^{Sign}_{\mathsf{EP3S}}$ can alter inadmissible blocks. However, in EP3S we extend such a security notion such that no adversary with no access to the signer's secret key $sk^{Sign}_{\mathsf{EP3S}}$ can modify the message m in any way that does not conform the modification policy p_{San}

Figure 6.3: EP3S security experiments.

set by the signer. Immutability is modeled by the security experiment defined in Fig. 6.3 in which adversary $\mathcal{A}$ has access to $\mathcal{O}$GetSanEP3S, and $\mathcal{O}$SignEP3S oracles. The signing oracle $\mathcal{O}$SignEP3S is initialized with $sk^{Sign}_{\mathsf{EP3S}}$. $\mathcal{A}$ can query $\mathcal{O}$SignEP3S by $m_i, p_{San_i}, \mathbb{A}_{San_i}$ for $i = 1, 2, ..., q$, the signing oracle outputs the message signature pairs (m_i, σ_{m_i}). The adversary wins if it could generate a verifiable (m^*, σ_{m^*}) such that for all $i = 1, 2, ..., q$,

m^* is not valid a modification of any m_i under any p_{San_i} where $\mathsf{CheckMoD}(m_i, p_{San}) = 1$. Note that the definition considers adversaries who are valid sanitizers trying to alter the message m in a way that was not specified by the modification policy set by the signer thus the adversary may access some sanitization key $sk^{\mathcal{A}}_{\mathsf{EP3S}}$ for a predefined attribute set $\mathbb{S}_{\mathcal{A}}$ using $\mathcal{O}\mathsf{GetSanEP3S}$ oracle.

Definition 16. *(Immutability)* EP3S *is an immutable sanitizable signature scheme if for any PPT adversary* $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{Immutability}_{\mathcal{A},\mathsf{EP3S}}(\lambda) = 1] \leq \epsilon(\lambda)$, *where the immutability experiment is defined in Fig. 6.3.*

Transparency This notion requires that no adversary can distinguish between sanitizable signatures created by the signer or the sanitizer. In other words, no adversary can figure out whether a sanitizable signature is a freshly signed signature or a sanitized one. Transparency is modeled by the experiment in Fig. 6.3 in which adversary $\mathcal{A}$ has access to $\mathcal{O}\mathsf{SignEP3S}$, $\mathcal{O}\mathsf{SanitEP3S}$, and $\mathcal{O}\mathsf{ProveEP3S}$. $\mathcal{A}$ gets signatures (resp. sanitized signatures) for messages of its choice by querying $\mathcal{O}\mathsf{SignEP3S}$ (resp. $\mathcal{O}\mathsf{SanitEP3S}$), and learns proofs by querying $\mathcal{O}\mathsf{ProveEP3S}$. At the end, $\mathcal{A}$ queries $\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitEP3S}$ with a message m, a modification mod, possible sanitizers access structure $\mathbb{A}_{San}$ and a modification policy p_{San}. $\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitEP3S}$ which is initialized by a secret random bit b, outputs the message signature pair (m', σ'_m) as follows.

- For $b = 0$, $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$, $\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitEP3S}$ runs the signing algorithm to create $\sigma'_m \leftarrow \mathsf{Sign}$ (pk_{EP3S}, $pk^{Sign}_{\mathsf{EP3S}}$, $sk^{Sign}_{\mathsf{EP3S}}$, m, p_{San}, $\mathbb{A}_{San}$) and outputs the message signature pair (m', σ'_m).
- For $b = 1$, $\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitEP3S}$ runs the signing algorithm to create $\sigma_m \leftarrow \mathsf{Sign}$ (pk_{EP3S}, $pk^{Sign}_{\mathsf{EP3S}}$, $sk^{Sign}_{\mathsf{EP3S}}$, m, p_{San}, $\mathbb{A}_{San}$), and returns $(m', \sigma'_m) \leftarrow \mathsf{Sanitize}$ (pk_{EP3S}, $pk^{Sign}_{\mathsf{EP3S}}$, sk^{San}_{EP3S}, m, σ_m, mod).

$\mathcal{A}$ wins if it can guess b with probability better than the random guess. Note that access to $\mathcal{O}$ProveEP3S oracle is restricted to (m, σ_m) pairs that have never been queried to $\mathcal{O}$Sign-or-SanitEP3S oracle, to prevent trivial attacks where the attacker submits the output of $\mathcal{O}$Sign-or-SanitEP3S oracle to $\mathcal{O}$ProveEP3S and determines if the message is signed or sanitized.

Definition 17. *(Transparency)* EP3S *is transparent if for any PPT adversary $\mathcal{A}$,*

$$\left| \Pr[\boldsymbol{Exp}^{Transparency}_{\mathcal{A},\mathsf{EP3S}}(\lambda) = 1] - \frac{1}{2} \right| \leq \epsilon(\lambda)$$

, where the transparency experiment is defined in Fig 6.3

Unlinkability Unlinkability is defined using the experiment in Fig 6.3, where the adversary has access to left-or-right sanitization oracle $\mathcal{O}$LoRSanitEP3S (see Fig. 6.2) among other oracles. The adversary inputs two sanitizable messages-signature pairs { (m_0, σ_{m0}), (m_1, σ_{m1}) } along with their modifications (mod_0, mod_1) to $\mathcal{O}$LoRSanitEP3S, the oracle is initialized with a secret random bit '$b \in \{0, 1\}$'. Depending on 'b', the oracle outputs a sanitized signature of either the left or right input message signature pair. The adversary wins if it can determine which pair is used in the sanitization process with probability better than the random guess. Unlike the definition in [5, 28], where the adversary is restricted to input two messages with identical fixed parts, $m_{0_{!adm}} = m_{1_{!adm}}$ and the two messages' admissible blocks indices must be the same. We only restrict the adversary to input two messages such that their modified outputs are the same $m'_0 = m'_1$ to prevent linking a sanitized message to its original source. Note that, unlike group signature schemes where unlinkability is defined as the infeasibility to link two messages and their signatures to the same signer [13], in sanitizable signature, unlinkability is defined as the infeasibility to link signatures of two

or more sanitized versions of a message to the same source message [28].

Definition 18. *(Unlinkability)* EP3S *scheme is unlinkable if for any PPT adversary $\mathcal{A}$,* $\left|\mathsf{Pr}[\boldsymbol{Exp}^{Unlinkability}_{\mathcal{A},\mathsf{EP3S}}(\lambda)=1]-\frac{1}{2}\right| \leq \epsilon(\lambda)$*, where the unlinkability experiment is described in Fig. 6.3.*

Accountability This security notion ensures that the original signer of a message whether signed or sanitized is undeniable. More precisely, accountability implies that if a signer did not sign a message (resp. sanitizer did not sanitize a message), then a malicious sanitizer (resp. signer) should not be able to convince the judge to accuse the signer (resp. sanitizer). Accountability is modeled by the security experiment defined in Fig. 6.3, in which adversary $\mathcal{A}$ has access to either sk^{San}_{EP3S} (resp. $sk^{Sign}_{\mathsf{EP3S}}$), in addition to two oracles $\mathcal{O}$SanitEP3S (resp. $\mathcal{O}$SignEP3S) and $\mathcal{O}$ProveEP3S. $\mathcal{A}$ can query $\mathcal{O}$SanitEP3S (resp. $\mathcal{O}$SignEP3S) with $(m_i, \sigma_{mi}, mod_i)$ (resp. $m_i, p_{San_i}, \mathbb{A}_{San_i}$) to get (m'_i, σ'_{mi}) (resp. (m_i, σ_{mi})) for $i = \{1, 2, ..., q\}$. The adversary wins if it outputs a verifiable message signature pair (m^*, σ_{m^*}) where $m^* \notin \{m_1, \ldots, m_q\}$ and the output $\mathcal{O}$ProveEP3S oracle on the input of (pk_{EP3S}, $pk^{Sign}_{\mathsf{EP3S}}$, m, σ_m) is falsely traced back to i_{Sign} if $\mathcal{A}$ has access to sk^{San}_{EP3S}, or to i_{San} if $\mathcal{A}$ has access to $sk^{Sign}_{\mathsf{EP3S}}$, and such a result is verified by the JudgeEP3S algorithm. Note that The adversary wins also if the tracing algorithm outputs $\perp$ in either case.

Definition 19. *(Accountability)* EP3S *ensures accountability if for any PPT adversary $\mathcal{A}$,* $\mathsf{Pr}[\boldsymbol{Exp}^{Accountability}_{\mathcal{A},\mathsf{EP3S}}(\lambda) = 1] \leq \epsilon(\lambda)$*, where the accountability experiment is defined in Fig. 6.3*

Privacy This notion implies that it is infeasible to use sanitized signatures to recover information about the sanitized parts of the message. Privacy is defined using an experiment where the adversary inputs two message-modifications tuples (m_0, p_{San_0}, $\mathbb{A}_{San_0}$, mod_0) and

$(m_1, p_{San_1}, \mathbb{A}_{San_1}, mod_1)$ to $\mathcal{O}$LoRSignSanitEP3S oracle which is initialized with a secret random bit 'b'. Depending on 'b', the oracle outputs a sanitized signature of either the left or right input message modification tuple. The adversary wins if it can determine which pair is used in the sanitization process with probability better than the random guess.

Definition 20. *(Privacy)* EP3S *scheme is private if for any PPT adversary $\mathcal{A}$,*

$$|\mathsf{Pr}[\boldsymbol{Exp}^{Privacy}_{\mathcal{A},\mathsf{EP3S}}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$$

where the privacy experiment is defined in Fig. 6.3.

6.4 EP3S Security Analysis

It has been proven in [28] that unlinkable sanitizable signature schemes are private. More precisely, Brzuska *et al.* have shown how to convert an adversary against privacy into an adversary against unlinkability. Accordingly, in what follows we prove that EP3S is unforgeable, immutable, transparent, unlinkable (implies private), and accountable sanitizable signature scheme.

Theorem 11. *Given an extractable, anonymous, policy-private, non-frameable, traceable TPBS scheme, and an IND-CCA-2 secure ABE scheme, the policy-based sanitizable signature scheme in Fig. 6.1 is unforgeable (Def. 15), immutable (Def. 16), transparent (Def. 17), unlinkable (Def 18)(implies private), and accountable (Def. 19) sanitizable signature scheme.*

Proof. In this proof, we leverage the security definitions and accompanying oracles from the TPBS scheme outlined in [4] and Chapter 5, specifically in Section 5.2. To enhance clarity, we append the postscript *TPBS* to each of the referenced oracles. We prove each

security property individually.

Unforgeability Recall that for adversary $\mathcal{A}$ to win the unforgeability game $\mathbf{Exp}^{Unforgeability}_{\mathcal{A},EP3S}$ in Fig. 6.3, it has to output a verifiable (m^*, σ_{m^*}) under the signer public key where i) m^* has never been queried to the $\mathcal{O}\mathsf{SignEP3S}$ oracle, ii) (m^*, σ_{m^*}) has never been queried to the $\mathcal{O}\mathsf{SanitEP3S}$ oracle, and iii) m^* is not a valid modification for any message queried to $\mathcal{O}\mathsf{SignEP3S}$ whenever the access structure that defines the possible sanitizers is satisfied by any of the attributes sets $\mathcal{A}$ possesses. Since the adversary has no access to the signer secret key sk^{Sign}_{EP3S} or the scheme secret key sk_{EP3S}, a verifiable (m^*, σ_{m^*}) generation requires access to some sk^{San}_{TPBS} and $sk^{p_{San}}_{TPBS}$. Furthermore, since the adversary has access to $\mathcal{O}\mathsf{GetSanEP3S}$ oracle where it can obtain a valid sk^{San}_{TPBS}, here we distinguish between two types of adversaries, i) $\mathcal{A}$ is of type-1 if it can generate a verifiable (m^*, σ_{m^*}) without holding the corresponding modification policy secret key $sk^{p_{San}}_{TPBS}$, ii) $\mathcal{A}$ is of type-2 if it can generate a verifiable (m^*, σ_{m^*}) where it does not possess some attribute sets that satisfy the sanitizers access structure that protects $sk^{p_{San}}_{TPBS}$ even if it holds keys for other attribute sets. By contradiction, we show that if there exists an adversary $\mathcal{A}$ of type-1 or type-2 that wins $\mathbf{Exp}^{Unforgeability}_{\mathcal{A},EP3S}$, we can construct an adversary $\mathcal{B}$ that wins the extractability game $\mathbf{Exp}^{Ext}_{\mathcal{B},TPBS}$ of the underlying TPBS scheme, or an adversary $\mathcal{B}'$ that wins the IND-CCA-2 game $\mathbf{Exp}^{CCA-2}_{\mathcal{B}',ABE}$ of the underlying ABE scheme in Fig. A.15, respectively. For a type-1 $\mathcal{A}$ adversary, $\mathcal{B}$ is constructed as follows. $\mathcal{B}$ receives the tuple $(pp_{TPBS}, pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS})$ from its challenger in extractability game $\mathbf{Exp}^{Ext}_{\mathcal{B},TPBS}$, generates $(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^\lambda)$ honestly, creates an empty key-value pair registry $C_{p_{San}} = [\ ]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$. Then $\mathcal{B}$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$, $pk_{EP3S} = (pk^{TA}_{TPBS}, pk_{ABE})$, and initializes the empty sets $\mathcal{M} = \mathcal{M}' = \mathcal{L} = \mathcal{S} = \{\}$. Finally, $\mathcal{B}$ sends some i_{Sign} (resp. $i_{San'}$) along with any dummy

policy p to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{Sign}, sk^{Sign}_{TPBS}, sk^{p}_{TPBS})$ (resp. $(pk_{San}, sk^{San'}_{TPBS}, sk^{p}_{TPBS})$). To simulate $\mathcal{A}$ calls to $\mathcal{O}$GetSanEP3S oracle upon inputting $(i_{\mathcal{A}}, \mathbb{S}_{\mathcal{A}})$, $\mathcal{B}$ sends $i_{\mathcal{A}}$ along with any dummy policy p to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{San}, sk^{San}_{TPBS}, sk^{p}_{TPBS})$, and uses its knowledge of msk_{ABE} to generate $sk^{\mathbb{S}_{San}}_{ABE} \leftarrow$ KeyGenABE$(msk_{ABE}, \mathbb{S}_{\mathcal{A}})$. Finally, $\mathcal{B}$ returns $(pk^{San}_{EP3S}, sk^{San}_{EP3S}) = (pk_{San}, (sk^{San}_{TPBS}, sk^{\mathbb{S}_{San}}_{ABE}))$ to $\mathcal{A}$, and sets $\mathcal{S} = \mathcal{S} \cup \{pk^{San}_{EP3S}, sk^{San}_{EP3S}, \mathbb{S}_{\mathcal{A}})$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$SignEP3S oracle upon inputting $(m, p_{San}, \mathbb{A}_{San})$, If $\mathbb{A}_{San}(\mathbb{S}_{\mathcal{A}_i}) = 1 \ \forall \ \mathbb{S}_{\mathcal{A}_i} \in \mathcal{S}$, $\mathcal{B}$ sends p_{San} along with a dummy identity i to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{Sign}, sk^{i}_{TPBS}, sk^{p_{San}}_{TPBS})$, generates $c_{p_{San}} \leftarrow$ EncryptABE$(pk_{ABE}, \mathbb{A}_{San}, p_{San}||sk^{p_{San}}_{TPBS})$, sets $C_{p_{San}}[H(m)||p_{San}] = c_{p_{San}}$, generates $\sigma_m \leftarrow$ SignTPBS $(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^{p_{San}}_{TPBS}, sk^{Sign}_{TPBS}, m, p_{San})$, sets $\mathcal{M} = \mathcal{M} \cup \{ m, \sigma_m, p_{San} \}$ and $\mathcal{M}' = \mathcal{M}' \cup \{$ MoD $(m, p_{San}, .)$ | CheckMoD $(m, p_{San}) = 1\}$; finally, $\mathcal{B}$ returns (m, σ_m) to $\mathcal{A}$. Else, $\mathcal{B}$ sends (m, i_{Sign}, p_{San}) to its challenger $\mathcal{O}$SignTPBS oracle to receive σ_m, sets $C_{p_{San}}[H(m)||p_{San}]$ to a dummy value, sets $\mathcal{M} = \mathcal{M} \cup \{m, \sigma_m, p_{San}\}$, keeps track of (m, σ_m, p_{San}) in a list $\mathcal{P}$; finally, $\mathcal{B}$ returns (m, σ_m) to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$SanitEP3S oracle upon inputting (m, σ_m, mod), $\mathcal{B}$ obtains the corresponding p_{San} from $\mathcal{P}$ or returns $\perp$ if doesn't exist, generates $m' \leftarrow$ MoD(m, p_{San}, mod), sends (m', i_{San}, p_{San}) to its challenger $\mathcal{O}$SignTPBS oracle to receive $\sigma_{m'}$, sets $\mathcal{L} = \mathcal{L} \cup \{m', \sigma_{m'}\}$ and returns $(m', \sigma_{m'})$ to $\mathcal{A}$. Once $\mathcal{A}$ outputs (m^*, σ_{m^*}), $\mathcal{B}$ forwards them to its challenger in $\mathbf{Exp}^{Ext}_{\mathcal{B},TPBS}$, which constitutes a winning condition where the adversary has no access to the modification policy secret key.

For a type-2 $\mathcal{A}$ adversary, $\mathcal{B}'$ is constructed as follows. $\mathcal{B}'$ receives pk_{ABE} from its challenger in extractability game $\mathbf{Exp}^{CCA-2}_{\mathcal{B},ABE}$, $\mathcal{B}'$ sets the access structure $\mathbb{A}^*$, generates $pp_{TPBS} \leftarrow$ ppGenTPBS(1^{λ}), creates an empty key-value pair registry $C_{p_{San}} = [\ ]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$. Then $\mathcal{B}'$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$. Then, $\mathcal{B}'$ gener-

ates $(pk^{TA}_{TPBS}, sk^{TA}_{TPBS}, Reg) \leftarrow \mathsf{TASetupTPBS}(pp_{TPBS})$, sets $pk_{EP3S} = (pk^{TA}_{TPBS}, pk_{ABE})$, and initializes the empty sets $\mathcal{M} = \mathcal{M}' = \mathcal{L} = \mathcal{S} = \{\}$. $\mathcal{B}'$ generates $(pk^{Issuer}_{TPBS}, sk^{Issuer}_{TPBS}) \leftarrow \mathsf{IssuerSetupTPBS}(pp_{TPBS})$, $(pk_{Sign}, sk_{Sign}, ID_{Sign}) \leftarrow \mathsf{UserKeyGenTPBS}(pp_{TPBS}, i_{Sign})$, and $((Reg[i_{Sign}]), (sk^{Sign}_{TPBS})) \leftarrow \mathsf{IDKeyGenTPBS}((sk^{TA}_{TPBS}) \xleftarrow[\sigma^{Sign}_{ID}]{(i_{Sign}, ID_{Sign})} (sk_{Sign}))$ for some i_{Sign}. $\mathcal{B}'$ generates $(pk_{San}, sk_{San}, ID_{San}) \leftarrow \mathsf{UserKeyGenTPBS}(pp_{TPBS}, i_{San})$, and $((Reg[i_{San}]), (sk^{San}_{TPBS})) \leftarrow \mathsf{IDKeyGenTPBS}((sk^{TA}_{TPBS}) \xleftarrow[\sigma^{San}_{ID}]{(i_{San}, ID_{San})} (sk_{San}))$ for some i_{San}. To simulate $\mathcal{A}$ calls to $\mathcal{O}$GetSanEP3S oracle upon inputting $(i_{\mathcal{A}}, \mathbb{S}_{\mathcal{A}})$, if $\mathbb{A}^*(\mathbb{S}_{\mathcal{A}}) = 1$ returns $\perp$, else $\mathcal{B}'$ sends $\mathbb{S}_{\mathcal{A}}$ to its challenger $\mathcal{O}$KeyGenABE oracle to receive $sk^{\mathbb{S}_{\mathcal{A}}}_{ABE}$ and generates $(pk_{San}, sk_{San}, ID_{\mathcal{A}}) \leftarrow \mathsf{UserKeyGenTPBS}(pp_{TPBS}, i_{\mathcal{A}})$, and $((Reg\ [i_{\mathcal{A}}]), (sk^{San}_{TPBS}))$ $\leftarrow$ IDKeyGenTPBS $((sk^{TA}_{TPBS}) \leftrightarrow [\sigma^{\mathcal{A}}_{ID}]\ (i_{\mathcal{A}},\ ID_{\mathcal{A}})\ (sk_{San}))$. Finally $\mathcal{B}'$ return $(pk^{San}_{EP3S}$, $sk^{San}_{EP3S}) = (pk_{San}$, $(sk^{San}_{TPBS}$, $sk^{\mathbb{S}_{\mathcal{A}}}_{ABE}))$ to $\mathcal{A}$, and sets $\mathcal{S}$ = $\mathcal{S} \cup \{\ pk^{San}_{EP3S}$, sk^{San}_{EP3S}, $\mathbb{S}_{\mathcal{A}})$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$SignEP3S oracle upon inputting $(m$, p_{San}, $\mathbb{A}_{San})$, generates $sk^{p_{San}}_{TPBS}$ $\leftarrow$ PolicyKeyGenTPBS $(sk^{Sign}_{TPBS}, p_{San})$, If $\mathbb{A}_{San}(\mathbb{S}_{\mathcal{A}_i}) = 1\ \forall\ \mathbb{S}_{\mathcal{A}_i} \in \mathcal{S}$, $\mathcal{B}'$ generates $c_{p_{San}} \leftarrow$ $\mathsf{EncryptABE}(pk_{ABE}, \mathbb{A}_{San}, p_{San}||sk^{p_{San}}_{TPBS})$, sets $C_{p_{San}}[H(m)||p_{San}] = c_{p_{San}}$, generates σ_m $\leftarrow$ SignTPBS $(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^{p_{San}}_{TPBS}, sk^{Sign}_{TPBS}, m, p_{San})$, sets $\mathcal{M} = \mathcal{M} \cup \{m, \sigma_m, p_{San}\}$ and $\mathcal{M}' = \mathcal{M}' \cup \{\mathsf{MoD}(m, p_{San}, .) | \mathsf{CheckMoD}(m, p_{San}) = 1\}$; finally, $\mathcal{B}'$ returns (m, σ_m) to $\mathcal{A}$. Else, $\mathcal{B}'$ generates a random bit string R such that $|R| = |p_{San}||sk^{p_{San}}_{TPBS}|$, sends R and $p_{San}||sk^{p_{San}}_{TPBS}$ to its challenger oracle $\mathcal{O}\mathsf{LoREncryptABE}(m_0, m_1)$ to receive C^*_b, sets $C_{p_{San}}[H(m)||p_{San}] = C^*_b$, generates $\sigma_m \leftarrow$ SignTPBS $(pk^{TA}_{TPBS}$, pk^{Issuer}_{TPBS}, $sk^{p_{San}}_{TPBS}$, sk^{Sign}_{TPBS}, m, $p_{San})$, sets $\mathcal{M} = \mathcal{M} \cup \{m, \sigma_m, p_{San}\}$ and $\mathcal{M}' = \mathcal{M}' \cup \{$MoD $(m$, p_{San}, .) | CheckMoD $(m$, $p_{San}) = 1\}$; finally, $\mathcal{B}'$ returns (m, σ_m) to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$SanitEP3S oracle upon inputting $((m, \sigma_m, mod))$, $\mathcal{B}'$ obtains $c_{p_{San}} = C_{p_{San}}[H(m)]$, sends $c_{p_{San}}$ to its challenger $\mathcal{O}$DecryptABE oracle to obtain $p_{San}||sk^{p_{San}}_{TPBS}$, generates $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$, generates $\sigma_{m'} \leftarrow \mathsf{SignTPBS}(pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS}, sk^{p_{San}}_{TPBS}, sk^{San}_{TPBS}, m', p_{San})$, sets $\mathcal{L} = \mathcal{L} \cup$

$\{m', \sigma_{m'}\}$ and returns $(m', \sigma_{m'})$ to $\mathcal{A}$. Once $\mathcal{A}$ outputs (m^*, σ_{m^*}), if Verify $(pk_{EP3S}, pk^{Sign}_{EP3S}$, m^*,$\sigma_{m^*}) = \top$ $\mathcal{B}'$ returns $b' = 1$ to its challenger in $\mathbf{Exp}^{CCA-2}_{\mathcal{B},ABE}$ or $b' = 0$ elsewhere.

Immutability Recall that for adversary $\mathcal{A}$ to win the immutability game $\mathbf{Exp}^{Immutability}_{\mathcal{A},EP3S}$ in Fig. 6.3, it has to output a verifiable (m^*, σ_{m^*}) under the signer public key where m^* has never been queried to the $\mathcal{O}$SignEP3S oracle and m^* is not a valid modification of any message queried to the $\mathcal{O}$SignEP3S oracle under the message's corresponding policy. For $\mathcal{A}$ to succeed, it either created the message signature pair (m^*, σ_{m^*}) without holding the corresponding keys or it has modified some m_i in a way that doesn't conform with its corresponding policy p_{San_i} i.e. $\mathsf{PC}(p_{San_i}, m) \neq 1$ for all $i = 1, 2, ..., q$ queried to the $\mathcal{O}$SignEP3S sign oracle. Since EP3S unforgeability handles the first case, we show that by contradiction if there exists an adversary $\mathcal{A}$ that wins $\mathbf{Exp}^{Immutability}_{\mathcal{A},EP3S}$, we can construct an adversary $\mathcal{B}$ that wins the extractability game $\mathbf{Exp}^{Ext}_{\mathcal{B},TPBS}$ of the underlying TPBS scheme. $\mathcal{B}$ is constructed as follows. $\mathcal{B}$ receives the tuple $(pp_{TPBS}, pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS})$ from its challenger in extractability game $\mathbf{Exp}^{Ext}_{\mathcal{B},TPBS}$, generates $(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^\lambda)$ honestly, creates an empty key-value pair registry $C_{p_{San}} = [\,]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$. Then $\mathcal{B}$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$, $pk_{EP3S} = (pk^{TA}_{TPBS}, pk_{ABE})$, and initializes the empty set $\mathcal{M} = \{\}$. Finally, $\mathcal{B}$ sends some i_{Sign} along with any dummy policy p to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{Sign}, sk^{Sign}_{TPBS}, sk^{p}_{TPBS})$, sets $pk^{Sign}_{EP3S} = (pk^{Issuer}_{TPBS}, pk_{Sign})$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$GetSanEP3S oracle upon inputting $(i_{\mathcal{A}}, \mathbb{S}_{\mathcal{A}})$, $\mathcal{B}$ sends $i_{\mathcal{A}}$ along with any dummy policy p to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{San}, sk^{San}_{TPBS}, sk^{p}_{TPBS})$, and uses its knowledge of msk_{ABE} to generate $sk^{\mathbb{S}_{San}}_{ABE} \leftarrow \mathsf{KeyGenABE}(msk_{ABE}, \mathbb{S}_{\mathcal{A}})$. Finally, $\mathcal{B}$ returns $(pk^{San}_{EP3S}$, $\mathrm{sk}^{San}_{EP3S}) = (pk_{San}$, $(sk^{San}_{TPBS}$, $sk^{\mathbb{S}_{San}}_{ABE}))$ to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$SignEP3S oracle upon inputting $(m$, p_{San_i}, $\mathbb{A}_{San})$, $\mathcal{B}$ sends (m, i_{Sign}, p_{San}) to its challenger $\mathcal{O}$SignTPBS oracle to receive σ_m,

sets $C_{p_{San}}[H(m)||p_{San}]$ to a dummy value, sets $\mathcal{M} = \mathcal{M} \cup \{m, \sigma_m, p_{San}\}$, and returns (m, σ_m) to $\mathcal{A}$. Once $\mathcal{A}$ outputs (m^*, σ_{m^*}), $\mathcal{B}$ forwards them to its challenger in $\mathbf{Exp}^{Ext}_{\mathcal{B},TPBS}$, which constitutes a winning condition such that $\mathsf{PC}(p^*_{San}, m^*) \neq 1.$.

Transparency The privacy of the TPBS scheme ensures that the generated signature reveals no information about the identity of the real signer of the message nor the policy used in a signature generation other than the fact that the message conforms to such a policy. Hence, such a signature hides both the original signer's identity and the policy used in signature generation. Since both signer and sanitizer use the same policy p_{San} in generating a freshly signed signature and sanitized signature. Therefore, by contradiction, we assume that the EP3S scheme is not a transparent sanitizable signature scheme. We then show that the anonymity of the underlying TPBS scheme cannot hold. Let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}^{Transparency}_{\mathcal{A},EP3S}$, we show how to build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the anonymity of the underlying TPBS scheme and win in $\mathbf{Exp}^{Anonymity}_{\mathcal{B},TPBS}$. $\mathcal{B}$ is constructed as follows; $\mathcal{B}$ receives the tuple $(pp_{TPBS}, pk^{TA}_{TPBS}, pk^{Issuer}_{TPBS})$ and the honest user list $\mathcal{U}$ from its challenger in anonymity game $\mathbf{Exp}^{Anonymity}_{\mathcal{B},TPBS}$, generates $(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^\lambda)$ honestly, creates an empty key-value pair registry $C_{p_{San}} = [\,]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$. Then $\mathcal{B}$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$, $pk_{EP3S} = (pk^{TA}_{TPBS}, pk_{ABE})$, chooses i_{Sign} and i_{San} from $\mathcal{U}$ and receive their corresponding public keys (pk_{Sign}, pk_{San}), and sets $pk^{Sign}_{EP3S} = (pk^{Issuer}_{TPBS}, pk_{Sign})$. To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{SignEP3S}$ oracle upon inputting $(m, p_{San}, \mathbb{A}_{San})$, $\mathcal{B}$ sends (m, i_{Sign}, p_{San}) to its challenger $\mathcal{O}\mathsf{USignTPBS}$ oracle to receive σ_m and $sk^{p_{San}}_{TPBS}$, generates $c_{p_{San}} \leftarrow \mathsf{EncryptABE}\ (pk_{ABE}$, $\mathbb{A}_{San}$, $p_{San}\ ||\ sk^{p_{San}}_{TPBS})$, sets $C_{p_{San}}[H(m)||p_{San}] = c_{p_{San}}$, and returns (m, σ_m) to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{SanitEP3S}$ oracle upon inputting $((m, \sigma_m, mod))$, $\mathcal{B}$ obtains $c_{p_{San}} = C_{p_{San}}[H(m)]$, decrypts $c_{p_{San}}$ using its knowledge of msk_{ABE} to obtain $p_{San}||sk^{p_{San}}_{TPBS}$, gen-

erates $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$, sends (m', i_{San}, p_{San}) to its challenger $\mathcal{O}\mathsf{USignTPBS}$ oracle to receive σ'_m, and returns $(m', \sigma_{m'})$ to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{ProveEP3S}$ oracle upon inputting (m, σ_m), $\mathcal{B}$ forwards (m, σ_m) to its challenger $\mathcal{O}\mathsf{TraceTPBS}$ to receive (i, π). To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{Sign\text{-}or\text{-}SanitEP3S}$ oracle upon inputting $(m, p_{San}, \mathbb{A}_{San}, mod)$, $\mathcal{B}$ generates $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$, and and passes the message $(m', i_{Sign}, i_{San}, p_{San})$ to its challenger $\mathcal{O}\mathsf{IdLoRSignTPBS}$ oracle which responds with a challenge TPBS signature $\sigma_{m'}$ as either the signer or sanitizer signature over m'. At the end, $\mathcal{A}$ outputs a bit 'a' which $\mathcal{B}$ forwards as its answer to the $\mathcal{O}\mathsf{IdLoRSignTPBS}$ oracle.

Unlinkability The privacy guarantees provided by the TPBS scheme ensure the hiding of the signer of the message identity and the policy used in the signature generation other than the fact that the message conforms to such a policy. Since the challenging oracle $\mathcal{O}\mathsf{LoRSanitEP3S}$ in the unlinkability experiment in Fig. 6.3 uses the same sanitizer identity in generating the signature over m_0 and m_1, Therefore, by contradiction, we assume that the EP3S scheme is not an unlinkable sanitizable signature scheme. We then show that the policy-privacy of the underlying TPBS scheme cannot hold. Let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}^{Unlinkability}_{\mathcal{A},EP3S}$, we show how to build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the policy-privacy of the underlying TPBS scheme and win in $\mathbf{Exp}^{Policy-privacy}_{\mathcal{B},TPBS}$. $\mathcal{B}$ is constructed as follows; $\mathcal{B}$ receives the tuple $(pp_{EP3S}, pk^{TA}_{EP3S}, pk^{Issuer}_{EP3S}, sk^{TA}_{EP3S}, sk^{Issuer}_{EP3S})$ from its challenger in policy-privacy game $\mathbf{Exp}^{Policy-privacy}_{\mathcal{B},TPBS}$, generates $(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^\lambda)$ honestly, creates an empty key-value pair registry $C_{p_{San}} = [\,]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$. Then $\mathcal{B}$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$, $pk_{EP3S} = (pk^{TA}_{TPBS}, pk_{ABE})$, chooses i_{Sign} and i_{San} and generates their corresponding keys $(pk^{Sign}_{EP3S}, sk^{Sign}_{EP3S})$ and $(pk^{San}_{EP3S}, sk^{San}_{EP3S})$ using its knowledge of sk^{TA}_{EP3S} and msk_{ABE}, sets $pk^{Sign}_{EP3S} = (pk^{Issuer}_{TPBS}, pk_{Sign})$. To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{SignEP3S}$ oracle upon inputting $(m, p_{San}, \mathbb{A}_{San})$,

$\mathcal{B}$ generates $sk_{TPBS}^{p_{San}} \leftarrow$ PolicyKeyGenTPBS$(sk_{TPBS}^{Issuer}, p_{San})$ and $c_{p_{San}} \leftarrow$ EncryptABE (pk_{ABE}, $\mathbb{A}_{San}$, p_{San} || $sk_{TPBS}^{p_{San}}$), sets $C_{p_{San}}[H(m)||p_{San}] = c_{p_{San}}$, generates $\sigma_m \leftarrow$ SignTPBS (pk_{TPBS}^{TA}, pk_{TPBS}^{Issuer}, $sk_{TPBS}^{p_{San}}$, sk_{TPBS}^{Sign}, m, p_{San}) and finally returns (m, σ_m). To simulate $\mathcal{A}$ calls to $\mathcal{O}$SanitEP3S oracle upon inputting $((m, \sigma_m, mod))$, $\mathcal{B}$ obtains $c_{p_{San}} = C_{p_{San}}[H(m)]$, decrypts $c_{p_{San}}$ using its knowledge of msk_{ABE} to obtain $p_{San}||sk_{TPBS}^{p_{San}}$, generates $m' \leftarrow$ MoD(m, p_{San}, mod), generates $\sigma_{m'} \leftarrow$ SignTPBS (pk_{TPBS}^{TA}, pk_{TPBS}^{Issuer},$sk_{TPBS}^{p_{San}}$, sk_{TPBS}^{San}, m, p_{San}) and finally returns $(m', \sigma_{m'})$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$LoRSanitEP3S oracle upon inputting $(m_0, \sigma_{m_0}, mod_0, m_1, \sigma_{m_1}, mod_1)$, $\mathcal{B}$ decrypts $c_{p_{San_0}} = C_{p_{San}}[H(m)]$ and $c_{p_{San_1}} = C_{p_{San}}[H(m)]$ using its knowledge of msk_{ABE} to obtain $p_{San_0}||sk_{TPBS}^{p_{San_0}}$ and $p_{San_1}||sk_{TPBS}^{p_{San_1}}$, generates $m'_0 \leftarrow$ MoD(m, p_{San_0}, mod_0) and $m'_1 \leftarrow$ MoD(m, p_{San_1}, mod_1), if $m'_0 \neq m'_1$ returns $\perp$, else it passes the message $(m', i_{San}, p_{San_0}, p_{San_1})$ to its challenger $\mathcal{O}$PLoRSignTPBS oracle which responds with a challenge TPBS signature $\sigma_{m'_b}$ as signature over m' using p_{San_0} or p_{San_1}. At the end, $\mathcal{A}$ outputs a bit 'a' which $\mathcal{B}$ forwards as its answer to the $\mathcal{O}$PLoRSignTPBS oracle.

Accountability Recall that the non-frameability security property of a TPBS scheme ensures that even if the tracing authority, issuer, and all corrupt users in the scheme collude together, they cannot produce a valid signature that is traced back to an honest user. In other words, any generated signature must be traced back to the entity that holds the secret key used in signing such a message. Moreover, the traceability security property of a TPBS scheme ensures that every message signature pair generated can be traced. By contradiction, we let an adversary $\mathcal{A}$ be successful in $\mathbf{Exp}_{\mathcal{A},EP3S}^{Accountability}$ and show that we can build an adversary $\mathcal{B}$ (resp. $\mathcal{B}'$) which can break the non-frameability (resp. traceability) of the underlying TPBS scheme and win in $\mathbf{Exp}_{\mathcal{B},TPBS}^{Non-frameability}$ (resp. $\mathbf{Exp}_{\mathcal{B}',TPBS}^{Traceability}$. $\mathcal{B}$ simulates $\mathcal{A}$'s oracles as follows. $\mathcal{B}$ receives the tuple $(\mathcal{U}, pp_{EP3S}, pk_{EP3S}^{TA}, pk_{EP3S}^{Issuer}, sk_{EP3S}^{Issuer}, sk_{EP3S}^{TA})$

from its challenger in the non-frameability experiment, generates $(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^\lambda)$ honestly, creates an empty key-value pair registry $C_{p_{San}} = [\,]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Then $\mathcal{B}$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$, $pk_{EP3S} = (pk_{TPBS}^{TA}, pk_{ABE})$. To simulate the experiment when $\mathcal{A}$ has access to sk_{EP3S}^{Sign} (resp. sk_{EP3S}^{San}), $\mathcal{B}$ chooses $i_{Sign} \notin \mathcal{U}$ (resp. $i_{San} \notin \mathcal{U}$) and generates its corresponding keys $(pk_{EP3S}^{Sign}, sk_{EP3S}^{Sign})$ (resp. $(pk_{EP3S}^{San}, sk_{EP3S}^{San})$) using its knowledge of sk_{EP3S}^{TA}, sk_{EP3S}^{Issuer}, and msk_{ABE}, $\mathcal{B}$ chooses chooses $i_{Sign'}$ and $i_{San'}$ from $\mathcal{U}$ and receive their corresponding public keys $(pk_{Sign'}, pk_{San'})$, and finally sets $pk_{EP3S}^{Sign} = (pk_{TPBS}^{Issuer}, pk_{Sign})$ (resp. $pk_{EP3S}^{Sign} = (pk_{TPBS}^{Issuer}, pk_{Sign'})$). To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{SignEP3S}$ oracle upon inputting $(m, p_{San}, \mathbb{A}_{San})$, $\mathcal{B}$ sends $(m, i_{Sign'}, p_{San})$ to its challenger $\mathcal{O}\mathsf{USignTPBS}$ oracle to receive σ_m and $sk_{TPBS}^{p_{San}}$, generates $c_{p_{San}} \leftarrow \mathsf{EncryptABE}\,(\,pk_{ABE}, \mathbb{A}_{San}, p_{San} \,||\, sk_{TPBS}^{p_{San}})$, sets $C_{p_{San}}[H(m)||p_{San}] = c_{p_{San}}$, and returns (m, σ_m) to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{SanitEP3S}$ oracle upon inputting $((m, \sigma_m, mod))$, $\mathcal{B}$ obtains $c_{p_{San}} = C_{p_{San}}[H(m)]$, decrypts $c_{p_{San}}$ using its knowledge of msk_{ABE} to obtain $p_{San}||sk_{TPBS}^{p_{San}}$, generates $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$, sends $(m', i_{San'}, p_{San})$ to its challenger $\mathcal{O}\mathsf{USignTPBS}$ oracle to receive σ'_m, and returns $(m', \sigma_{m'})$ to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}\mathsf{ProveEP3S}$ oracle upon inputting (m, σ_m), $\mathcal{B}$ forwards (m, σ_m) to its challenger $\mathcal{O}\mathsf{TraceTPBS}$ to receive (i, π). At the end $\mathcal{A}$ outputs the message signature pair (m^*, σ_{m^*}) which $\mathcal{B}$ copies and forward to its non-frameability challenger in $\mathbf{Exp}_{\mathcal{B},TPBS}^{Non-frameability}$ experiment.

On the other hand, $\mathcal{B}'$ could be constructed as follows, $\mathcal{B}'$ receives the tuple $(pp_{EP3S}$, pk_{EP3S}^{TA}, $pk_{EP3S}^{Issuer})$ from its challenger in the Traceability experiment, generates $(pk_{ABE}$, $msk_{ABE}) \leftarrow \mathsf{SetupABE}\ (1^\lambda)$ honestly, creates an empty key-value pair registry $C_{p_{San}} = [\,]$, and chooses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Then $\mathcal{B}'$ sets $pp_{EP3S} \leftarrow (pp_{TPBS}, H, C_{p_{San}})$, $pk_{EP3S} = (pk_{TPBS}^{TA}, pk_{ABE})$. To simulate the experiment when $\mathcal{A}$ has access to sk_{EP3S}^{Sign}

(resp. sk_{EP3S}^{San}), $\mathcal{B}'$ chooses i_{Sign} (resp. i_{San}) and sends it along with a dummy policy p to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{Sign}, sk_{TPBS}^{Sign}, sk_{TPBS}^{p})$ (resp. (pk_{San}, sk_{TPBS}^{San}, sk_{TPBS}^{p})).

To simulate $\mathcal{A}$ calls to $\mathcal{O}$SignEP3S oracle upon inputting $(m, p_{San}, \mathbb{A}_{San})$, $\mathcal{B}'$ sends p_{San} along with a dummy identity i to its challenger $\mathcal{O}$KeyGenTPBS oracle to receive $(pk_{Sign}, sk_{TPBS}^{i}, sk_{TPBS}^{p_{San}})$, generates $c_{p_{San}} \leftarrow \mathsf{EncryptABE}(pk_{ABE}, \mathbb{A}_{San}, p_{San}||sk_{TPBS}^{p_{San}})$, sets $C_{p_{San}}[H(m)||p_{San}] = c_{p_{San}}$, generates $\sigma_m \leftarrow \mathsf{SignTPBS}$ (pk_{TPBS}^{TA}, pk_{TPBS}^{Issuer}, $sk_{TPBS}^{p_{San}}$, sk_{TPBS}^{Sign}, m, p_{San}); finally, $\mathcal{B}'$ returns (m, σ_m) to $\mathcal{A}$. To simulate $\mathcal{A}$ calls to $\mathcal{O}$SanitEP3S oracle upon inputting (m, σ_m, mod), $\mathcal{B}'$ obtains decrypts the corresponding $c_{p_{San}}$ using its knowledge of msk_{ABE} to obtain $p_{San}||sk_{TPBS}^{p_{San}}$, generates $m' \leftarrow \mathsf{MoD}(m, p_{San}, mod)$, generates $\sigma_{m'} \leftarrow \mathsf{SignTPBS}(pk_{TPBS}^{TA}, pk_{TPBS}^{Issuer}, sk_{TPBS}^{p_{San}}, sk_{TPBS}^{San}, m', p_{San})$; finally, $\mathcal{B}'$ returns $(m', \sigma_{m'})$ to $\mathcal{A}$. When $\mathcal{A}$ outputs a tuple (m^*, σ_{m^*}), $\mathcal{B}'$ forwards to its traceability challenger in $\mathbf{Exp}_{\mathcal{B}',TPBS}^{Traceability}$. Therefore, the success of $\mathcal{A}$ in $\mathbf{Exp}_{\mathcal{A},EP3S}^{Accountability}$ implies the success of $\mathcal{B}$ and $\mathcal{B}'$ in $\mathbf{Exp}_{\mathcal{B},TPBS}^{Non-frameability}$ and $\mathbf{Exp}_{\mathcal{B}',TPBS}^{Traceability}$, respectively. □

6.5 Instantiation and Efficiency

We instantiate EP3S with the TPBS Scheme in section 5 because it supports the delegation of signing keys in addition to providing all the security properties required by EP3S. For the CP-ABE scheme, we utilize the FAME scheme in [7] because it has no restrictions on the utilized monotone policies, its efficiency (constant decryption time) and it is adaptive CPA secure under standard assumptions. However, since FAME only provides IND-CPA security and EP3S requires IND-CCA2, we follow the same procedure in [44] to achieve IND-CCA2 which is based on the well-known transform of Fujisaki-Okamoto [51]. Both

TPBS and FAME are instantiated in a type-3 bilinear group setting. Finally, any collision-resistant hash function could be used to instantiate H.

Efficiency. To sign a message m, the signer needs to generate TPBS policy secret key $sk_{TPBS}^{p_{San}}$ for the policy p_{San} which in fact a digital signature over p_{San}, encrypts $p_{San}||sk_{TPBS}^{p_{San}}$ using the ABE scheme public key, hashes the message m, and finally generates a TPBS signature σ_m over m using the generated policy key $sk_{TPBS}^{p_{San}}$ and his TPBS identity key sk_{TPBS}^{Sign}. To verify a message signature pair, the verifier verifies The TPBS signature σ_m. To sanitize a message, the sanitizer performs an ABE decryption to obtain $p_{San}||sk_{TPBS}^{p_{San}}$, modifies the message according to p_{San} to obtain m' and generates a TPBS signature $\sigma_{m'}$ over m' using the policy key $sk_{TPBS}^{p_{San}}$ and his TPBS identity key sk_{TPBS}^{San}. Tracing a signature to its origin requires verifying the sanitizable signature, and running the tracing algorithm of the underlying TPBS scheme. The computation and communication complexities of the instantiated EP3S are as follows. Let $l \times t$ be the size monotone span program that represents the access structure that defines the future possible sanitizers [7] and let the EP3S be initialized with n users and the policy p_{San} be expressed in 1 PPE uniquely defined by $(M, \tilde{N}) \in \mathbb{G} \times \tilde{\mathbb{G}}$ group elements. To sign a message m, The proposed instantiation produces a total signature size of 14 elements in $\mathbb{G}$ + 16 elements in $\tilde{\mathbb{G}}$. Signing costs approximately $44 + M + 6l$ exponentiations in $\mathbb{G}$ in addition to $73 + 2N$ exponentiations in $\tilde{\mathbb{G}}$ plus one hashing of m. More precisely, generating sk_{TPBS}^{San} costs $2 + M$ exponentiations in $\mathbb{G}$ in addition to $2N$ exponentiations in $\tilde{\mathbb{G}}$, The ABE encryption of $p_{San}||sk_{TPBS}^{p_{San}}$ costs $6l$ exponentiations in $\mathbb{G}$ in addition to 3 exponentiations in $\tilde{\mathbb{G}}$, and the TPBS signature costs 42 exponentiations in $\mathbb{G}$ in addition to 70 exponentiations in $\tilde{\mathbb{G}}$. Sanitizing a message costs 42 exponentiations in $\mathbb{G}$ plus 70 exponentiations in $\tilde{\mathbb{G}}$ in addition to 6 pairing operations which are the decryption cost of the ABE encryption to obtain $p_{San}||sk_{TPBS}^{p_{San}}$. Verifying a given

EP3S message signature pair costs approximately a total of 100 pairing operations to verify σ_m. To trace a signature to its origin, at most $n+2$ pairing operations are performed, and the produced proof is of size 16 group elements in $\tilde{\mathbb{G}}$, which costs around 10 exponentiations in $\mathbb{G}$ and 20 exponentiations in $\tilde{\mathbb{G}}$ in addition to the verification cost of the EP3S signature. To verify the output of the tracing algorithm, the Judge performs around 40 pairing operations in addition to the verification cost of the EP3S signature.

Comparing EP3S ***to the existing PB3S***. To the best of our knowledge, the only proposed policy-based sanitizable signature schemes in literature are P3S [89] and UP3S (section 4). In a nutshell, P3S utilizes a policy-based chameleon hash (PCH) and a group signature scheme (GSS) technique as its main building blocks. On the other hand, UP3S adopts a traceable attribute-based signature scheme (TABS) and a rerandomizable digital signature scheme (RDS) as its main building blocks. Unlike EP3S, both proposals specify a static modification policy that controls which message blocks the future sanitizers are allowed to modify. Such a static policy cannot be changed without signing the message again. In EP3S, on the other hand, new fine-grained policies can be delegated with the same message-signature pair. Furthermore, P3S and UP3S do not consider specifying multiple modification policies for the same message or the delegation of sanitizer keys. In terms of security, unlike P3S, both UP3S and EP3S provide unlinkability (section 4). Similar to EP3S, P3S uses a secret key sharing sanitization technique, however, it requires a group manager that controls the sharing of sanitization secret keys among the possible sanitizers which is not required in EP3S. On the other hand, in UP3S, the sanitization rights are controlled by the underlying TABS scheme. Table 6.1 summarizes the comparison between EP3S, P3S, and UP3S.

In terms of performance, we compare EP3S only to UP3S, since P3S does not pro-

Table 6.1: Comparison between EP3S, P3S and UP3S.

	EP3S (this work)	P3S [89]	UP3S (section 4)
Building blocks	TPBS and ABE	PCH and GSS	TABS and RDS
Unlinkability	yes	no	yes
Mod. Policy	granular	static	static
Delegatability	supported	NA	NA
Multiple Mod. Policies	supported	NA	NA
Sanitization technique	secret key sharing	secret key sharing	ABS

NA denotes an unavailable feature/entity.

vide specific performance analysis or benchmarking data. Let UP3S and EP3S be used to sign/sanitize a message according to an $l \times t$ monotone span program and EP3S is used to to sign/sanitize the message in accordance to a policy expressed in one PPE uniquely defined by $(1, 1) \in \mathbb{G} \times \tilde{\mathbb{G}}$ group elements. To sign a message m, UP3S produces a signature of size $(27l + 21)$ elements in $\mathbb{G}$, $(22l + 15)$ elements in $\tilde{\mathbb{G}}$, and $(t + 3)$ elements in $\mathbb{Z}_p$, while the signature size in EP3S is 14 elements in $\mathbb{G}$ and 16 elements in $\tilde{\mathbb{G}}$. The signing cost in UP3S is $(27l + 32)$ modular exponentiations in $\mathbb{G}$ and $(38l + 34)$ modular exponentiations in $\tilde{\mathbb{G}}$, while the signing cost in EP3S is $45+6l$ exponentiations in $\mathbb{G}$ and 75 exponentiations in $\tilde{\mathbb{G}}$. Sanitizing a message in UP3S costs the same as signing a message, while in EP3S, it costs 42 exponentiations in $\mathbb{G}$, 70 exponentiations in $\tilde{\mathbb{G}}$, and 6 pairing operations. Verifying a UP3S signature costs a total of $(32l + 80)$ pairing operations, 1 modular exponentiation in $\mathbb{G}$, and 2 modular exponentiations in $\tilde{\mathbb{G}}$, while it costs 100 pairing operations to verify it in EP3S. Tracing a signature in UP3S costs 2 modular exponentiations in $\tilde{\mathbb{G}}$, while in EP3S, it costs at most $n + 2$ pairing operations, where n is the total number of users. The judge procedure in UP3S performs 4 pairing operations, while in EP3S, it costs around 40 pairing operations. We can see that EP3S produces smaller signatures compared to UP3S. The signing and verification costs in UP3S and EP3S are comparable. However, in terms of sanitization, EP3S incurs a cost more than that of UP3S with 6 pairing operations. Additionally, EP3S exhibits higher tracing and tracing verification costs than UP3S. However,

tracing and verification of tracing results are believed to be executed less frequently than signing, sanitizing, and verification. Hence, this slightly higher cost is acceptable given the high policy expressiveness of EP3S when compared to UP3S.

Chapter 7

V2VFx: A Privacy-Preserving Peer-to-Peer Fair Exchange Framework for Vehicle Energy Trading

In this chapter, we present V2VFx, a privacy-preserving fair exchange protocol designed for peer-to-peer electric vehicle (EV) energy trading. V2VFx addresses key challenges in decentralized energy markets by ensuring fair exchange between energy buyers and sellers while preserving the privacy of trading participants, including their identities, sale offers, locations, and contractual terms. By leveraging zero-knowledge proofs (ZKPs) and micropayments on public blockchains, V2VFx guarantees transaction unlinkability and supports anonymous payments, preventing external entities from linking transactions to specific users.

Electric vehicles (EVs) are revolutionizing the transportation sector by offering energy efficiency and environmental benefits, such as reducing greenhouse gas emissions and reliance on oil. Additionally, EVs can function as distributed energy storage systems, support-

ing renewable energy integration through Vehicle-to-Grid (V2G) applications [18, 82]. The rapid adoption of EVs is exceeding expectations, with projections indicating that they will dominate global car sales by 2030 [84]. Government and industry initiatives, such as the Accelerating to Zero Coalition and the EV100 initiative, are driving this transition [3, 59]. However, the expansion of EV adoption faces challenges, particularly in charging infrastructure. In regions like Canada, a lack of widespread and accessible charging stations has led to range anxiety, deterring potential EV buyers [8,62,68]. To address this issue, Vehicle-to-Vehicle (V2V) charging has emerged as a promising solution, allowing EVs to transfer energy directly [60, 92], enhancing grid stability and promoting greater energy independence [81].

Despite its potential, V2V energy trading introduces *privacy* and *fairness* challenges. Sharing energy between EVs may expose sensitive data, such as location, charging patterns, and financial transactions, leading to security concerns [64, 80]. Additionally, ensuring fairness in trading sessions is crucial, as participants must be protected from premature transaction aborts. Blockchain-based approaches have been explored to address these concerns [11, 64, 66, 72, 79, 80, 91, 95, 97, 99], but existing solutions often rely on trusted third parties (TTPs), which undermine decentralization, introduce single points of failure, and limit scalability [11, 72, 80, 97]. Moreover, many prior works lack formal security proofs [72], failing to provide comprehensive privacy protection. To overcome these limitations, we propose *V2VFx*, a privacy-preserving fair exchange framework for V2V energy trading. *V2VFx* eliminates TTP dependencies, ensures transaction unlinkability, and leverages public blockchain technology to provide a secure, decentralized, and transparent energy-trading experience.

Furthermore, We propose a Universal Composability (UC) security model to formally

define and prove that V2VFx satisfies essential privacy and fairness properties. Additionally, we implement V2VFx on the Ethereum blockchain to evaluate its performance, analyzing the execution time of transactions and the computational complexity of its cryptographic components.

Compared to existing fair exchange and privacy-preserving energy trading protocols, V2VFx offers a comprehensive solution that ensures both security and efficiency without compromising decentralization. It extends previous models by incorporating enhanced privacy guarantees and ensuring that transactions remain unlinkable while maintaining fairness in energy exchanges.

In the following, we redefine Simulation Sound Extractable Non - Interactive Zero - Knowledge Proof System (SSE-NIZK) which will be used throughout this chapter for simplicity.

An SSE-NIZK system enables a prover with a witness w to non-interactively prove the truthfulness of a statement st to a verifier without disclosing why [57]. For st in an $\mathbb{NP}$-language $\mathcal{L}$ such that (st, w) is within a relation $\mathbb{R}$ associated with $\mathcal{L}$, an SSE-NIZK consists of six polynomial-time algorithms, NIZK = {Setup, SimSetup, Prove, SimProve, Verify, Extr} , defined as follows.

- Setup: This algorithm outputs the common reference string of the system, crs, obtained from $\mathsf{Setup}(1^{\lambda})$.
- SimSetup: This algorithm outputs a simulated crs and a trapdoor, (crs, tr), obtained from $\mathsf{SimSetup}(1^{\lambda})$.
- Prove: Given crs, st, and w for the relation $\mathbb{R}$, this algorithm outputs a proof π, obtained from $\mathsf{Prove}(crs, st, w)$.

- SimProve: With crs, st, and the trapdoor tr, this algorithm outputs a proof π, obtained from $\mathsf{SimProve}(crs, st, tr)$.
- Verify: This algorithm verifies the proof π on st, returning $\{\top, \bot\}$ from $\mathsf{Verify}(crs, st, \pi)$.
- Extr: This procedure extracts w from π using tr, resulting from $\mathsf{Extr}(crs, tr, st, \pi)$.

SSE-NIZK schemes ensure zero knowledge, implying negligible success for an adversary distinguishing between a proof for a statement st with witness w and a simulated one. Additionally, they provide simulation-extractability, making it difficult for an adversary to produce a verifiable proof for a statement st with witness w where $\mathbb{R}(st, w) = 0$. The formal definitions of these security notions are available in [57].

Payment Channels

Before presenting the V2VFx framework, we briefly discuss payment channels, an essential mechanism enabling efficient and fair energy trading.

Payment channels are off-chain mechanisms that allow two parties to perform multiple transactions without burdening the blockchain for each payment, requiring only two on-chain interactions: channel opening and channel settlement. This technique significantly reduces transaction fees, confirmation delays, and blockchain congestion, making it particularly suitable for micropayment scenarios such as vehicle-to-vehicle (V2V) energy trading [23, 78].

In V2VFx, we implement a unidirectional, hashchain-based payment channel optimized for energy trading. Before trading begins, the buyer locks a deposit into a blockchain smart contract and commits to a hashchain, where each hash preimage represents a unit payment [83]. During energy transfer, the buyer reveals successive preimages off-chain as proofs of

incremental payment for delivered energy. At the end of the session, the seller redeems the last revealed preimage on-chain to claim payment proportional to the energy delivered.

This approach ensures fairness, since payments are proportional to delivered energy even if communication is disrupted. It also enhances privacy, as only the initial commitment and final settlement appear on-chain, and improves scalability by minimizing blockchain interaction. By combining escrowed deposits with hashchain micropayments, V2VFx achieves an efficient, privacy-preserving, and decentralized fair exchange protocol for vehicular energy trading.

7.1 V2VFx Framework

V2VFx is designed to facilitate a fair exchange for both parties involved in a peer-to-peer energy trading transaction. An EV seeking to charge its battery can utilize V2VFx to purchase the required energy from another EV with excess battery charge (V2V).

Throughout this chapter, we use P_b and P_s to denote the buyer's and seller's blockchain public keys, respectively. Each public blockchain address is associated with a secret key, which is implicitly used when sending a message to the blockchain. The buyer possesses a key pair $(sk^b_{\mathsf{PRF}}, pk^b_{\mathsf{PRF}})$ for a Pseudo-Random Function (PRF). The current time is represented by T. For secure key exchange, the buyer and seller use Diffie-Hellman (DH) key pairs, denoted as $(sk^b_{\mathsf{DH}}, pk^b_{\mathsf{DH}})$ for the buyer and $(sk^s_{\mathsf{DH}}, pk^s_{\mathsf{DH}})$ for the seller. The shared symmetric secret keys established between the buyer and seller are denoted as ssk^b_s (buyer to seller) and ssk^s_b (seller to buyer), where the correctness of the DH protocol ensures that $ssk^b_s = ssk^s_b$. A zero-knowledge proof for a specific language $\mathcal{L}_*$ is denoted by π_*. Additionally, $\mathsf{H}(.)$ represents a collision-resistant one-way hash function.

System model and overview As depicted in Fig. 7.1, the V2VFx system encompasses three primary entities; *Energy Buyers*, *Energy Sellers*, and the *Blockchain*. These entities can communicate through both on-chain and off-chain communication protocols, covering data flow, flow of charge, and monetary transactions. The logic of such communication can be described in four conceptual steps; *Bidding*, *Evaluate&Commit*, *Off-Chain Exchange*, and *Pay&Refund*. In what follows, a brief description of each step is given.

Bidding. This is the protocol's initial step where the energy buyer submits a buying request to the blockchain, specifying the required energy amount within a defined service area and an expiration time for the request. During pre-expiration time, sellers within the service area can respond to the request with a concealed anonymized offer, containing the seller's location and the offered energy sale value.

Evaluate&Commit. After the request expiration, the buyer evaluates the received offers and selects the best offer based on preferences such as seller proximity or preferred time slot. Subsequently, the buyer commits to the chosen offer by sending a hiding and binding commitment of the offer contractual terms and payment to the blockchain. This commitment enables the blockchain contract to hold the agreed-upon sale value until the energy exchange is completed. Subsequently, the contract transactions are scraped by the seller's application to get notified.

Off-Chain Exchange. Upon reaching the seller's location, the seller authenticates the buyer and initiates an off-chain energy exchange session using their respective hardware. Throughout the session, the seller collects payment tokens in exchange for each unit of transferred power to ensure fair exchange.

Pay&Refund. In this phase, the seller receives payment by presenting the payment tokens received in the previous phase as proof by submitting a payment request to the

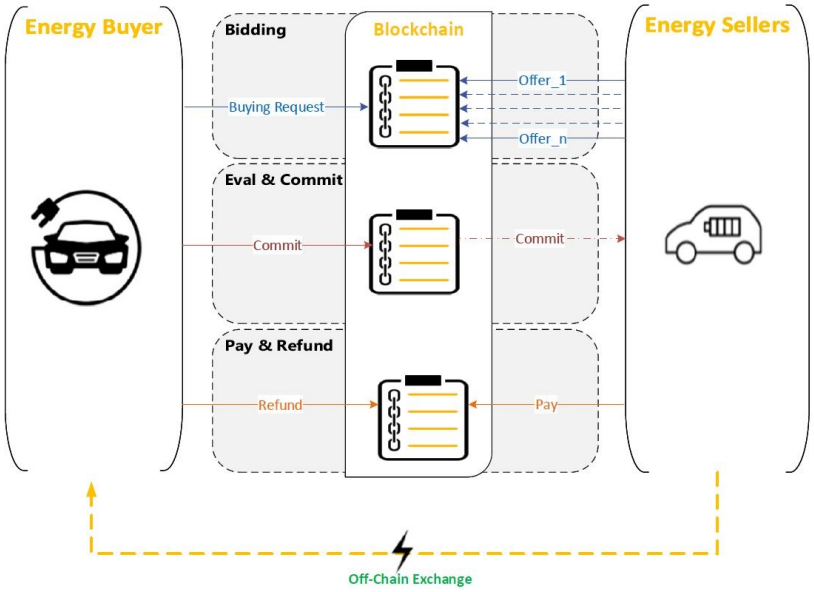

Figure 7.1: V2VFx system architecture.

blockchain contract. Once the seller's payment is processed, the buyer can request a refund for any remaining amount from their previous commitment by submitting a refund request to the blockchain.

Security model

In what follows, we list our adversarial assumptions.

- *Energy buyers and sellers*. We assume that energy buyers and sellers may act selfishly to maximize their financial interests. This includes the possibility of arbitrary deviations from the prescribed protocol or premature aborting.

- *Blockchain*. The blockchain is considered a conceptual trusted party, relied upon for correctness and availability but not deemed trustworthy for privacy.

- *Off-chain applications.* All off-chain applications are assumed to be trusted client-side code, primarily functioning as data scrapers and interacting with the blockchain.

It is worth noting that blockchains inherently incorporate a discrete concept of time, where a

clock advances with each newly mined block. The presence of this reliable clock is essential for achieving fair exchange in our protocols.

Security notions In V2VFx, we consider the following security properties.

- *Offer Privacy*. Sellers' offers remain private from each other and the public. Each seller provides their offer without knowledge of other offers, ensuring independent and fair offer selections.

- *Location Privacy*. None of the participants learns the exact position of the energy buyer. Furthermore, the buyer and seller locations cannot be tracked over time.

- *Contractual Terms Privacy*. Agreed-upon conditions and obligations related to payments are kept secret from the public and any other energy trade participants, except for the involved buyer and seller. These terms include the total payment amount ($\$val$[1]) and the number of agreed-upon micro-payments (N).

- *Identity Privacy*. The true identities of the buyer and seller are kept private and no adversary can recover any information about the identity of the participants in a trading session.

- *Unlinkability*. The linkage between the energy buyer and the seller is obscured from the public, safeguarding against the tracking of money flow. This ensures that a payment sender (buyer) cannot be associated with the corresponding recipient (seller).

- *Fair Exchange*. V2VFx offers a mechanism for fair asset exchange between parties, ensuring that, at the end of the protocol execution, either each party holds the counterpart's

[1]the symbol $ is only adopted for readability (to distinguish variables associated with money and other variables) and does not have special meaning or significance.

asset or neither of them does. The maximum loss for either party is limited to a small fraction of the transaction payment/charge. V2VFx also provides mechanisms for preventing coin freezing and enforcing payments and refunds upon premature aborts.

Note that maintaining the privacy of offers serves to diminish strategic bidding behaviors. In this setting, bidders are less susceptible to external influences, as they lack real-time information about ongoing bids. This fosters more authentic bids that genuinely reflect each bidder's valuation of the item. Additionally, the confidentiality of offers aids in mitigating collusion among bidders, as they are devoid of opportunities for direct communication during the bidding process. This, in turn, contributes to fostering a fairer auction environment.

V2VFx ideal functionality We articulate the aforementioned security properties through an ideal functionality within the universal composition framework. Our approach aligns with the Universal Composability (UC) model introduced in [16, 67, 95]. In Figure 7.2, we provide an ideal functionality $\mathcal{F}_{V2VFx}$ to capture all security requirements we defined earlier. $\mathcal{F}_{V2VFx}$ is composed of six operations, namely MintCoin, BuyRequest, SellResponse, CommitSale, SellerPayment, and Refund. Moreover, $\mathcal{F}_{V2VFx}$ utilizes a clock that advances in rounds so that timeout operations can be implemented after a specific number of rounds to protect against premature aborts and coin freezing.

In what follows, we give a brief description of each of $\mathcal{F}_{V2VFx}$ operations:

- MintCoin: After the private sets Coins, Reqs, Offrs, and ComCoins are initialized in Init, this operation allows the buyer, whose identity is P_b, to transfer an actual amount of money from their blockchain public ledger to the private ledger Coins, where the buyer can decide the value $\$val$ of each coin. The adversary $\mathcal{A}$ learns the buyer's identity P_b and the value(s) of the coin(s) $\$val_i$.

Ideal Functionality $\mathcal{F}_{V2VFx}$

Init: Coins:=[], Reqs:=[], Offrs:=[], ComCoins:=[]

MintCoin: Upon receiving (MintCoin, $\$val_1, \ldots, \$val_n$) from P_b:
Assert Ledger(P_b)$\geq \sum_{i=1}^{n} \$val_i$
Send (MintCoin,$P_b, \$val_1, \ldots, \$val_n$) to $\mathcal{A}$

for $i \in \{1, \ldots, n\}$
Ledger(P_b) = Ledger(P_b)-$\$val_i$
Add $(P_b, \$val_i)$ to Coins

BuyRequest: Upon receiving (BuyRequest, id_b, e, T_{exp}, w_b) from some P_b:
Assert $id_b \notin$ Reqs
Assert $T < T_{exp}$
$\mathcal{R} \leftarrow w_b$
Send (BuyRequest,$id_b, e, \mathcal{R}, T_{exp}$) to $\mathcal{A}$

Add $(id_b, e, \mathcal{R}, T_{exp})$ to Reqs

SellResponse: Upon receiving (SellResponse, id_b, e, $\mathcal{R}_i$, T_{exp},w_s, N,$\$val$,$slot$,$T_{com}$) from some P_s:
Assert $(id_b, e, \mathcal{R}, T_{exp}) \in$ Reqs
Assert $T < T_{exp}$
Send (SellResponse, id_b, T_{com}) to $\mathcal{A}$

Add $(id_b, w_s, N, \$val, slot, T_{com})$ to Offrs

CommitSale: Upon receiving (CommitSale, ϕ_{pay}, id_b, N, $\$val$, P_b, P_s, T_{com}) from P_b
Assert $T < T_{com}$
Assert $(P_b, \$val) \in$ Coins
Send (CommitSale, ϕ_{pay}, T_{com}) to $\mathcal{A}$

Remove $(P_b, \$val)$ from Coins
Add $(\phi_{pay}, id_b, N, \$val, P_b, P_s, T_{com})$ to ComCoins

SellerPayment: Upon receiving (SellerPayment, ϕ_{pay}, id_b, N, $\$val$, P_b, P_s, T_{com},h_l,l,$\$val'$) from P_s
Assert $(\phi_{pay}, id_b, N, \$val, P_b, P_s, T_{com}) \in$ ComCoins
Assert $l \in$ ComCoins$(id_b) = \varnothing$ Assert $T < T_{com}$
Assert $\phi_{pay}(h_l, l) = true \wedge l \leq N$
Assert $\$val' = \$val \times \frac{l}{N}$
Send (SellerPayment, ϕ_{pay}, h_l, l) to $\mathcal{A}$

Add l to ComCoins(id_b)
Add $(P_s, \$val')$ to Coins

Refund: Upon receiving (Refund, ϕ_{pay}, id_b, N, $\$val$, P_b, P_s, T_{com}, l, $\$val''$) from P_b
Assert $(\phi_{pay}, id_b, N, \$val, P_b, P_s, T_{com}, l) \in$ ComCoins
Assert $T > T_{com}$
Send (Refund, ϕ_{pay}) to $\mathcal{A}$

if $l = \varnothing$ set $l = 0$
Assert $\$val'' = 1 - \$val \times \frac{l}{N}$
Add $(P_b, \$val'')$ to Coins

The instructions inside rectangle frames are deferred to the next round.

Figure 7.2: The ideal functionality $\mathcal{F}_{V2VFx}$.

- BuyRequest: This operation allows an energy buyer to submit a buying request. Each request has a request ID id_b, and the required amount of energy e, the region $\mathcal{R}_i$ where the buyer would like to buy the energy, and the request expiry time T_{exp}. Once a request is received, it is logged into Reqs after some basic assertions. During this operation, the adversary $\mathcal{A}$ has access to all the request data but does not learn the identity of the buyer P_b nor their precise location information w_b.

- SellResponse: This operation allows sellers to respond to suitable buyer requests with some offers. The offer contains the buyer's request ID id_b, the seller's location w_s, the energy sale value $\$val$, the number of payments N, the charging time slot $slot$, and the time by which the seller will collect their money T_{com} (which will later be used by the Refund operation to prevent coin freezing). The request is added to Offrs after some basic assertions, where the adversary $\mathcal{A}$ only learns id_b and T_{com} and has no access to the seller's identity P_s nor their precise location w_s.

- CommitSale: This operation enables the buyer to commit to an offer of their choice. The commitment includes a payment coin $coin$ of the value in the offer $\$val$ and a payment promise ϕ_{pay}. After some assertions, the payment coin $coin$ is removed from the private list Coins to prevent double-spending, and the full commitment is added to ComCoins, where the adversary $\mathcal{A}$ only learns the payment promise ϕ_{pay} and T_{com} and does not learn any other pieces of information beyond that.

- SellerPayment: This operation enables sellers to claim their payment after the energy exchange, as long as T_{com} has not expired. The seller submits the payment tokens (h_l, l) they obtained during the energy exchange that authorize them to claim the value of the energy exchanged $\$val'$ using a coin $coin'$. SellerPayment asserts that the claim is true

by verifying (h_l, l) against the stored payment promise ϕ_{pay} in ComCoins and ensuring $\$val'$ is the correct amount of money for the actual energy exchanged. Then, the new coin $coin'$ is added to Coins and is tied to the seller's identity P_s. During this operation, the adversary $\mathcal{A}$ only learns the payment promise ϕ_{pay} and claim witnesses (h_l, l) and does not learn the seller's identity P_s or the value of the new coin $\$val'$.

- Refund: This operation allows the buyer to refund any remaining amount of their previously committed coin $coin$ after the expiration of T_{com}. The buyer submits a coin $coin''$ of value $\$val''$, which represents the remaining amount of $coin$ if the energy exchange is partially performed or the full value of $coin$ if the energy exchange was never performed. Refund asserts that the claim is true by correlating the submitted parameters with their corresponding value in ComCoins. If true, the new coin $coin''$ is added to Coins and is tied to the buyer's identity P_b. During this operation, the adversary $\mathcal{A}$ only learns the payment promise ϕ_{pay} and does not learn the buyer's identity P_b or the value of the new coin $\$val''$.

In what follows, we demonstrate how $\mathcal{F}_{V2VFx}$ encapsulates the aforementioned security notions.

In SellResponse, when a seller responds to an energy buyer request, the adversary $\mathcal{A}$ gains no knowledge about the proposed offer or terms and conditions. It only learns the original request ID id_b and the premature abortion prevention timer of the offer T_{com}, hence maintaining *offer privacy*. Throughout the protocol phases, particularly in the BuyRequest and SellResponse phases, neither the exact buyer nor the seller locations are disclosed to $\mathcal{A}$ who only learns the energy sale's service area $\mathcal{R}_i$ which ensures *location privacy*. When a buyer calls CommitSale, the adversary cannot observe the agreed-upon money value of the energy trade, $\$val$, or the agreed-upon number of payments, N, thus guaranteeing *Contrac-*

tual Terms Privacy. Throughout the protocol execution, the buyer's and seller's blockchain identities P_b and P_s are kept secret from the adversary, except during the MintCoin procedure. However, the disclosed P_b in MintCoin is not used in subsequent steps, ensuring it is not linked to further procedures. Specifically, in the CommitSale phase, neither the identity P_b that owns the minted coin nor the identity P_s that can claim the coin, is sent to $\mathcal{A}$. Similarly, in the SellerPayment and Refund phases, P_b and P_s are not revealed to $\mathcal{A}$. Therefore $\mathcal{F}_{V2VFx}$ models our notion of *Identity Privacy*. Moreover, as identity information is not revealed to $\mathcal{A}$ during the BuyRequest, SellResponse, CommitSale, SellerPayment, and Refund phases, attempts for linking the buyer and seller, as defined in our notion of *Unlinkability* becomes infeasible. This holds even if certain transactions are linked, notably through the same id_b in the BuyRequest\SellResponse phase or through the same h_0 in the CommitSale\SellerPayment\Refund phases.

$\mathcal{F}_{V2VFx}$ utilizes a gradual release mechanism between the buyer and the seller. Specifically, (h_l, l) represents a payment token provided by the buyer and given to the seller. This token can be checked against the buyer's committed payment promise ϕ_{pay} in the CommitSale phase. Throughout the off-chain energy trading phase, the buyer consistently transmits (h_l, l) to the seller for $l = 1, 2, \ldots, N$, where $h_l = H(h_{l-1})$, functioning as the payment token for the subsequent rounds of energy exchange. The seller, in turn, validates (h_l, l) against ϕ_{pay} and, based on the result, decides whether to continue the trading session or abort. This design limits the maximum potential loss for either party to $\frac{\$val}{N}$ or its equivalent amount of energy. Furthermore, $\mathcal{F}_{V2VFx}$ requires the energy trading session and the seller payment to be completed by time $T < T_{com}$ to prevent coin freezing and enforce refunds upon premature aborts.

7.2 Construction of V2VFx

We adopt the blockchain model proposed by Kosba et al. [67], which addresses relevant threats and provides program wrappers to simplify protocol descriptions. In this model, party identifiers inherently serve as pseudonyms by default. For instance, by referring to an '*anonsend*', we imply sending a message anonymously from any pseudonym. Moreover, '*anonreceive*' is described by receiving '*from some* P'. Conversely, specifying receiving a message '*from* P' denotes the knowledge of a predetermined pseudonym P, usually evident from context. Users can generate a finite number of pseudonyms, with each pseudonym owned by the generating party. The mapping of pseudonyms to true identities remains concealed from adversaries.

Refined private ledger. V2VFx expands upon the concept of a private ledger facilitating private currency transfers. This private ledger is constructed atop a public ledger. In particular, V2VFx embraces the mint and pour terminologies outlined in [67], which is based on the foundational concept presented in [90]. In essence, the mint operation enables a user, denoted by P, to transfer funds from their public ledger to a private pool. Each transfer generates a private coin for user P, associated with a value $\$val$. This value is deducted from P's public ledger and held by the on-chain contract. On the other hand, with the help of a manager, the pour operation enables the coin owner, P, to transfer coin ownership to another user, denoted as $\tilde{P}$, utilizing a zero-knowledge proof. This proof validates the coin to be transferred, its associated value, and the identity of P in terms of the coin's associated public key. Note that we have modified the coin format and its structure in the private pool. This modification is made because our proposed protocol eliminates the need for a manager, and the coin's opening is not shared with any party other than the coin's owner. Therefore,

there is no necessity to associate the coin with the owner's public key in the private pool. This change is implemented for the sake of simplicity in the required zero-knowledge proofs.

The V2VFx system comprises a Buyer protocol ***B*uyerP** (Fig. 7.3), an on-chain smart contract ***V*2VFx BC** (Fig. 7.4), a Seller protocol ***S*ellerP** (Fig. 7.5), and an off-chain energy exchange mechanism ***O*ff-ChainP** (Fig. 7.6). The energy buyer and seller execute ***B*uyerP** and ***S*ellerP** through client-side code, respectively. The actual energy exchange occurs off-chain using the underlying hardware of both the buyer and seller and is controlled by ***B*uyerP** and ***S*ellerP**. Finally, ***V*2VFx BC** is executed on-chain by blockchain validators. Energy charging using V2VFx proceeds over seven phases, namely *Coin minting*, *Request for offers*, *Offer response*, *Evaluation&Commit*, *Off-chain exchange*, *Seller payment*, and *Buyer refund*. Such phases span over the aforementioned four system protocols.

Coin minting phase In this phase, a system user can transfer funds from their public ledger denoted by Ledger to the private pool Coins. Typically, the buyer initiates this phase before engaging in an energy transaction. In this context, The buyer initiates the $\mathsf{MintCoin}$ function in ***B*uyerP** in Fig. 7.3, where the buyer's true-identity P_b is employed to mint a new coin(s). For each $\$val_i$, where $i = 1, \ldots, n$, that the buyer intends to generate a coin for, freshly generated random values s_i and sn_i are utilized to compute the coin's value $coin_i = \mathsf{Comm}(s_i, sn_i||\$val_i)$. Subsequently, the tuple $s_i, sn_i, \$val_i, coin_i$ is added to the buyer's P_b local wallet $Wallet$, and a zero-knowledge proof π_{coin} is computed for the relation $\mathbb{R}_{coin} \subseteq \mathcal{L}_{coin}$. This proof informally verifies that the value of $coin_i$ is indeed $\$val_i$ and is formally defined as:

$$((coin_i, \$val_i), (s_i, sn_i)) \in \mathbb{R}_{coin} \Leftrightarrow$$

$$coin_i = \mathsf{Comm}(s_i, sn_i||\$val_i)$$

Buyer Protocol (*B*uyerP)

Init(1^λ): $Wallet := []$, SOffrs:=[], $sk^b_{PRF} \xleftarrow{\$} \{0,1\}^\lambda$

MintCoin($\$val_1 \dots \$val_n$):
for $i \in \{1, \dots, n\}$
$(s_i, sn_i) \xleftarrow{\$} \{0,1\}^\lambda$
$coin_i = \mathsf{Comm}(s_i, sn_i || \$val_i)$
Add $\{s_i, sn_i, \$val_i, coin_i\}$ to $Wallet$
$st_{coin} = (coin_i, \$val_i)$
$w_{coin} = (s_i, sn_i)$
$\pi_{coin} = \mathsf{NIZK.Prove}(crs, st_{coin}, w_{coin})$
Send as P_b (MintCoin, $coin_1, \dots, coin_n, \$val_1, \dots, \$val_n, \pi_{coin_1}, \dots, \pi_{coin_n}$) to ***V2VFx BC***

BuyRequest($e \in (\mathcal{Q}^+ \setminus \{0\}), T_{exp}, w_b \in \mathcal{W}$):
$\mathcal{R} \leftarrow w_b$
$(sk^b_{DH}, pk^b_{DH}) \xleftarrow{\$} \mathsf{DH.KeyGen}(1^\lambda)$
$id_b \xleftarrow{\$} \{0,1\}^\lambda$
anonsend (BuyRequest, $id_b, pk^b_{DH}, e, \mathcal{R}_i, T_{exp}$) to ***V2VFx BC***

Evaluate(id_b, pk^s_{DH}, ct):
if $T < T_{exp}$
SOffrs = SOffrs $\cup \{pk^s_{DH}, ct\}$
if $T \geq T_{exp}$
$\forall j = (1 \dots n) : pk^{s_j}_{DH}, ct_j \in$ SOffrs
$sk^b_{s_j} = \mathsf{DH.KeyEx}(pk^{s_j}_{DH}, sk^b_{DH})$
$w_{s_j} || N_j || \$val_j || pk^{s_j}_{PRF} || slot_j || T_{com_j} = \mathsf{Sym.Dec}\ (ssk^b_s, ct_j)$
$(N, \$val, pk^s_{PRF}, slot, T_{com}) \leftarrow$ Eval (.)

CommitSale(id_b,sk^b_{PRF},N, $\$val$, pk^s_{PRF}, $slot$, T_{com}):
Find $(s, sn, coin) \in Wallet$ of value ($\$val$)
$b_{coin} \leftarrow MT(coin)$
Remove $(s, sn, \$val, coin)$ from $Wallet$
$pk^b_{PRF} = \mathsf{PRF}(sk^b_{PRF}, id_b)$
$a \xleftarrow{\$} \{0,1\}^\lambda$
$cm = \mathsf{Comm}(a, id_b, N, \$val)$
hash chain of length $N+1$: $h_N \rightarrow h_{N-1} \rightarrow \dots \rightarrow h_1 \rightarrow h_0$ where $h_i = \mathsf{H}(h_{i+1})$
$st_{com} = (MT.root, sn, cm)$
$w_{com} = (coin, b_{coin}, s, \$val, id_b, a)$
$\pi_{com} =$ NIZK.Prove (crs st_{com}, w_{com})
anonsend (CommitSale, h_0, pk^b_{PRF}, pk^s_{PRF}, sn, cm, T_{com}, π_{com}) to ***V2VFx BC***

BuyerRefund($h_0, pk^b_{PRF}, cm, T_{com}, l, sk^b_{PRF}, a, id_b, N, \$val$):
Assert $T \geq T_{com}$
$s'' \xleftarrow{\$} \{0,1\}^\lambda$
$coin'' = \mathsf{Comm}(s'', sn'' || \$val \times (1 - \frac{l}{N}))$
$st_{ref} = (pk^b_{PRF}, cm, coin'', l)$
$w_{ref} = (sk^b_{PRF}, s'', sn'', a, id_b, N, \$val)$
$\pi_{ref} = \mathsf{NIZK.Prove}(crs, st_{ref}, w_{ref})$
Add $\{s'', sn'', \$val \times (1 - \frac{l}{N}), coin''\} \rightarrow$Wallet
anonsend (BuyerRefund, $h_0, coin'', \pi_{ref}$) to ***V2VFx BC***

Figure 7.3: Buyer protocol.

Finally, a request for the coin(s) is generated and sent to ***V*2VFx BC** in Fig. 7.4 as (MintCoin, $coin_1, \dots, coin_n, \$val_1, \dots, \$val_n, \pi_{coin_1}, \dots, \pi_{coin_n}$) from P_b.

The MintCoin function in Fig. 7.4 first validates that P_b has sufficient funds in its public ledger Ledger(P_b) which is greater than or equal to $\sum_{i=1}^{n} \$val_i$. It verifies that π_{coin} is valid with respect to the relation $\mathbb{R}_{coin} \subseteq \mathcal{L}coin$ and finally adds $coin_i$ to the private pool Coins and appends $coin_i$ to an on-chain Merkle tree MT. Note that $\$val_i$ has to be disclosed in the MintCoin function to ensure P_b has enough funds that covers $coin_i$. Nevertheless, s_i and sn_i remain secret and are never disclosed publicly, serving as proof of ownership for $coin_i$. Additionally, it is noteworthy that the presence of s_i ensures that $coin_i$ could not be identified at the time of spending, where sn is revealed. Note that coin minting could be invoked in a pre-processing stage as it is independent of the bidding/trading session

Request for offers phase In this phase, a buyer aiming to purchase a specific energy amount $e \in \mathcal{Q}^+ \setminus 0$ where $\mathcal{Q}^+$ denotes the set of positive energy amount for EVs, executes the BuyRequest function in ***B*uyerP** in Fig. 7.3. The BuyRequest function takes as input the energy amount required e, and the buyer's precise location $w_b \in \mathcal{W}$, where $\mathcal{W}$ denotes the set of all possible world positions e.g., latitude and longitude. It then determines the service region $\mathcal{R}_i$ that includes the buyer's location w_b, and generates a new DH public-secret key pair (sk^b_{DH}, pk^b_{DH}). Subsequently, it generates a unique random request ID id_b and sends the request to be registered with ***V*2VFx BC** in Fig. 7.4 along with a request expiry time T_{exp} as $(\text{BuyRequest}, id_b, pk^b_{DH}, e, \mathcal{R}, T_{exp})$ anonymously. Upon receiving the request, The BuyRequest function in ***V*2VFx BC** first checks that $\mathcal{R}_i$ is within a serviceable area $\mathcal{R}_i \in \mathcal{R}$ where $\mathcal{R}$ denotes the set of serviceable areas $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$, and that the current time T is less than T_{exp}. It then logs the request in the Reqs list.

Offer response phase In this phase, a seller intending to respond to a request logged in Reqs in Fig. 7.4 with an offer executes the SellResponse function in ***S*ellerP** in Fig. 7.5. Taking the seller's location w_s and the offer $(id_b, pk^b_{DH}, e, \mathcal{R}, T_{exp})$ as inputs, the Sell-

V2VFx Blockchain Contract (*V2VFx BC*)

Init(1^λ): Coins:=[], SpentCoins:=[], Reqs:=[], Offrs:=[], ComCoins:=[]
$CRS \leftarrow$ NIZK.Setup(1^λ)
$pp \leftarrow$ DH.Setup(1^λ)
$MT \leftarrow []$

MintCoin($coin_1$,...,$coin_n$,$\$val_1$,...,$\$val_n$,$\pi_{coin_1}$,...,$\pi_{coin_n}$) from P_b:
Assert $Ledger(P_b) \geq \sum_{i=1}^{n} \$val_i$
for $i \in \{1, \ldots, n\}$
Assert NIZK.Verify($crs, st_{coin}), \pi_{coin_i}$
$Ledger(P_b) = Ledger(P_b) - \$val_i$
Add $coin_i$ to Coins
Add $coin_i$ to MT

BuyRequest($id_b, pk^b_{DH}, e, \mathcal{R}, T_{exp}$) from some P:
Assert $\mathcal{R} \in \mathcal{S}$
Assert $id_b \notin$ Reqs
Assert $T < T_{exp}$
Add $(id_b, pk^b_{DH}, e, \mathcal{R}, T_{exp})$ to Reqs

SellResponse(id_b, pk^s_{DH}, ct) from some P:
Assert $id_b \in$ Reqs
Assert $T < T_{exp}(id_b)$
Add (id_b, pk^s_{DH}, ct) to Offrs

CommitSale(h_0,pk^b_{PRF},pk^s_{PRF},sn,cm,T_{com},π_{com}) from some P:
Assert $T < T_{com}$
Assert $sn \notin$ SpentCoins
Assert NIZK.Verify(crs, st_{com}, π_{com})
Add sn to SpentCoins
$r \xleftarrow{\$} \{0, 1\}$
Add $(h_0, r, pk^b_{PRF}, pk^s_{PRF}, cm, T_{com})$ to ComCoins

SellerPayment($h_0, h_l, l, coin', \pi_{pay}$) from some P:
Assert $(X = \{h_0, r, pk^b_{PRF}, pk^s_{PRF}, cm, T_{com}\}) \in$ ComCoins for h_0
Assert $T < X.T_{com}$
Assert X.l=$\varnothing$
Assert for $i = (l, \ldots, 0)$, $h_{i-1} = \mathsf{H}(h_i)$
Assert NIZK.Verify(crs, st_{pay}, π_{pay})
Set $X.l = l$
Add $coin'$ to Coins
Add $coin'$ to MT

BuyerRefund($h_0, coin'', \pi_{ref}$) from some P:
Assert $(X = \{h_0, r, pk^b_{PRF}, pk^s_{PRF}, cm, T_{com}, l\}) \in$ ComCoins for h_0
Assert $T \geq X.T_{com}$
if $X.l = \varnothing$, set $X.l = 0$
AssertNIZK.Verify(crs, st_{ref}, π_{ref})
Add $coin''$ to Coins
Add coin" to MT

Figure 7.4: Blockchain V2VFx contract.

Response function first ensures that w_s is within the requested service region $\mathcal{R}_i$ and that the current time T is less than to T_{exp}. It then generates a DH public-secret key

Seller Protocol (*SellerP*)

Init(1^λ): $Wallet := []$, $sk^s_{PRF} \xleftarrow{\$} \{0,1\}^\lambda$

SellResponse($w_s \in \mathcal{W}$,id_b,sk^s_{PRF},pk^b_{DH},e,T_{char},$\mathcal{R}_i$,T_{exp}):
Assert $w_s \in \mathcal{R} \wedge T < T_{exp}$
$pk^s_{PRF} = \mathsf{PRF}\{sk^s_{PRF}, id_b\}$
$(sk^s_{DH}, pk^s_{DH}) \xleftarrow{\$} DH.KeyGen(1^\lambda)$
$(N, \$val, slot, T_{com}) \xleftarrow{\$} \mathsf{F}(e, T_{char})$
$ssk^s_b = DH.KeyEx(pk^b_{DH}, sk^s_{DH})$
$ct = Sym.Enc(ssk^s_b, w_s||N||\$val||pk^s_{PRF}||slot||T_{com})$
Save $id_b, \$val, pk^s_{PRF}, slot, T_{com}$ for later
anonsend (SellResponse,id_b,pk^s_{DH},ct) to ***V2VFx BC***

SellerPayment(h_0,pk^s_{PRF},cm,T_{com},h_l,l,sk^s_{PRF},a,id_b,N,$\$val$):
Assert $T < T_{com}$
$s' \xleftarrow{\$} \{0,1\}^\lambda$
$coin' = \mathsf{Comm}(s', sn'||\$val \times \frac{l}{N})$
$st_{pay} = (pk^s_{PRF}, cm, coin', l)$
$w_{pay} = (sk^s_{PRF}, s', sn', a, id_b, N, \$val)$
$\pi_{pay} = \mathsf{NIZK.Prove}(crs, st_{pay}, w_{pay})$
Add $\{s', sn', \$val \times \frac{l}{N}, coin'\}$ to $Wallet$
anonsend (SellerPayment, h_0, h_l, l, $coin'$, π_{pay}) to ***V2VFx BC***

Figure 7.5: Seller protocol.

pair (sk^s_{DH}, pk^s_{DH}) and a PRF public key $pk^s_{PRF} = \mathsf{PRF}\{sk^s_{PRF}, id_b\}$ where sk^s_{PRF} is the seller's secret key of the PRF function. The function creates an offer for the buy request based on the requested energy amount e using the local function $\mathsf{F}(e)$. The offer includes the total number of micro-payments N, the sale value $\$val$, the possible charging time slot $slot$, and the time required by the seller to collect the payment T_{com}, where T_{com} is employed to prevent coin freezing. Subsequently, the SellResponse function generates a symmetric encryption key ssk^s_b using the buyer's public DH key pk^b_{DH} and the seller's DH secret key sk^s_{DH}. This key is then used to create ct as the encryption of $w_s||N||\$val||pk^s_{PRF}||slot||T_{com}$. Finally, the function sends the response to be registered

with ***V*2VFx BC** anonymously as $(\mathsf{SellResponse}, id_b, pk^s_{DH}, ct)$. It is worth noting that the function $\mathsf{F}(.)$ can be configured with specific parameters to optimize the seller's gain from each sale. For instance, the selling price of a small amount of energy may differ from that of large amounts, or the selling price may depend on the charging time slot. Additionally, $\mathsf{F}(.)$ can generate multiple offers for the same request, each with different selling values according to the charging time slot.

Upon receiving the request (id_b, pk^s_{DH}, ct), the SellResponse in Fig. 7.4 first checks whether the request ID id_b is logged in Reqs and retrieves its corresponding T_{exp} to assert that the request is not expired, i.e., the current time $T \leq T_{exp}$ then it stores in the public list offrs. Note that given the approximate knowledge of the power unit price, the sales value $\$val$ can be deduced from the number of payments N. Accordingly, we conceal N in this phase and all subsequent phases.

Evaluate and Commit Phase In this phase, the buyer evaluates offers corresponding to their request with ID id_b by executing the Evaluate function in ***B*uyerP** in Fig. 7.3. The function first ensures that the id matches the buyer's previously generated id_b, and the corresponding $BuyRequest$ has not expired i.e., the current time $T \leq T_{exp}$. Subsequently, (pk^s_{DH}, ct) is added to the local list SOffrs. After expiration, for each j-th offer in SOffrs, the function evaluates the symmetric encryption key $sk^b_{s_j}$ using the corresponding seller's public DH key $pk^{s_j}_{DH}$ and the buyer's DH secret key sk^b_{DH}. This key is then utilized to decrypt the corresponding ct in each offer. All decrypted offers are then processed through $\mathsf{Eval}(.)$ function, which outputs the best offer $(N, \$val, pk^s_{PRF}, slot, T_{com})$ based on buyer preferences. These preferences may include factors such as proximity to the seller, or the charging time deduced from $slot$, or the most convenient time slot. The $\mathsf{Eval}(.)$ function ensures that the buyer dynamically selects the optimal offer that matches their need.

Upon selecting the best offer, the buyer commits to such an offer by executing the CommitSale function in ***B*uyerP** in Fig. 7.3. The CommitSale function searches the buyer's wallet for a $coin$ matching the offer's $\$val$. If no matching coin is found, the buyer has the option to mint a new coin with a value of $\$val$. Subsequently, the function retrieves the branch b_{coin} of the selected $coin$ from the on-chain Merkle tree MT. The chosen $coin$ is then removed from the buyer's wallet. A PRF public key $pk^b_{PRF} = \mathsf{PRF}\{sk^b_{PRF}, id_b\}$ is generated and a random number a is chosen from the set $\{0,1\}^\lambda$, and the sale commitment cm is computed as $cm = \mathsf{Comm}(a, id_b, N, \$val)$. A hash chain of length $N+1$ is generated: $h_N \rightarrow h_{N-1} \rightarrow \ldots \rightarrow h_1 \rightarrow h_0$, where $h_i = \mathsf{H}(h_{i+1})$ and h_0 is the payment promise.

A zero-knowledge proof π_{com} is computed for the relation $\mathbb{R}_{com} \subseteq \mathcal{L}_{com}$, This proof serves to informally demonstrate the ownership of a coin of a specific value of $\$val$ and commits such a value for a particular energy transaction identified by id_b. Its formal definition is as follows:

$$
\begin{aligned}
((MT.root, sn, cm), (coin, b_{coin}, s, \$val, id_b, a)) \in \mathbb{R}_{com} \Leftrightarrow \\
coin = \mathsf{Comm}(s, sn||\$val) \\
\wedge\ MerkleBranch(MT.root, b_{coin}, coin) \\
\wedge\ cm = \mathsf{Comm}(a, id_b, N, \$val)
\end{aligned}
$$

Finally, the function sends the commitment to be registered with ***V*2VFx BC** as (CommitSale, h_0, pk^b_{PRF}, pk^s_{PRF}, sn, cm, T_{com}, π_{com}) anonymously. The CommitSale function in the on-chain code ***V*2VFx BC**, upon receiving the request (CommitSale, h_0, pk^b_{PRF}, pk^s_{PRF}, sn, cm, T_{com}, π_{com}), checks that the current time T is less than T_{com}. It verifies that the zero-knowledge proof π_{com} is valid with respect to the relation $\mathbb{R}_{com} \subseteq \mathcal{L}commit$. Fol-

lowing these checks, it adds sn to the SpentCoins list, flips a coin r to determine whether payment or service occurs first randomly, and adds $(h_0, r, pk^b_{PRF}, pk^s_{PRF}, cm, T_{com})$ to the ComCoins list.

Note: The use of r in the coin flip is essential to randomly determine whether payment or service occurs first. This approach ensures fairness, as the party who pays or serves first is at a disadvantage compared to the other party. This prevents either party from consistently gaining an advantage over the other. Additionally, it is worth noting that since pk^s_{PRF} and the sale parameters $(id_b, N, \$val)$ are encrypted in the Request\Response phase, and the sale parameters $(id_b, N, \$val)$ are concealed in a perfectly hiding commitment cm in the offer commitment phase, it is infeasible to link transactions in the Request\Response phase(s) with subsequent transactions.

Off-chain exchange phase In this phase, the seller has already accepted the buyer's offer for the designated time slot, i.e., $(id_b, N, \$val, slot)$ corresponding to pk^s_{PRF}. As depicted in Fig. 7.6, when the buyer reaches the seller's location[2], ***B*uyerP** is employed to transmit a to ***S*ellerP**, where the latter ensures that $cm = \mathsf{Comm}(a, id_b||N||\$val)$ and validates that the current time T falls within the specified $slot$. Note that a serves as an authentication token, thus ensuring that the buyer who presents it, is the one who committed to the sale. Once this information is verified, the exchange session can commence. Based on the value of r, either one unit of payment or charge is transferred first. Starting from $i = 1$, the seller receives one h_i for each unit charge transfer and ensures that $\mathsf{H}(h_i) = h_{i-1}$ before transferring the following unit charge. Note that h_0 is committed in the offer on the blockchain. At the conclusion of the exchange session, the seller acquires h_l such that l has a maximum value of N.

[2]In the scenario where the smart grid aims to obtain excess energy from Electric Vehicles (EVs), the selling EV is the one that travels to the SG preagreed-on location.

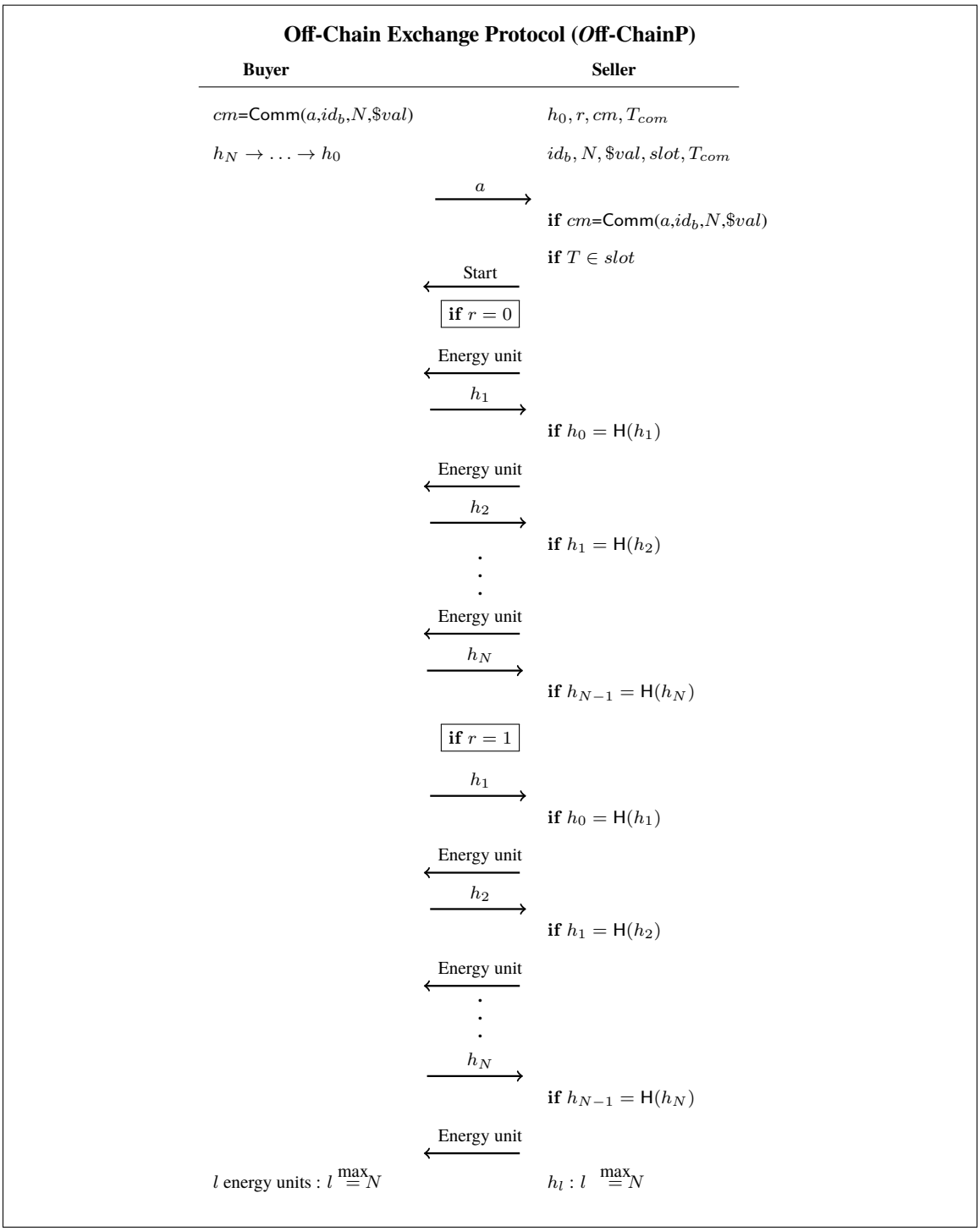

Figure 7.6: Off-chain energy exchange protocol.

Seller payment phase In this phase, the seller claims their payment from ***V*2VFx BC** using h_l which is collected in the off-chain exchange. The seller invokes the SellerPayment function in ***SellerP*** in Fig. 7.5. Upon providing (h_0, pk^s_{PRF}, cm, T_{com}, h_l, l, sk^s_{PRF}, a, id_b, N, $\$val$) as input, the function ensures that the commitment has not expired by checking that the current time T is less than T_{com}. It then selects two random numbers (s', sn') and calculates $coin'$ as $\mathsf{Comm}(s', sn'||\$val \times \frac{l}{N})$. Subsequently, the function computes a zero-knowledge proof π_{pay} for the relation $\mathbb{R}_{pay} \subseteq \mathcal{L}pay$, which informally establishes that the newly generated coin $coin'$ holds a value of $\frac{l}{N}$ of the previously committed $\$val$ by the buyer in cm. This zero-knowledge proof is formally defined as:

$$((pk^s_{PRF}, cm, coin', l), (sk^s_{PRF}, s', sn', a, id_b, N, \$val)) \in \mathbb{R}_{pay} \Leftrightarrow$$

$$pk^s_{PRF} = \mathsf{PRF}\{sk^s_{PRF}, id_b\}$$

$$\wedge cm = \mathsf{Comm}(a, id_b||N||\$val)$$

$$\wedge coin' = \mathsf{Comm}(s', sn'||\$val \times \frac{l}{N})$$

The function then adds $\{s', sn', \$val \times \frac{l}{N}, coin'\}$ to the seller's local Wallet. Finally, it sends the payment request to ***V*2VFx BC** as $(\mathsf{SellerPayment}, h_0, h_l, l, coin', \pi_{pay})$ anonymously. Upon receiving the request, The SellerPayment function in the on-chain code ***V*2VFx BC** ensures that there is an entry $X = \{h_0, r, pk^b_{PRF}, pk^s_{PRF}, cm, T_{com}, l\}$ in ComCoins for h_0, Verifies that the commitment has not expired, ensures that $X.l$ is empty, indicating that the payment has not been processed before, and finally, Checks that $h_{i-1} = \mathsf{H}(h_i)$ for $i = (l, \ldots, 0)$. If all checks pass, it verifies that the zero-knowledge proof π_{pay} is valid with respect to the relation $\mathbb{R}_{pay} \subseteq \mathcal{L}_{pay}$. Following these checks, it updates the entry $X.l$ in

ComCoins with l and adds $coin'$ to Coins.

Buyer refund phase

In this phase, the buyer claims the remaining amount of the initially committed coin if any from ***V*2VFx BC**. The Buyer invokes the BuyerRefund function in ***B*uyerP**. Upon providing $(h_0, pk^b_{PRF}, cm, T_{com}, l, sk^b_{PRF}, a, id_b, N, \$val)$ as input, where l is set to 0 if the off-chain exchange phase has not yielded any energy units, the function ensures that the commitment has expired by checking that the current time T is greater than T_{com}. It then selects two random numbers (s'', sn'') and calculates $coin''$ as $\mathsf{Comm}(s'', sn''||\$val \times (1 - \frac{l}{N}))$. Subsequently, the function computes a zero-knowledge proof π_{ref} for the relation $\mathbb{R}_{ref} \subseteq \mathcal{L}ref$, which informally proves that the generated coin $coin''$ holds a value of $1 - \frac{l}{N}$ of the previously committed $\$val$ by the buyer in cm. This zero-knowledge proof is formally defined as:

$$
\begin{aligned}
((Pk^B_{PRF}, cm, coin'', l), (sk^b_{PRF}, s'', sn'', a, id_b, N, \$val)) \in \mathbb{R}_{ref} \Leftrightarrow \\
pk^b_{PRF} = \mathsf{PRF}\{sk^b_{PRF}, id_b\} \\
\wedge cm = \mathsf{Comm}(a, id_b||N||\$val) \\
\wedge coin'' = \mathsf{Comm}(s'', sn''||\$val \times (1 - \frac{l}{N}))
\end{aligned}
$$

The function then adds $\{s'', sn'', \$val \times (1 - \frac{l}{N}), coin''\}$ to the buyer's local Wallet. Finally, it sends the refund request anonymously to ***V*2VFx BC** as $(\mathsf{BuyerRefund}, h_0, coin'', \pi_{ref})$. Upon receiving the request, the BuyerRefund function ***V*2VFx BC** ensures that there is an entry $X = \{h_0, r, pk^b_{PRF}, pk^s_{PRF}, cm, T_{com}, l\}$ in ComCoins for h_0, verifies that the commitment has expired, i.e., $T > T_{com}$, if l is empty, set $l = 0$ which indicates that T_{com}

has expired and the seller did not claim any payment. It then verifies that the π_{ref} is valid with respect to the relation $\mathbb{R}_{ref} \subseteq \mathcal{L}_{ref}$. Following these checks, it adds $coin''$ to Coins.

While it is possible to conceal h_l and l in π_{pay} and π_{ref}, this would significantly increase the computation cost due to the high zk-knowledge complexity of the hash function. Consequently, we choose to disclose l and h_l specifically in the seller payment and buyer refund phases. It is crucial to emphasize that the sale value $\$val$ remains concealed throughout all protocol transactions. Moreover, it is infeasible to establish links between transactions in the request for offers and offer response phases and subsequent transactions. Additionally, the spending of $coin'$ and $coin''$ requires a zero-knowledge operation similar to $\mathbb{R}_{com}$. This renders it infeasible to link seller payment and buyer refund transactions to any preceding or subsequent transactions. Note that we omit the details of transferring ownership of a private pool coin from P to another pseudonym $\tilde{P}$ or redepositing a coin value into P's public ledger. These operations align with zk-knowledge proof of the pour operation described in [67].

7.3 V2VFx Security

For our security analysis, we utilize the simulator wrapper for blockchains as presented in [67]. Such a wrapper is valuable in constructing the ideal-world simulator in our security proofs. The simulator wrapper modularizes the simulator construction by extracting the common aspects of the simulation that apply to all protocols within the blockchain execution model.

In this section, we demonstrate the security of the V2VFx protocol under the UC framework [16]. Specifically, we prove that the contract-based V2VFx protocol securely emulates

the ideal exchange functionality presented in Fig. 7.2. First, we present the following theorem.

Theorem 12. *Given a Simulation-Sound Extractable (SSE) zero-knowledge non-interactive zero-knowledge proof system, a correct and secure Diffie-Hillman key exchange protocol, A semantically secure encryption scheme, a perfectly hiding commitment scheme, and a secure PRF scheme, then V2VFx securely emulates the ideal functionality $\mathcal{F}_{V2VFx}$ in figure 7.2.*

Proof. We aim to establish the security and correctness of V2VFx protocol as an implementation of the ideal functionality $\mathcal{F}_{V2VFx}$ (see Figure 7.2). To achieve this, we present a security proof by constructing an ideal-world simulator $\mathcal{S}$ against any real-world adversary $\mathcal{A}$. The goal is to demonstrate that no polynomial-time environment $\mathcal{E}$ can distinguish between the real and ideal worlds. We begin by detailing the construction of the simulator $\mathcal{S}$ and subsequently provide a justification for the indistinguishability of the real and ideal worlds.

Ideal world simulator $\mathcal{S}$ According to Canetti [38], it suffices to construct a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$ that simply passes messages to and from the environment $\mathcal{E}$ that controls the execution of the ideal functionality $\mathcal{F}_{V2VFx}$. The ideal-world simulator $\mathcal{S}$ interacts with the $\mathcal{F}_{V2VFx}$ ideal functionality and simulates a $V2VFxBC$ instance internally. Below we construct the user-defined portion of our simulator. Our ideal adversary $\mathcal{S}$ can be obtained by applying the simulator wrapper we borrow from [67].

The simulator wrapper executes the standard setup procedure but preserves the trapdoor information utilized in generating the CRS for the NIZK proof system. This capability enables the simulator to fabricate proofs for false statements and extract witnesses from valid proofs. In the real-world scenario, where the adversary has access to the entire contract

state, the simulator facilitates the environment's visibility into the complete state of the local contract instance.

The environment $\mathcal{E}$ can issue instructions to parties in the ideal world, which are then directly conveyed to $\mathcal{F}_{V2VFx}$. This action triggers our simulator $\mathcal{S}$; however, the simulator gains access to only limited information regarding the instructions. Specifically, $\mathcal{S}$ only acquires knowledge of the values that would be disclosed to the real-world adversary. To emulate the ideal functionality in Fig. 7.2, our simulator $\mathcal{S}$ is constructed as follows:

- *Init*. $\mathcal{S}$ runs $(crs, tr) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$ and provides crs to the environment $\mathcal{E}$.

- *MintCoin*. $\mathcal{E}$ issues a MintCoin instruction to party P. Subsequently, $\mathcal{S}$ receives (MintCoin, P, $\$val_1$, ..., $\$val_n$) from the ideal functionality $\mathcal{F}_{V2VFx}$. $\mathcal{S}$ then generates $(s_i, sn_i) \leftarrow \{0,1\}^\lambda$ and $coin_i = \mathsf{Comm}(s_i, sn_i || \$val_i)$. Now, equipped with sufficient information, $\mathcal{S}$ can submit a valid MintCoin transaction to the contract.

- *BuyRequest*. $\mathcal{E}$ issues a BuyRequest instruction to party P. Subsequently, the $\mathcal{S}$ receives $(\mathsf{BuyRequest}, id_b, e, \mathcal{R}, T_{exp})$ from the ideal functionality $\mathcal{F}_{V2VFx}$. Then $\mathcal{S}$ generates $(sk^b_{DH}, pk^b_{DH}) \leftarrow \mathsf{DH.KeyGen}(1^\lambda)$; now, $\mathcal{S}$ can submit a valid BuyRequest transaction to the contract.

- *SellResponse*. $\mathcal{E}$ issues a SellResponse instruction to party P. Subsequently, $\mathcal{S}$ receives $(\mathsf{SellResponse}, id_b, T_{com})$ from the ideal functionality $\mathcal{F}_{V2VFx}$. $\mathcal{S}$ then generates $sk^s_{PRF} \leftarrow \{0,1\}^\lambda$, $pk^s_{PRF} = \mathsf{PRF}\{sk^s_{PRF}, id_b\}$, and $(sk^s_{DH}, pk^s_{DH}) \leftarrow \mathsf{DH.KeyGen}(1^\lambda)$. It then generates a dummy offer $(N, \$val, slot, T_{com})$. Now, $\mathcal{S}$ can submit a valid SellResponse transaction to the contract.

- *CommitSale*. $\mathcal{E}$ issues a CommitSale instruction to party P. Subsequently, $\mathcal{S}$ receives $(\mathsf{CommitSale}, \phi_{pay}, T_{com})$ from the ideal functionality $\mathcal{F}_{V2VFx}$. $\mathcal{S}$ does not learn the identity of the sender P, or the correct values of $(id_b, N, \$val, coin, s, sn, pk^b_{PRF}, pk^s_{PRF})$. It

randomly chooses arbitrary values for these parameters and selects random elements cm, pk^b_{PRF}, and pk^s_{PRF} from the co-domains of Comm and PRF, respectively. Using its knowledge of tr, $\mathcal{S}$ creates a false proof for the relation $\mathbb{R}_{com}$ and can submit a valid CommitSale transaction to the contract.

- *SellerPayment.* $\mathcal{E}$ issues a SellerPayment instruction to party P. Subsequently, $\mathcal{S}$ receives $(\mathsf{SellerPayment}, \phi_{pay}, h_l, l)$ from $\mathcal{F}_{V2VFx}$. $\mathcal{S}$ uses ϕ_{pay} (or h_0) to exhaustively search all previous transactions to obtain the corresponding (pk^s_{PRF}, cm); if it could not locate a previous transaction, it selects random elements (pk^s_{PRF}, cm) from the co-domains of PRF and Comm, respectively. Then, It randomly chooses arbitrary values for (sk^s_{PRF}, s', sn', a, id_b, N, $\$val$), creates $coin' = \mathsf{Comm}(s', sn'||\$val \times \frac{l}{N})$ and a false proof for the relation $\mathbb{R}_{pay}$ using its knowledge of tr. Now, $\mathcal{S}$ can submit a valid SellerPayment transaction to the contract.

- *Refund.* $\mathcal{E}$ issues a Refund instruction to party P and $\mathcal{S}$ receives $(\mathsf{Refund}, \phi_{pay})$ from $\mathcal{F}_{V2VFx}$. $\mathcal{S}$ uses ϕ_{pay} (or h_0) to exhaustively search all previous transactions to obtain the corresponding (pk^b_{PRF}, cm). It randomly chooses values for $(sk^b_{PRF}, s'', sn'', a, id_b, N, \$val)$, creates $coin'' = \mathsf{Comm}(s'', sn''||\$val \times \frac{l}{N})$ and a false proof for the relation $\mathbb{R}_{ref}$ using its knowledge of tr. Now, $\mathcal{S}$ can submit a valid Refund transaction to the contract.

Indistinguishability of real and ideal Worlds Recall that the environment $\mathcal{E}$ serves as a distinguisher between the ideal and real worlds. To prove that $\mathcal{E}$ cannot distinguish between the ideal-world transactions conducted through the ideal functionality $\mathcal{F}_{V2VFx}$ and the real-world transactions executed with the help of our simulator $\mathcal{S}$, we proceed through a series of indistinguishability games where the adversary is denoted by our simulator $\mathcal{S}$.

- **Real world**: In the real-world scenario, $\mathcal{S}$ simply acts as a dummy intermediary, passing messages between parties.

- **Game 1**: This game is identical to the real world, except that $\mathcal{S}$ simulates the setup of the underlying NIZK system as $(\text{crs}, \text{tr}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$. $\mathcal{S}$ then passes crs to the environment $\mathcal{E}$. Additionally, $\mathcal{S}$ internally simulates interactions with the contract *V2VFx BC*), replacing any NIZK proofs with simulated ones based on tr. This exploits the property that possessing tr allows for the generation of verifiable NIZK proofs for any statement. Given a simulatable-sound NIZK proof system, no polynomial-time environment $\mathcal{E}$ can distinguish Game 1 from the real world except with negligible probability.

- **Game 2**: Similar to Game 1, but when a party P sends a message to the contract*V2VFx BC*, $\mathcal{S}$ replaces P's signature on the message using freshly generated pseudonyms. Since $\mathcal{S}$ possesses the secret keys for these pseudonyms, the generated signatures are verifiable. Therefor, from the perspective of $\mathcal{E}$, Game 2 is indistinguishable from Game 1.

- **Game 3**: Similar to Game 2, but when a party P submits a ciphertext ct to the contract, $\mathcal{S}$ generates a fresh Diffie-Hellman key and replaces ct with the encryption of 0 using the freshly generated Diffie-Hellman shared secret key with the recipient. Given a correct and secure Diffie-Hellman key exchange protocol and a semantically secure encryption scheme, no polynomial-time $\mathcal{E}$ can distinguish Game 3 from Game 2 except with negligible probability.

- **Game 4**: Similar to Game 3, but when a party P submits a commitment cm to the contract, $\mathcal{S}$ replaces it with a commitment of 0. Given a perfectly hiding commitment scheme, no polynomial-time $\mathcal{E}$ can distinguish Game 4 from Game 3 except with negligible probability.

- **Game 5**: Similar to Game 4, but when a party P submits a PRF public key PK^B_{PRF} or PK^S_{PRF}, $\mathcal{S}$ replaces it with a randomly chosen value from the co-domain of the PRF.

Given a secure PRF function, no polynomial-time $\mathcal{E}$ can distinguish Game 5 from Game 4 except with negligible probability.

Therefore, game 5 is computationally indistinguishable from an ideal simulation to any polynomial-time environment $\mathcal{E}$.

7.4 Prototyping and Performance Evaluation

In this section, we provide an overview of our implementation of V2VFx and present comprehensive experimental results to demonstrate its performance.

Prototyping We developed a prototype to evaluate the performance and efficiency of the V2VFx protocol on Ethereum. The implementation confirms the feasibility and practicality of deploying V2VFx on top of Ethereum, utilizing zkSNARK as the underlying zero-knowledge proof system to realize the relations $\mathbb{R}_{coin}$, $\mathbb{R}_{com}$, $\mathbb{R}_{pay}$ and $\mathbb{R}_{ref}$ in section 7.2. For clarity, we reference each of these relations by the corresponding function name depicted in Fig. 7.4. Our prototype comprises of (i) a *Smart contract* which implements the contract depicted in Fig. 7.4 using Solidity on the Ethereum Virtual Machine, and (ii) *Client-side codes*. which implement the buyer and seller protocols depicted in Fig. 7.3 and Fig. 7.5, respectively, using Javascript. In what follows, we highlight the details of the key components of our implementation.

zkSNARK. We utilize the circom compiler [15] [3], an open-source tool, to construct and compile all the zkSNARK circuits required for V2VFx relations. Circom is a specialized domain-specific language designed to define arithmetic circuits that generate zkSNARKs. For keys and proofs generation, as well as generating verification code of zkSNARK circuits,

[3] https://docs.circom.io/

we employ Snarkjs[4], a JavaScript and Pure Web Assembly implementation of zkSNARK. In our implementation, we store the generated verification keys on the blockchain during the setup phase. Meanwhile, the generated proving keys are stored on the corresponding client-side code for later use in generating zkSNARK proofs.

Merkle tree. Coin ownership is proven in zero-knowledge by providing proof of membership of a coin in the on-chain Merkle tree generated by the contract during the execution of the MintCoin function. We utilize an incremental Merkle tree for this purpose [93], where all the tree leaves are pre-initialized with a "hash of zero", and intermediate nodes are pre-computed accordingly. The contract keeps track of the first "hash of zero" leaf in the Merkle tree and updates it with the coin commitment whenever a coin is minted. This update triggers the recomputation of the Merkle tree root by recomputing only the affected nodes along the path from the updated leaf to the root. Furthermore, within the MintCoin function, the coin commitment $coin_i$ is realized as the Poseidon hashing of $(s_i, sn_i||\$val_i)$ to circumvent unnecessary hashing by the contract. Additionally, to accommodate the fixed height requirement of the Merkle tree in zkSNARK circuit definition, we implemented a technique allowing the contract to deploy multiple Merkle trees of relatively small height (5 in this prototype) identified by a "*TreeID*". During the execution of the CommitSale function, the *TreeID* of the Merkle tree containing the coin to be proved for membership is provided. This technique strikes a balance between security and computation efficiency. In our implementation, the height of the Merkle tree can be adjusted either as a system-wide parameter or dynamically during the protocol execution.

zkSNARK-friendly hash function. Traditional popular hash functions like SHA256 are unsuitable for zkSNARK operations on finite fields due to the overhead in proving and

[4]https://github.com/iden3/snarkjs

verifying a large number of bitwise operations. ZKSNARK-friendly hash functions are optimized for modular arithmetic in a finite field, significantly reducing circuit complexity and enhancing overall circuit performance [61]. Thus, we employed the Poseidon [56] zkSNARK-friendly hash function in V2VFx's Merkle tree implementation. To illustrate the significant reduction in zkSNARK circuit size achieved by using zkSNARK-friendly hash functions, consider that a zkSNARK circuit to prove the knowledge of the preimage of SHA256 hash requires around 30k quadratic constraints, whereas implementing the same circuit using Poseidon requires only 200-300 quadratic constraints.

Micropayment. This mechanism requires careful consideration of settlement costs within smart contracts. In this context, we selected the more economical Keccak-256 as the one-way hash function for constructing hash chains. However, as the hash chain length increases, miners encounter a significant computational burden in verifying the final hash value due to the need to compute numerous hashes. To address this challenge, an advanced optimization technique could be used to reduce verification cost as outlined in [95] and originally proposed in [94] to mitigate computation costs. Specifically, consumers generate a signature over the payment amount, enabling miners to verify the signature rather than calculating hundreds or thousands of hashes, thereby streamlining the verification process. We omit micropayment measurement and analysis, we refer the reader to [95] for further details.

Performance evaluation We assess the performance of the V2VFx prototype in terms of (i) zkSNARK performance, and (ii) smart contract transaction gas used and execution time. Our experiments are conducted on an Amazon AWS m5.2xlarge machine equipped with 8 vCPUs and 32GB of RAM, running Ubuntu 20.04 LTS.

zkSNARK Performance We analyze the circuit size in terms of the required number of quadratic constraints and the size of proving/verification keys for each of the zkSNARK

circuits for the relations utilized in V2VFx. As illustrated in Table 7.1, the key sizes of all implemented circuits remain relatively small, primarily attributable to the compact circuit size facilitated by the zkSNARK-friendly hash function and our approach of employing multiple Merkle trees with smaller heights.

Circuit (Function)	Circuit Size (QC)	Proving Key Size (MB)	Verification Key Size (KB)
MintCoin	265	4.7	3.0
CommitSale	1785	5.7	3.2
SellerPayment	802	5.2	3.4
BuyerRefund	802	5.2	3.4

QC: Quadratic Constrains

Table 7.1: Circuit and key sizes for the different zkSNARK implementation used in V2VFx

We conduct tests to measure the execution time of setup, proof generation, and verification for each of the zkSNARK circuits employed in V2VFx. The results are depicted in Fig.7.7, with the time measured in seconds. Consistent with the reported circuit sizes, the time consumed by setup, proof generation, and verification for all zkSNARK circuits remains relatively minimal, primarily due to the compact circuit size facilitated by the zkSNARK-friendly hash function. The setup time is deemed acceptable as it is executed only once during contract initialization. Additionally, it is worth noting that the proof generation time and verification time exhibit similar trends across all circuits, aligning with results reported in [46] for benchmarking zkSNARK implementations. Furthermore, the measurements for the SellerPayment and BuyerRefund circuits are identical, as they exhibit the same level of complexity.

Smart Contract Measurements In this test, we utilize the Hardhat framework[5], a development environment for Ethereum software, to deploy the V2VFx smart contract on a local Ethereum network to measure transaction gas usage and execution time.

[5]https://hardhat.org

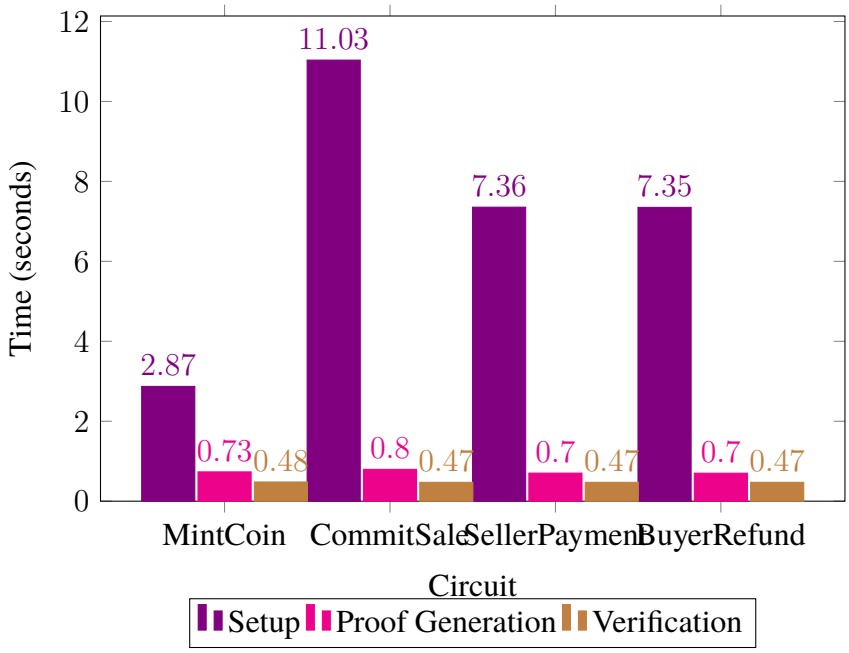

Figure 7.7: Execution times for zkSNARK circuits.

We report the cost of deploying our contract in Ethereum gas units, along with the time and gas consumption of the transactions in our implementation of the functions in Figure 7.4. The gas measurements are reported in Table 7.2.

Transaction	Time (ms)	Gas Used
Contract Deployment	-	50313
BuyRequest	6	163369
SellResponse	4	46665
MintCoin	26	241383
CommitSale	106	497721
SellerPayment	60	383574
BuyerRefund	23	267926

Table 7.2: Gas consumption and execution times of different transactions in V2VFx

Note that since contract deployment is a a trivial operation that is only done once, reporting its time is not relevant to the system's performance. We observe that while the required gas and execution times for BuyRequest, and SellResponse transactions are at

very acceptable levels, our measurements indicate higher gas requirements and hence longer execution times for the MintCoin, CommitSale, SellerPayment, and BuyerRefund functions. This is primarily because each of these functions verifies a zero-knowledge proof that relies on the Poseidon hash function. This becomes particularly evident in the case of the CommitSale function, where the contract has to perform multiple Poseidon hashing operations to verify the coin membership in the Merkle tree. The main reason for the higher gas requirement and execution times is that we implement the Poseidon hash as a stand-alone contract, which significantly impacts the gas requirements to execute such functions. However, in the future, once the Poseidon hash is implemented as a precompiled contract in the Ethereum protocol[6], this will substantially reduce the gas requirements of these functions. Moreover, we notice that the gas requirements and the execution time for these functions align with the zero-knowledge circuit complexity reported in Table 7.1. However, we report a higher gas requirement and a longer execution time for the SellerPayment function compared to the BuyerRefund function, although both functions' zero-knowledge circuits are almost the same. This discrepancy arises because the SellerPayment function also verifies the Keccak256 micropayments chain.

7.5 Comparing V2VFx to Related Works

Wan *et al.* [95] have proposed a blockchain-based Vehicle-to-Grid (V2G) charging system with a privacy-preserving fair exchange scheme. They primarily focus on preserving identity and transaction amount privacy, as well as ensuring fairness. However, their system is designed such that the smart grid always sets the price and initiates the trading process, bypassing the energy price negotiation phase necessary between the smart grid and the energy

[6]https://eips.ethereum.org/EIPS/eip-5988

seller. This oversight could potentially compromise transaction privacy. Additionally, their scheme requires sharing secret information between the smart grid and the seller but lacks clarity on how such information is shared confidentially. Furthermore, the scheme neglects to address location privacy as a stand-alone security property of the system. Radi *et al.* [80] have introduced a blockchain-based peer-to-peer energy-trading ecosystem that incorporates an anonymous payment system to safeguard the privacy of individual owners and their specific charging locations. However, their system necessitates the involvement of an online trusted financial entity for operation. Moreover, it operates over a consortium blockchain managed jointly by energy traders and financial institutions to ensure transaction party privacy. While this approach aims to enhance privacy and authenticity, it imposes limitations on system adoptability and escalates setup overheads. Additionally, it is worth noting that the proposed system does not guarantee fairness between the transacting parties. Expanding on the previous system, Baza *et al.* [11] have introduced a novel blockchain-based Charging V2V system that eliminates the need for a consortium blockchain as seen in the prior attempt. However, the new system still requires the participation of an online off-chain trusted financial entity for its operation. Furthermore, despite the extensive use of cryptographic primitives, it does not ensure fairness between the transacting parties. The authors in [97] have proposed a blockchain privacy-preserving authentication scheme with anonymous payment for V2G networks. However, their approach relies on off-chain trusted third parties for registration and accountability. Nevertheless, their system overlooks fair exchange between the transacting parties. Liang *et al.* [72] have proposed a decentralized V2G system to facilitate efficient and robust energy-trading within campus EV networks. However, their scheme utilizes a complex structure involving multiple consortium blockchains to ensure trading security and data privacy between energy requests and offers. Furthermore, their

approach relies on off-chain trusted third parties and lacks a formal security review of the proposed system. Zhao *et al.* [99] introduced a secure intra-regional-inter-regional peer-to-peer electricity trading system, leveraging blockchain technology for transaction payments. However, a key limitation of their work is the absence of transaction data protection within their private Ethereum module. Kim *et al.* [64] have proposed a design for a distributed peer-to-peer auction-based energy-trading mechanism. However, the proposed system lacks consideration for transaction data privacy as one of its fundamental properties. Knirsch *et al.* [66] proposed an automated peer-to-peer dynamic tariff decision system, wherein electric vehicle owners select a charging station based on the supply-side offers they receive. However, the system solely focuses on preserving location and sale value privacy, overlooking identity or transaction data privacy. Additionally, their system emphasizes finding the optimal tariff without specifying a dedicated payment scheme. The authors suggest using membership cards or credit cards for payment, which could potentially undermine the overall privacy objectives of the system. Pustišek *et al.* [66] proposed a blockchain Vehicle-to-Charging station (V2C) negotiation system to select the most convenient electric vehicle charging station. The EVs send their planned routes and battery status to the blockchain, and then charging stations offer their prices so that the EVs select the best-offered price. However, their proposal outlines the architecture of a simple system for autonomous selection for electric vehicle charging stations and only focuses on the best price selection. Su *et al.* [91] have introduced a permissioned V2C energy blockchain system to implement secure charging services for EVs. The paper uses a Byzantine fault tolerance consensus algorithm to reach the consensus in the energy blockchain. The contract theory has also been used to satisfy the energy needed from EVs while maximizing the operatorâĂŹs profit.

Table 7.3 provides a comparison between V2VFx and other related protocols. In our

comparison, we consider whether the system provides an end-to-end solution that covers all aspects of the energy-trading session, such as price agreement, energy-trading, payment, and refund. We also evaluate whether the proposed solution uses a public blockchain as its core architecture, supports V2V energy trading, offers a dynamic-optimal mechanism for energy price agreement, features an anonymous payment mechanism, and requires an off-chain trusted third party (TTP). Additionally, we consider the security properties that V2VFx offers, including fair exchange, which protects the buyer and seller from premature aborts, ensuring the buyer only pays for the amount of energy they receive. We also examine privacy provisions, including obscuring the buyer and seller identities, concealing proposed sellers' offers from the public, maintaining the privacy of buyer and seller locations, contractual terms privacy ensuring that the agreed-upon energy sale value and number of payments are kept hidden from the public, and transaction unlinkability, ensuring that a payment sender (buyer) cannot be associated with the corresponding recipient (seller).

Table 7.3: Comparison with other related protocols.

Criteria	[95]	[80]	[11]	[97]	[72]	[99]	[64]	[66]	[79]	[91]	V2VFx
end-to-end Trading	×	✓	✓	✓	✓	✓	×	×	×	×	✓
public blockchain	✓	×	✓	×	×	×	×	✓	✓	×	✓
vehicle-to-vehicle(V2V)	×	×	✓	×	×	×	×	×	×	×	✓
dynamic-optimal sale price	×	✓	✓	×	×	×	✓	✓	✓	✓	✓
anonymous payment	✓	✓	✓	✓	×	×	×	×	×	×	✓
no off-chain TTP	✓	×	×	×	×	✓	✓	✓	✓	✓	✓
fair exchange	✓	×	×	×	×	×	×	×	×	×	✓
identity privacy	✓	✓	✓	✓	✓	✓	×	×	×	×	✓
offer privacy	×	×	✓	×	×	×	×	✓	×	×	✓
location privacy	×	✓	✓	×	×	×	×	✓	×	×	✓
contractual terms privacy	℗	×	✓	×	×	×	×	℗	×	×	✓
transaction unlinkability	✓	✓	✓	✓	×	×	×	×	×	×	✓

✓ denotes a realized feature, × denotes an unrealized feature, and ℗ denotes a partially realized feature.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation presented novel advancements in the field of privacy-preserving protocols, focusing on two core areas: Policy-Based Sanitizable Signature Schemes and privacy-preserving Fair Exchange Mechanisms. Through the development of the UP3S, TPBS, and EP3S frameworks, the dissertation addressed critical limitations in existing policy-based sanitizable signature schemes, notably introducing unlinkability, enhanced fine-grained modification control, and the delegation of sanitization rights with robust accountability guarantees.

Furthermore, the V2VFx framework proposed a new model for privacy-preserving, fair exchange, and decentralized vehicle-to-vehicle energy trading. By combining blockchain technologies with advanced cryptographic primitives, V2VFx ensures fair exchange without the need for trusted third parties, while simultaneously protecting sensitive user and transactional data.

The research presented in this dissertation advances the state-of-the-art in privacy-preserving

protocols by offering practical, scalable, and formally secure solutions for real-world applications that demand a high degree of both flexibility and privacy. Looking ahead, future work can further investigate the integration of advanced policy-based sanitizable signature schemes within decentralized fair exchange systems, leveraging the fine-grained control and unlinkability properties to enable more adaptive, privacy-respecting transaction frameworks.

8.2 Future Work

While the protocols introduced in this dissertation significantly improve privacy, accountability, and fairness, several promising directions for future research remain:

- **Revocation in EP3S Schemes**: Future work should focus on extending EP3S to include efficient revocation mechanisms. This will enhance control and adaptability, allowing signers to revoke sanitization rights or modification policies dynamically, which is critical for real-world deployments where policy requirements and trust relationships evolve over time.

- **New Privacy-Preserving Fair Exchange Protocols with EP3S**: Building upon the foundations of EP3S, there is a compelling opportunity to design a novel privacy-preserving fair exchange protocol for digital assets. Such a protocol would leverage the unlinkability, traceability, and fine-grained control offered by EP3S to ensure both fairness and strong privacy guarantees in the exchange of digital goods, credentials, or cryptocurrencies.

- **Generalizing V2VFx to V2XFx**: A natural extension of the V2VFx framework is its expansion into a more generalized *Vehicle-to-Everything Fair Exchange (V2XFx)*

protocol. V2XFx would support broader energy and data trading scenarios, including vehicle-to-grid (V2G), vehicle-to-home (V2H), and vehicle-to-infrastructure (V2I) exchanges, thereby creating a unified platform for fair and privacy-preserving trading in increasingly interconnected smart environments.

Through these future directions, the research can continue to contribute to building more secure, privacy-aware, and user-centric systems in the evolving landscape of decentralized and digitally governed ecosystems.

Appendix A

A.1 Building Blocks Security Definitions

RDS schemes security

Unlinkability. This security notion requires that given access to oracles $\mathcal{O}\mathsf{Sign}(.)$ and $\mathcal{O}\mathsf{LoRRDS}(.)$ which are defined in Fig. A.1, the adversary $\mathcal{A}$ inputs two valid message signature pairs $(m_0, \sigma_{RDS,0})$ and $(m_1, \sigma_{RDS,1})$ to $\mathcal{O}\mathsf{LoRRDS}(.)$ oracle, the oracle is initialized with a secret random bit '$b \in \{0, 1\}$'. Depending on 'b', the oracle calls $\mathsf{RandomizeRDS}$ on either the left or right input message signature pair and outputs $\sigma'_{RDS,b}$. The adversary wins if it could determine which message signature pair is used in the rerandomization process with probability better than the random guess [100]. Note that RDS unlinkability implies that no adversary can distinguish between a freshly signed message signature pair and rerandomized version of the same message as with the case if the adversary obtains two different signatures for the same message m (since RDS schemes are probabilistic schemes) by querying $\mathcal{O}\mathsf{SignRDS}$ twice with the same message m, then inputs $(m, \sigma_{RDS,0})$ and $(m, \sigma_{RDS,1})$ to $\mathcal{O}\mathsf{LoRRDS}(.)$.

Note: According to [77] the unlinkability game of the underlying RDS scheme in Fig. A.2

can only be possible if the adversary does not explicitly know the RDS signed message, hence the adversary cannot link the Challenger output to the originating message using the RDS verification algorithm. However, since the adversary inputs two identical messages to $\mathcal{O}\mathsf{LoRRDS}(.)$, thus the aforementioned restriction does not apply.

$\mathcal{O}\mathsf{SignRDS}(m)$

$(m, \sigma_{RDS}) \leftarrow \mathsf{SignRDS}(sk_{RDS}, m)$
$\mathcal{M} = \mathcal{M} \cup \{m, \sigma_{RDS}\}$
return (m, σ_{RDS})

$\mathcal{O}\mathsf{LoRRDS}(m_0, \sigma_{RDS,0}, m_1, \sigma_{RDS,1})$

if $\mathsf{VerifyRDS}(pk_{RDS}, m_0, \sigma_{RDS,0}) \wedge \mathsf{VerifyRDS}(pk_{RDS}, m_1, \sigma_{RDS,1})$
  $(m_b, \sigma'_{RDS,b}) \leftarrow \mathsf{RandomizeRDS}(m_b, \sigma_{RDS,b})$
  return $(\sigma'_{RDS,b})$
return 0

Figure A.1: RDS security experiments oracles.

Definition 21. *(RDS Unlinkability) The RDS scheme is unlinkable if for any PPT adversary* $\mathcal{A}$, $|\Pr[\boldsymbol{Exp}^{Unlinkability}_{\mathcal{A},RDS}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$, *where the unlinkability experiment is defined in Fig. A.2.*

$\mathbf{Exp}^{Unlinkability}_{\mathcal{A},RDS}(\lambda)$

$pp_{RDS} \leftarrow \mathsf{ParGenRDS}(1^\lambda)$
$(pk_{RDS}, sk_{RDS}) \leftarrow \mathsf{KeyGenRDS}(pp_{RDS})$
$b \xleftarrow{\$} \{0, 1\}$
$a \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignRDS}(.), \mathcal{O}\mathsf{LoRRDS}(., b)}(pk_{RDS})$
if $a = b$
  return 1
return 0

Figure A.2: RDS unlinkability experiment.

Definition 22. *(RDS EUF-CMA) The RDS scheme is EUF-CMA secure if the for any PPT adversary* $\mathcal{A}$, $\Pr[\boldsymbol{Exp}^{EUF-CMA}_{\mathcal{A},RDS}(\lambda) = 1] \leq \epsilon(\lambda)$, *where the RDS EUF-CMA experiment is defined in Fig. A.3.*

$\mathbf{Exp}^{EUF-CMA}_{\mathcal{A},RDS}(\lambda)$

$\mathcal{M} = \{\}$
$pp_{RDS} \leftarrow \mathsf{ParGenRDS}(1^{\lambda})$
$(pk_{RDS}, sk_{RDS}) \leftarrow \mathsf{KeyGenRDS}(pp_{RDS})$
$(m^*, \sigma^*_{RDS}) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignRDS}(.)}(pk_{RDS})$
if $(m^*, \sigma^*_{RDS}) \notin \mathcal{M}$
  return $\mathsf{VerifyRDS}(pk_{RDS}, m^*, \sigma^*_{RDS})$
return 0

Figure A.3: RDS EUF-CMA experiment.

TABS schemes security

Unforgeability. This notion requires that an adversary cannot produce a verifiable signature σ_{TABS} for a message m under a predicate Υ such that $\Upsilon(\mathbb{S}) \neq 1$ where $\mathbb{S}$ is the set of attributes that the adversary holds. In other words, an adversary cannot generate a valid signature under a predicate where they do not possess the corresponding set of attributes that satisfy such a predicate [47]. The experiment, defined in Fig. A.5, models the unforgeability security notion in which the adversary is given access to the three oracles $\mathcal{O}\mathsf{KeyGenTABS}$, $\mathcal{O}\mathsf{SignTABS}$, and $\mathcal{O}\mathsf{ProveTABS}$ which are defined in Fig. A.4. The adversary wins if it could generate a verifiable signature $(m^*, \sigma^*_{TABS}, \Upsilon^*)$ such that $\Upsilon^*(\mathbb{S}_{Adv}) = 0$ for all the set of attributes $\mathbb{S}_{Adv}$ queried by the adversary to $\mathcal{O}\mathsf{KeyGenTABS}$ and the pair (m^*, Υ^*) have not been queried before to $\mathcal{O}\mathsf{SignTABS}$.

Definition 23. *(TABS Unforgeability) a TABS scheme is unforgeable if for any PPT adversary $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{Unforgeability}_{\mathcal{A},TABS}(\lambda) = 1] \leq \epsilon(\lambda)$, where the unforgeability experiment is defined in Fig. A.5.*

Privacy. Generally speaking, TABS privacy implies that the generated signature only attests to the fact that a set of attributes possessed by a signer satisfies a predicate while hiding the signer's identity and the set of attributes used to satisfy such a predicate. While

$\mathcal{O}\mathsf{KeyGenTABS}(i, \mathbb{S}_i)$

$\mathbb{S}_{Adv} = \mathbb{S}_{Adv} \cup \{i, \mathbb{S}_i\}$
$sk_{TABS}^{User,i} \leftarrow \mathsf{SignerKeyGenTABS}(pp_{TABS}, msk_{TABS}^{AA}, i, \mathbb{S}_i)$
return $sk_{TABS}^{User,i}$

$\mathcal{O}\mathsf{SignTABS}(m, \Upsilon)$

$\sigma_{TABS} \leftarrow \mathsf{SignTABS}(pp_{TABS}, sk_{TABS}^{User,i}, m, \Upsilon)$
$\mathcal{M} = \mathcal{M} \cup (m, \sigma_{TABS}, \Upsilon)$
return $(m, \sigma_{TABS}, \Upsilon)$

$\mathcal{O}\mathsf{ProveTABS}(m, \sigma_{TABS}, \Upsilon)$

if $(m, \sigma_{TABS}, \Upsilon) \in \mathcal{M}$
$(i, \pi) \leftarrow \mathsf{TraceTABS}(tsk_{TABS}^{TA}, m, \sigma_{TABS}, \Upsilon)$
return (i, π)
return 0

$\mathcal{O}\mathsf{LoRSignTABS}(m, \Upsilon)$

$\sigma_{TABS} \leftarrow \mathsf{SignTABS}(pp_{TABS}, sk_{TABS}^{User,b}, m, \Upsilon)$
return $(m, \sigma_{TABS}, \Upsilon)$

Figure A.4: TABS security experiments oracles.

$\mathbf{Exp}_{\mathcal{A},TABS}^{Unforgeability}(\lambda)$

$pp_{TABS} \leftarrow \mathsf{ppGenTABS}(1^{\lambda})$
$tsk_{TABS}^{TA} \leftarrow \mathsf{TAKeyGenTABS}(pp_{TABS})$
$(pk_{TABS}, msk_{TABS}^{AA}) \leftarrow \mathsf{AAKeyGenTABS}(pp_{TABS})$
$\mathcal{M} = \mathbb{S}_{Adv} = \{\}$
$(m^*, \sigma_{TABS}^*, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGenTABS}(.), \mathcal{O}\mathsf{SignTABS}(.), \mathcal{O}\mathsf{ProveTABS}(.)}(pk_{TABS})$
if $\mathsf{VerifyTABS}(pp_{TABS}, pk_{TABS}, m^*, \sigma_{TABS}^*, \Upsilon^*) \wedge (m^*, \sigma_{TABS}^*, \Upsilon^*) \notin \mathcal{M} \wedge$
$\forall \{\mathbb{S}'\} \in \mathbb{S}_{Adv}, \Upsilon^*(\mathbb{S}') = 0$
return 1
return 0

Figure A.5: TABS unforgeability experiment.

preserving the anonymity of the signer. Privacy also implies unlinkability, where an observer cannot distinguish if two valid signatures for the same signing policy have been computed by the same signer [74]. TABS privacy is modeled by an indistinguishability experiment that is defined in Fig. A.6, in which, the adversary has access to key generation oracle $\mathcal{O}\mathsf{KeyGenTABS}$, a signing oracle $\mathcal{O}\mathsf{SignTABS}$, and proving oracle $\mathcal{O}\mathsf{ProveTABS}$ where anonymity revocation is restricted to signatures generated by $\mathcal{O}\mathsf{SignTABS}$ only, see Fig. A.4. The adversary is challenged by $\mathcal{O}\mathsf{LoRSignTABS}$ oracle, which is initialized by two signing secret signing keys $sk_{TABS}^{User,0}$ and $sk_{TABS}^{User,1}$ of two different users identities, and a random bit

$b \in \{0,1\}$. Upon the input of a message m, $\mathcal{O}\mathsf{LoRSignTABS}$ outputs $(m, \sigma_{TABS}, \Upsilon)$ signed by $sk_{TABS}^{User,b}$ such that $\Upsilon(\mathbb{S}_{User,0}) = \Upsilon(\mathbb{S}_{User,1}) = 1$. The adversary wins if it can guess the bit b.

Definition 24. *(TABS Privacy) TABS scheme is private if for any PPT adversary $\mathcal{A}$,*

$$|\mathsf{Pr}[\boldsymbol{Exp}_{\mathcal{A},TABS}^{privacy}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$$

where the privacy experiment is defined in Fig. A.6.

$\mathbf{Exp}_{\mathcal{A},TABS}^{Privacy}(\lambda)$

$pp_{TABS} \leftarrow \mathsf{ppGenTABS}(1^\lambda)$
$tsk_{TABS}^{TA} \leftarrow \mathsf{TAKeyGenTABS}(pp_{TABS})$
$(pk_{TABS}, msk_{TABS}^{AA}) \leftarrow \mathsf{AAKeyGenTABS}(pp_{TABS})$
$sk_{TABS}^{User,0} \leftarrow \mathsf{SignerKeyGenTABS}(pp_{TABS}, msk_{TABS}^{AA}, i_0, \mathbb{S}_{User,0})$
$sk_{TABS}^{User,1} \leftarrow \mathsf{SignerKeyGenTABS}(pp_{TABS}, msk_{TABS}^{AA}, i_1, \mathbb{S}_{User,1})$
$\mathcal{M} = \{\}$
$b \xleftarrow{\$} \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGenTABS}(.),\mathcal{O}\mathsf{SignTABS}(.),\mathcal{O}\mathsf{LoRSignTABS}(.,b)}(pk_{TABS})$
if $a = b$
  return 1
return 0

Figure A.6: TABS privacy experiment.

Non-frameability. This property ensures that even if all authorities (AA and TA) and users in the scheme collude together dishonestly, they cannot produce a valid signature that is traced back to an honest user [53]. TABS non-frameability is modeled by the experiment defined in Fig. A.7, in which the adversary has access to both TA and AA secret keys $(tsk_{TABS}^{TA}, msk_{TABS}^{AA})$, in addition to $\mathcal{O}\mathsf{KeyGenTABS}$, $\mathcal{O}\mathsf{SignTABS}$, and $\mathcal{O}\mathsf{ProveTABS}$. The adversary wins if it outputs a verifiable $(m^*, \sigma_{TABS}^*, \Upsilon^*)$ under pk_{TABS} that has not been queried to $\mathcal{O}\mathsf{SignTABS}$ and when $(m^*, \sigma_{TABS}^*, \Upsilon^*)$ is traced back to its signer, the tracing algorithm outputs an identity that has never been queried to $\mathcal{O}\mathsf{KeyGenTABS}$. Additionally,

the output of the tracing algorithm is verifiable using the JudgeTABS algorithm.

$\mathbf{Exp}^{Non-frameability}_{\mathcal{A},TABS}(\lambda)$

$pp_{TABS} \leftarrow \mathsf{ppGenTABS}(1^{\lambda})$
$tsk^{TA}_{TABS} \leftarrow \mathsf{TAKeyGenTABS}(pp_{TABS})$
$(pk_{TABS}, msk^{AA}_{TABS}) \leftarrow \mathsf{AAKeyGenTABS}(pp_{TABS})$
$\mathcal{M} = \mathbb{S}_{Adv} = \{\}$
$(m^*, \sigma^*_{TABS}, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGenTABS}(.),\mathcal{O}\mathsf{SignTABS}(.),\mathcal{O}\mathsf{ProveTABS}(.)}(tsk^{TA}_{TABS}, pk_{TABS}, msk^{AA}_{TABS})$
if $\mathsf{VerifyTABS}(pp_{TABS}, pk_{TABS}, m^*, \sigma^*_{TABS}, \Upsilon^*)$
  $(i^*, \pi^*) \leftarrow \mathsf{TraceTABS}(tsk^{TA}_{TABS}, m^*, \sigma^*_{TABS}, \Upsilon^*)$
  if $\mathsf{JudgeTABS}(pp_{TABS}, pk_{TABS}, m^*, \sigma^*_{TABS}, \Upsilon^*, i^*, \pi^*) \wedge i^* \notin \mathbb{S}_{Adv} : \Upsilon^*(\mathbb{S}_i) = 1$
  $\wedge (m^*, \sigma^*_{TABS}, \Upsilon^*) \notin \mathcal{M}$
    return 1
return 0

Figure A.7: TABS non-frameability experiment.

Definition 25. *(TABS Non-frameability) a TABS scheme is non-frameable if for any PPT adversary $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{Non-frameability}_{\mathcal{A},TABS}(\lambda) = 1] \leq \epsilon(\lambda)$, where the non-frameability experiment is defined in Fig. A.7.*

Traceability. TABS traceability ensures that no efficient adversary can produce a signature that cannot be traced. TABS traceability is modeled by the experiment defined in Fig. A.8, in which the adversary has access to $\mathcal{O}$KeyGenTABS, $\mathcal{O}$SignTABS, and $\mathcal{O}$ProveTABS where identity revocation is restricted to signatures generated by $\mathcal{O}$SignTABS only. The Adversary wins if it outputs a verifiable $(m^*, \sigma^*_{TABS}, \Upsilon^*)$ under pk_{TABS}, (m^*, Υ^*) has been never queried to the signing oracle, and when $(m^*, \sigma^*_{TABS}, \Upsilon^*)$ is traced back, either the ProveTABS or JudgeTABS outputs $\perp$.

Definition 26. *(TABS Traceability) a TABS scheme is traceable if for any PPT adversary $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{Traceability}_{\mathcal{A},TABS}(\lambda) = 1] \leq \epsilon(\lambda)$, where the traceability experiment is defined in Fig. A.8.*

$\mathbf{Exp}^{Traceability}_{\mathcal{A},TABS}(\lambda)$

$pp_{TABS} \leftarrow \mathsf{ppGenTABS}(1^{\lambda})$
$tsk^{TA}_{TABS} \leftarrow \mathsf{TAKeyGenTABS}(pp_{TABS})$
$(pk_{TABS}, msk^{AA}_{TABS}) \leftarrow \mathsf{AAKeyGenTABS}(pp_{TABS})$
$\mathcal{M} = \mathbb{S}_{Adv} = \{\}$
$(m^*, \sigma^*_{TABS}, \Upsilon^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGenTABS}(.),\mathcal{O}\mathsf{SignTABS}(.),\mathcal{O}\mathsf{ProveTABS}(.)}(pk_{TABS})$
if $\mathsf{VerifyTABS}(pp_{TABS}, pk_{TABS}, m^*, \sigma^*_{TABS}, \Upsilon^*) \wedge (m^*, \sigma^*_{TABS}, \Upsilon^*) \notin \mathcal{M}$
  $(i^*, \pi^*) \leftarrow \mathsf{TraceTABS}(tsk^{TA}_{TABS}, m^*, \sigma^*_{TABS}, \Upsilon^*)$
    if $i^* = \perp \vee \mathsf{JudgeTABS}(pp_{TABS}, pk_{TABS}, m^*, \sigma^*_{TABS}, \Upsilon^*, i^*, \pi^*) = \perp$
      return 1
return 0

Figure A.8: TABS traceability experiment.

Digital signature schemes security

In what follows, we give the formal definition of Existential Unforgeability under Chosen Message Attack (EUF-CMA) of digital signature schemes that are required for proving the security of TPBS.

Existential Unforgeability under Chosen Message Attack (EUF-CMA). This security notion implies that given access to a signing oracle $\mathcal{O}\mathsf{SignSig}$ (see Fig. A.9), it is hard for an adversary $\mathcal{A}$ who does not have access to the signing keys to output a valid message signature pair (m^*, σ^*_{Sig}) for which m^* was never queried to the signing oracle.

$\mathcal{O}\mathsf{SignSig}(m)$
$\sigma_{Sig} \leftarrow \mathsf{SignSig}(sk_{Sig}, m)$
$\mathcal{M} = \mathcal{M} \cup \{m, \sigma_{Sig}\}$
return σ_{Sig}

Figure A.9: Digital signature security experiments oracle.

Definition 27. *(Digital signature scheme EUF-CMA) The digital signature scheme is EUF-CMA secure, if the for any PPT adversary $\mathcal{A}$, $\mathsf{Pr}[\boldsymbol{Exp}^{EUF-CMA}_{\mathcal{A},Sig}(\lambda) = 1] \leq \epsilon(\lambda)$, where the EUF-CMA experiment is defined in Fig. A.10.*

$\mathbf{Exp}^{EUF-CMA}_{\mathcal{A},Sig}(\lambda)$

$\mathcal{M} = \{\}$
$pp_{Sig} \leftarrow \mathsf{ppGenSig}(1^{\lambda})$
$(pk_{Sig}, sk_{Sig}) \leftarrow \mathsf{KeyGenSig}(pp_{Sig})$
$(m^*, \sigma^*_{Sig}) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SignSig}(.)}(pk_{Sig})$
if $(m^*, \sigma^*_{Sig}) \notin \mathcal{M}$
  return $\mathsf{VerifySig}(pk_{Sig}, m^*, \sigma^*_{Sig})$

Figure A.10: Digital signature scheme EUF-CMA experiment.

SE-NIZK schemes security

In what follows, we give the formal definitions of the security properties of SE-NIZK schemes that are required for proving the security of TPBS. **Zero knowledge**. This security notion implies that given access to a prove oracle $\mathcal{O}\mathsf{Sim\text{-}or\text{-}ProveNIZK}$ (see Fig. A.11), it is hard for adversary $\mathcal{A}$ to distinguish between a proof for a statement x using a witness w from a simulated one.

$\mathcal{O}\mathsf{Sim\text{-}or\text{-}ProveNIZK}(x, w)$
$\pi_{NIZK_0} \leftarrow \mathsf{ProveNIZK}(crs, x, w)$
if $\mathbb{R}(x, w) = 1$
  $\pi_{NIZK_1} \leftarrow \mathsf{SimProveNIZK}(crs, x, tr_{NIZK})$
  $else$ **return** $\perp$
return π_{NIZK_b}

$\mathcal{O}\mathsf{SimNIZK}(\mathsf{x})$
$\pi_{NIZK} \leftarrow \mathsf{SimProveNIZK}(crs, x, tr_{NIZK})$
$\mathcal{M} = \mathcal{M} \cup (x, \pi_{NIZK})$
return π_{NIZK}

Figure A.11: NIZK system security experiments oracles.

Definition 28. *(NIZK Zero-knowledge) The NIZK system is zero-knowledge if for any PPT adversary $\mathcal{A}$, $|\mathsf{Pr}[\boldsymbol{Exp}^{ZK}_{\mathcal{A},NIZK}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$, where the zero-knowledge experiment is defined in Fig. A.12.*

$Exp^{ZK}_{\mathcal{A},NIZK}(\lambda)$

$crs_0 \leftarrow \mathsf{SetupNIZK}(1^\lambda)$
$(crs_1, tr_{NIZK}) \leftarrow \mathsf{SimSetupNIZK}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{Sim\text{-}or\text{-}ProveNIZK}(.,b,\mathsf{tr_{NIZK}})}(crs_b)$
if $a = b$
  return 1
return 0

Figure A.12: NIZK system zero-knowledge experiment.

Simulation-extractability. This security notion implies that given access to a simulated prove oracle $\mathcal{O}\mathsf{SimNIZK}$ (see Fig. A.11), it is hard for adversary $\mathcal{A}$ to output a verifiable proof for a statement x using a witness w such that $\mathbb{R}(x, w) = 0$.

Definition 29. *(NIZK Simulation-extractability) The NIZK system is simulation-extractable if the for any PPT adversary $\mathcal{A}$, $\mathsf{Pr}[Exp^{Sim-Extr}_{\mathcal{A},NIZK}(\lambda) = 1] \leq \epsilon(\lambda)$, where the NIZK simulation-extractability experiment is defined in Fig. A.13.*

$Exp^{Sim-Extr}_{\mathcal{A},NIZK}(\lambda)$

$(crs, tr_{NIZK}) \leftarrow \mathsf{SimSetupNIZK}(1^\lambda)$
$\mathcal{M} = \{\}$
$(x^*, \pi^*_{NIZK}) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{SimNIZK}(.,\mathsf{tr_{NIZK}})}(crs)$
if $(x^*, \pi^*_{NIZK}) \notin \mathcal{M} \wedge \mathsf{VerifyNIZK}(crs, x^*, \pi^*_{NIZK})$
  $w \leftarrow Extr(tr_{NIZK}, x^*, \pi^*_{NIZK})$
  if $\mathbb{R}(x, w) = 0$
    return 1
return 0

Figure A.13: NIZK system simulation-extractability experiment.

ABE security

In what follows, we give the formal definitions of IND-CCA-2 security property of ABE schemes that are required for proving the security of EP3S. The IND-CCA-2 of ABE

is defined by the experiment in Fig. A.15 [17, 87]. An adversary $\mathcal{A}$ who has access to $\mathcal{O}$KeyGenABE, $\mathcal{O}$DecryptABE, and $\mathcal{O}$LoREncryptABE oracles (Fig. A.14). $\mathcal{O}$LoREncrypt is the challenge oracle which is initialized with a secret bit "b", the adversary inputs two distinct messages (m_0, m_1) such that $|m_0| = |m_1|$ to the challenge oracle to obtain the ciphertext C_b^* of either m_0 or m_1 under some access structure $\mathbb{A}^*$ predefined by the adversary. The adversary can obtain decryption keys for any attribute set of its choice, the only restriction is that it can not obtain decryption keys for an attribute set that satisfies $\mathbb{A}^*$. The adversary can adaptively query the $\mathcal{O}$DecryptABE oracle to decrypt any ciphertext C of its choice as long as $C \neq C_b^*$. The adversary wins $\mathbf{Exp}_{\mathcal{A},ABE}^{CCA-2}$ if it guesses the secret bit "b" with probability better than the random guess. Note, the $\mathcal{O}$DecryptABE is initialized with a decryption key $sk_{ABE}^{\mathbb{S}}$ corresponding to the whole attribute universe.

$\mathcal{O}\mathsf{KeyGenABE}(\mathbb{S})$

if $A^*(\mathbb{S}) = 1$ **return** $\perp$
$sk_{ABE}^{\mathbb{S}} \leftarrow \mathsf{KeyGenABE}(msk_{ABE}, \mathbb{S})$
return $sk_{ABE}^{\mathbb{S}}$

$\mathcal{O}\mathsf{LoREncryptABE}(m_0, m_1)$

if $|m_0| \neq |m_1|$; **return** $\perp$
$C_b^* \leftarrow \mathsf{EncryptABE}(pk_{ABE}, \mathbb{A}^*, m_b)$
return C_b^*

$\mathcal{O}\mathsf{DecryptABE}(C)$

if $C = C_b^*$; **return** $\perp$
$m \leftarrow \mathsf{DecryptABE}(pk_{ABE}, sk_{ABE}^{\mathbb{S}}, C)$
return m

Figure A.14: ABE security oracles.

Definition 30. *(ABE IND-CCA-2) The ABE scheme is IND-CCA-2 secure if for any PPT adversary $\mathcal{A}$, $|\mathsf{Pr}[\boldsymbol{Exp}_{\mathcal{A},ABE}^{CCA-2}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon(\lambda)$, where $\boldsymbol{Exp}_{\mathcal{A},ABE}^{CCA-2}$ is defined in Fig. A.15.*

$\mathbf{Exp}^{CCA-2}_{\mathcal{A},ABE}(\lambda)$

$b \xleftarrow{\$} \{0,1\}, \mathbb{A}^* \xleftarrow{\$} \mathcal{A}$

$(pk_{ABE}, msk_{ABE}) \leftarrow \mathsf{SetupABE}(1^\lambda)$

$b' \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGenABE}(.),\mathcal{O}\mathsf{DecryptABE}(.),\mathcal{O}\mathsf{LoREncryptABE}(.)}(pk_{ABE})$

if $b = b'$

  return $\top$

return $\bot$

Figure A.15: ABE IND-CCA-2 experiment.

A.2 Instantiation Protocols

Pointcheval-Sanders (PS) RDS scheme

PS is a pairing-based RDS scheme that enables the produced signature to be rerandomized and still be verifiable using the verification keys of the signer [76]. It also allows signing a commitment on a hidden message such that the resulting signature is verifiable for the message itself. The PS scheme specifies the following six procedures.

- ppGenPS. The algorithm outputs the public parameters of the scheme such that $pp_{PS} = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ where $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ defines a type-3 bilinear group map [52].

- KeyGenPS. This procedure returns the signer's secret and public key pair, the signer picks $g_1 \xleftarrow{\$} \mathbb{G}$, $\tilde{g}_1 \xleftarrow{\$} \tilde{\mathbb{G}}$ and $(a, b) \xleftarrow{\$} \mathbb{Z}^*_p$, sets $sk^{signer}_{PS} = (a, b)$ and computes $(\tilde{A}, \tilde{B}) \leftarrow (\tilde{g}^a, \tilde{g}^b)$, sets $pk^{signer}_{PS} = (g_1, \tilde{g}_1, \tilde{A}, \tilde{B})$.

- SignPS. This algorithm outputs the digital signature σ_{PS} for a message $m \in \mathbb{Z}_p$ by randomly choosing $h \xleftarrow{\$} \mathbb{G}_1 \backslash \{1_{\mathbb{G}1}\}$ and sets $\sigma_{PS} = (\sigma_{PS1}, \sigma_{PS2}) \leftarrow (h, h^{(a+b.m)})$.

- RandomizePS. This algorithm rerandomizes the digital signature on a message m and outputs σ'_{PS} by randomly choosing $r' \xleftarrow{\$} \mathbb{G}_1 \backslash \{1_{\mathbb{G}1}\}$ and computing $\sigma'_{PS} \leftarrow ({\sigma^{r'}}_{PS1}, {\sigma^{r'}}_{PS2}) \leftarrow (h^{r'}, h^{r'(a+b.m)})$.

- VerifyPS. This algorithm verifies the signature σ_{PS} over m by verifying $e(\sigma_{PS1}, \tilde{A}\tilde{B}^m) =$

$e(\sigma_{PS2}, \tilde{g}_1)$ and outputs $\{\top, \bot\}$.

- SignComPS: This a special form of SignPS algorithm that allows the signer to generate a signature over a message m by only knowing a commitment of the message g_1^m by randomly choosing $r \xleftarrow{\$} \mathbb{Z}_p^*$ and computing $\sigma_{PS} \leftarrow (\sigma_{PS1}, \sigma_{PS2}) \leftarrow (g_1^r, (g_1^a (g_1^m)^b)^r$, $\sigma_{PS} \leftarrow \mathsf{SignComPS}(sk_{PS}^{Signer}, g_1^m)$.

Abe *et al.* optimal structure-preserving signatures

Abe *et al.* demonstrated that the lower bounds for a structure-preserving signature scheme to protect against random message attack as i) it must use at least two pairing product equations to verify a signature, and ii) the signature size must be at least 3 group elements [2]. Moreover, they presented a structure-preserving signature scheme that matches such lower bounds to sign a pair of group elements $(M, N) \in \mathbb{G} \times \tilde{\mathbb{G}}$ as follows:

- ppGenAbe. This algorithm outputs the public parameters of the system such that $pp_{Abe} = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$ where $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ defines a type-3 bilinear group map [52] which is generated by $(g, \tilde{g})$.

- KeyGenAbe. picks $u, v, h, z \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $U = g^u$, $V = \tilde{g}^v$, $H = \tilde{g}^h$, $Z = \tilde{g}^z$ $(pk_{Abe}, sk_{Abe}) = ((U, V, H, Z), (u, v, w, z))$.

- SignAbe. On input of a message m in the form of $(M, N) \in \mathbb{G} \times \tilde{\mathbb{G}}$, the signer picks $r \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $R = g^r$, $S = \mathbb{G}^{z-rv}.M^{-w}$, $T = (\tilde{g}.N^{-u})^{\frac{1}{r}}$ and outputs $\sigma_{Abe} = (R, S, T)$.

- VerifyAbe. This algorithm accepts σ_{Abe} over m if $M, R, S \in \mathbb{G}$ and $N, T \in \tilde{\mathbb{G}}$ and

$$e(R, V)e(S, \tilde{g})e(M, H) = e(g, Z) \wedge e(R, T)e(U, N) = e(g, \tilde{g})$$

Finally, the authors showed how the above scheme could be extended to sign messages in $\mathbb{G}^{k_M} \times \tilde{\mathbb{G}}^{k_N}$

Groth-Sahai zero-knowledge proof system

In what follows, we introduce Groth-Sahai (GS) pairing-based Non-interactive zero-knowledge system as one of the TPBS building blocks. Groth-Sahai enables a prover to convince a verifier that a set of variables $X_i \in \mathbb{G}_1$, $\tilde{Y}_j \in \mathbb{G}_2$, $x_i, y_j \in \mathbb{Z}_p$ simultaneously satisfy a set of equations [58]. Groth and Sahai presented four general equations that can be used to represent the statement to be proved.

Pairing-product equation. For known $A_j \in \mathbb{G}_1, \tilde{B}_i \in \mathbb{G}_2$ and $\gamma_{ij} \in \mathbb{Z}_p$

$$\prod_{j=1}^{n} e(A_j, \tilde{Y}_j) \prod_{i=1}^{m} e(X_i, \tilde{B}_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, \tilde{Y}_j)^{\gamma_{ij}} = 1$$

Multi-exponentiation equation in $\mathbb{G}_1$. For known $A_j, T \in \mathbb{G}_1$ and $b_i, \gamma_{ij} \in \mathbb{Z}_p$ (Can be written for $\mathbb{G}_2$ as well)

$$\prod_{j=1}^{n} A_j^{y_j} \prod_{i=1}^{m} X_i^{b_i} \prod_{i=1}^{m} \prod_{j=1}^{n} X_i^{y_j \gamma_{ij}} = T$$

Quadratic Equation. For known $a_j, b_i \gamma_{ij}, t \in \mathbb{Z}_p$

$$\sum_{j=1}^{n} a_j y_j \sum_{i=1}^{m} x_i b_i \sum_{i=1}^{m} \sum_{j=1}^{n} x_i \gamma_{ij} y_j = t$$

The Groth-Sahai NIZK system is defined by the following three procedures:

- SetupGS. The algorithm outputs the Common Reference String (CRS) parameters of the system either in hiding or binding setting, $CRS_{GS} \leftarrow \mathsf{CRSGenGS}(1^{\lambda})$.

- ProveGS. The prover uses this algorithm to generate the proof elements π_{GS} and θ_{GS}. $\{\pi_{GS}, \theta_{GS}\} \leftarrow \mathsf{ProveGS}(CRS_{GS}, X_i, \tilde{Y}_j, x_i, y_j, C, D)$ where (C, D) are the commitments of the secret variables $X_i \in \mathbb{G}_1$, $\tilde{Y}_j \in \mathbb{G}_2$, and $x_i, y_j \in \mathbb{Z}_p$ either in a hiding or a binding setting.

- VerifyGS. The verifier uses this algorithm to verify the proof elements π and θ satisfy the prover statement and outputs $\{\top, \bot\} \leftarrow \mathsf{VerifyGS}(CRS_{GS}, C_{GS}, D_{GS}, \pi_{GS}, \theta_{GS})$.

References

[1] Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Annual Cryptology Conference. pp. 209–236. Springer (2010)

[2] Abe, M., Groth, J., Haralambiev, K., Ohkubo, M.: Optimal structure-preserving signatures in asymmetric bilinear groups. In: Annual Cryptology Conference. pp. 649–666. Springer (2011)

[3] Accelerating to Zero (A2Z) Coalition: Accelerating to zero coalition | zero emission by 2040, https://acceleratingtozero.org/, [Accessed 2024-05-23]

[4] Afia, I., AlTawy, R.: Traceable policy-based signatures with delegation. In: International Conference on Cryptology and Network Security. pp. 51–72. Springer (2023)

[5] Afia, I., AlTawy, R.: Unlinkable policy-based sanitizable signatures. In: Cryptographers' Track at the RSA Conference. pp. 191–221. Springer (2023)

[6] Afia, I., AlTawy, R.: Extended policy-based sanitizable signatures. In: Information Security and Cryptology, INSCRYPT. vol. 15544. Springer (2024)

[7] Agrawal, S., Chase, M.: Fame: fast attribute-based message encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 665–682 (2017)

[8] Association, C.V.M.: New survey underscores need for more ambitious government efforts to convince Canadians to purchase electric vehicles. http://www.cvma.ca/press-release/new-survey-underscores-need-ambitious-government-efforts-convince-canadians-purchase-electric-vehicles/, [Accessed 23-05-2024]

[9] Ateniese, G., Chou, D.H., Medeiros, B.d., Tsudik, G.: Sanitizable signatures. In: European Symposium on Research in Computer Security. pp. 159–177. Springer (2005)

[10] Badertscher, C., Matt, C., Maurer, U.: Strengthening access control encryption. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 502–532. Springer (2017)

[11] Baza, M., Sherif, A., Mahmoud, M.M., Bakiras, S., Alasmary, W., Abdallah, M., Lin, X.: Privacy-preserving blockchain-based energy trading schemes for electric vehicles. IEEE Transactions on Vehicular Technology 70(9), 9369–9384 (2021)

[12] Bellare, M., Fuchsbauer, G.: Policy-based signatures. In: International Workshop on Public Key Cryptography. pp. 520–537. Springer (2014)

[13] Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: International conference on the theory and applications of cryptographic techniques. pp. 614–629. Springer (2003)

[14] Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: Cryptographers' Track at the RSA Conference. pp. 136–153. Springer (2005)

[15] Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A., Baylina, J.: Circom: A circuit description language for building zero-knowledge applications. IEEE Transactions on Dependable and Secure Computing 20(6), 4733–4751 (2022)

[16] Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Annual Cryptology Conference. pp. 421–439. Springer (2014)

[17] Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE symposium on security and privacy (SP'07). pp. 321–334. IEEE (2007)

[18] Bibak, B., Tekiner-Moğulkoç, H.: A comprehensive analysis of vehicle to grid (v2g) systems and scholarly literature on the application of such systems. Renewable Energy Focus 36, 1–20 (2021)

[19] Bichsel, P., Camenisch, J., Neven, G., Smart, N.P., Warinschi, B.: Get shorty via group signatures without encryption. In: International Conference on Security and Cryptography for Networks. pp. 381–398. Springer (2010)

[20] Bilzhause, A., Pöhls, H.C., Samelin, K.: Position paper: the past, present, and future of sanitizable and redactable signatures. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. pp. 1–9 (2017)

[21] Blazy, O., Fuchsbauer, G., Izabachene, M., Jambert, A., Sibert, H., Vergnaud, D.: Batch groth–sahai. In: International Conference on Applied Cryptography and Network Security. pp. 218–235. Springer (2010)

[22] Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Proceedings of the 11th ACM conference on Computer and communications security. pp. 168–177 (2004)

[23] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE symposium on security and privacy. pp. 104–121. IEEE (2015)

[24] Bossuat, A., Bultel, X.: Unlinkable and invisible γ-sanitizable signatures. In: International Conference on Applied Cryptography and Network Security. pp. 251–283. Springer (2021)

[25] Boyen, X.: Mesh signatures. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 210–227. Springer (2007)

[26] Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In: International Workshop on Public Key Cryptography. pp. 317–336. Springer (2009)

[27] Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Sanitizable signatures: How to partially delegate control for authenticated data. BIOSIG 2009: biometrics and electronic signatures (2009)

[28] Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In: International Workshop on Public Key Cryptography. pp. 444–461. Springer (2010)

[29] Brzuska, C., Pöhls, H.C., Samelin, K.: Non-interactive public accountability for sanitizable signatures. In: European Public Key Infrastructure Workshop. pp. 178–193. Springer (2012)

[30] Brzuska, C., Pöhls, H.C., Samelin, K.: Efficient and perfectly unlinkable sanitizable signatures without group signatures. In: European Public Key Infrastructure Workshop. pp. 12–30. Springer (2013)

[31] Bultel, X., Lafourcade, P., Lai, R.W., Malavolta, G., Schröder, D., Thyagarajan, S.A.K.: Efficient invisible and unlinkable sanitizable signatures. In: IACR International Workshop on Public Key Cryptography. pp. 159–189. Springer (2019)

[32] Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon-hashes with ephemeral trapdoors. In: IACR International Workshop on Public Key Cryptography. pp. 152–182. Springer (2017)

[33] Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: International conference on the theory and applications of cryptographic techniques. pp. 93–118. Springer (2001)

[34] Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Annual international cryptology conference. pp. 56–72. Springer (2004)

[35] Canard, S., Jambert, A.: On extended sanitizable signature schemes. In: Cryptographers' Track at the RSA Conference. pp. 179–194. Springer (2010)

[36] Canard, S., Jambert, A., Lescuyer, R.: Sanitizable signatures with several signers and sanitizers. In: International Conference on Cryptology in Africa. pp. 35–52. Springer (2012)

[37] Canard, S., Laguillaumie, F., Milhau, M.: Trapdoor sanitizable signatures and their application to content protection. In: International Conference on Applied Cryptography and Network Security. pp. 258–276. Springer (2008)

[38] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)

[39] Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Annual International Cryptology Conference. pp. 78–96. Springer (2006)

[40] Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of Crypto 82. pp. 199–203. Springer (1983)

[41] Cramer, R., Damgård, I., MacKenzie, P.: Efficient zero-knowledge proofs of knowledge without intractability assumptions. In: International Workshop on Public Key Cryptography. pp. 354–372. Springer (2000)

[42] Damgård, I., Haagh, H., Orlandi, C.: Access control encryption: Enforcing information flow with cryptography. In: Theory of Cryptography Conference. pp. 547–576. Springer (2016)

[43] Derler, D., Samelin, K., Slamanig, D., Striecks, C.: Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. Cryptology ePrint Archive (2019)

[44] Derler, D., Samelin, K., Slamanig, D., Striecks, C.: Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. Cryptology ePrint Archive (2019)

[45] Derler, D., Slamanig, D.: Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In: International Conference on Provable Security. pp. 455–474. Springer (2015)

[46] Ernstberger, J., Chaliasos, S., Kadianakis, G., Steinhorst, S., Jovanovic, P., Gervais, A., Livshits, B., Orrù, M.: zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. Cryptology ePrint Archive (2023)

[47] Escala, A., Herranz, J., Morillo, P.: Revocable attribute-based signatures with adaptive security in the standard model. In: Nitaj, A., Pointcheval, D. (eds.) Progress in Cryptology – AFRICACRYPT 2011. pp. 224–241. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[48] Escala, A., Herranz, J., Morillo, P.: Revocable attribute-based signatures with adaptive security in the standard model. In: International conference on cryptology in Africa. pp. 224–241. Springer (2011)

[49] Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., Simkin, M.: Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In: Public-Key Cryptography–PKC 2016, pp. 301–330. Springer (2016)

[50] Fuchsbauer, G., Pointcheval, D.: Anonymous proxy signatures. In: International Conference on Security and Cryptography for Networks. pp. 201–217. Springer (2008)

[51] Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Advances in Cryptology-CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings. pp. 537–554. Springer (1999)

[52] Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics 156(16), 3113–3121 (2008)

[53] Ghadafi, E.: Stronger security notions for decentralized traceable attribute-based signatures and more efficient constructions. In: Cryptographers' Track at the RSA Conference. pp. 391–409. Springer (2015)

[54] Goldreich, O.: Foundations of cryptography:(fragments of a book (1995)

[55] Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 89–98 (2006)

[56] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535 (2021)

[57] Groth, J.: Simulation-sound nizk proofs for a practical language and constant size group signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 444–459. Springer (2006)

[58] Groth, J., Sahai, A.: Efficient noninteractive proof systems for bilinear groups. SIAM Journal on Computing 41(5), 1193–1232 (2012)

[59] Group, T.C.: Making electric transport the new normal by 2030. https://www.theclimategroup.org/ev100, [Accessed 23-05-2024]

[60] Güldorum, H.C., Şengör, İ., Erdinç, O.: Charging management system for electric vehicles considering vehicle-to-vehicle (v2v) concept. In: 2020 12th International Conference on Electrical and Electronics Engineering (ELECO). pp. 188–192. IEEE (2020)

[61] Inbasekar, K.: Sok: Hash functions in zero knowledge proofs. https://github.com/ingonyama-zk/papers/blob/main/sok_zk_friendly_hashes.pdf, [Accessed 23-05-2024]

[62] INC., E.R.A.: Canadians' awareness, knowledge and attitudes related to zero emission vehicles (zevs). https://www.nrcan.gc.ca/sites/nrcan/files/057-21-NRCan_ZEVs_Final_Report_EN_accessible.pdf, [Accessed 23-05-2024]

[63] Kiltz, E., Mityagin, A., Panjwani, S., Raghavan, B.: Append-only signatures. In: International Colloquium on Automata, Languages, and Programming. pp. 434–445. Springer (2005)

[64] Kim, O.T.T., Le, T.H.T., Shin, M.J., Nguyen, V., Han, Z., Hong, C.S.: Distributed auction-based incentive mechanism for energy trading between electric vehicles and mobile charging stations. IEEE Access 10, 56331–56347 (2022)

[65] Klonowski, M., Lauks, A.: Extended sanitizable signatures. icisc, volume 4296 of lecture notes in computer science (2006)

[66] Knirsch, F., Unterweger, A., Engel, D.: Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions. Computer Science-Research and Development 33, 71–79 (2018)

[67] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)

[68] KPMG LLP: Canadians hot on electric vehicles but cold on the ability to get them charged, finds a new KPMG in Canada survey, https://www.newswire.ca/news-releases/canadians-hot-on-electric-vehicles-but-cold-on-the-ability-to-get-them-charged-finds-a-new-kpmg-in-canada-survey-828038710.html, [Accessed 2024-05-23]

[69] Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. Cryptology ePrint Archive, Paper 1998/010 (1998), https://eprint.iacr.org/1998/010

[70] Lai, J., Ding, X., Wu, Y.: Accountable trapdoor sanitizable signatures. In: International Conference on Information Security Practice and Experience. pp. 117–131. Springer (2013)

[71] Lai, R.W., Zhang, T., Chow, S.S., Schröder, D.: Efficient sanitizable signatures without random oracles. In: European Symposium on Research in Computer Security. pp. 363–380. Springer (2016)

[72] Liang, Y., Wang, Z., Abdallah, A.B.: Robust vehicle-to-grid energy trading method based on smart forecast and multi-blockchain network. IEEE Access (2024)

[73] Maji, H., Prabhakaran, M., Rosulek, M.: Attribute-based signatures: Achieving attribute-privacy and collusion-resistance. Cryptology ePrint Archive, Report 2008/328 (2008), https://ia.cr/2008/328

[74] Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: Cryptographers' track at the RSA conference. pp. 376–392. Springer (2011)

[75] Miyazaki, K., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H., Tezuka, S., Imai, H.: Digitally signed document sanitizing scheme with disclosure condition control. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 88(1), 239–246 (2005)

[76] Pointcheval, D., Sanders, O.: Short randomizable signatures. In: Cryptographers' Track at the RSA Conference. pp. 111–126. Springer (2016)

[77] Pointcheval, D., Sanders, O.: Reassessing security of randomizable signatures. In: Cryptographers' Track at the RSA Conference. pp. 319–338. Springer (2018)

[78] Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)

[79] Pustišek, M., Kos, A., Sedlar, U.: Blockchain based autonomous selection of electric vehicle charging station. In: 2016 international conference on identification, information and knowledge in the Internet of Things (IIKI). pp. 217–222. IEEE (2016)

[80] Radi, E.M., Lasla, N., Bakiras, S., Mahmoud, M.: Privacy-preserving electric vehicle charging for peer-to-peer energy trading ecosystems. In: ICC 2019-2019 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2019)

[81] Ram, S.K., Devassy, S., Verma, B.K., Mishra, S., Akbar, S.: Review on renewable energy based ev charging system with grid support functionality. In: 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS). vol. 1, pp. 482–487. IEEE (2021)

[82] Rituraj, G., Mouli, G.R.C., Bauer, P.: A comprehensive review on off-grid and hybrid charging systems for electric vehicles. IEEE Open Journal of the Industrial Electronics Society 3, 203–222 (2022)

[83] Rivest, R.L., Shamir, A.: Payword and micromint: Two simple micropayment schemes. In: International workshop on security protocols. pp. 69–87. Springer (1996)

[84] Rocky Mountain Institute, R.: X-Change: Cars, the end of the ice age. https://rmi.org/insight/x-change-cars/, [Accessed 23-05-2024]

[85] Ronald, L.: Rivest, adi shamir, yael tauman. how to leak a secret. Advances in Cryptology-ASIACRYPT 2001/C. Boyd (ed.).-Berlin, Heidelberg: Springer-Verlag 2248, 552 (2001)

[86] Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 457–473. Springer (2005)

[87] Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24. pp. 457–473. Springer (2005)

[88] Sakai, Y., Schuldt, J.C., Emura, K., Hanaoka, G., Ohta, K.: On the security of dynamic group signatures: Preventing signature hijacking. In: International Workshop on Public Key Cryptography. pp. 715–732. Springer (2012)

[89] Samelin, K., Slamanig, D.: Policy-based sanitizable signatures. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. pp. 538–563. Springer International Publishing, Cham (2020)

[90] Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE symposium on security and privacy. pp. 459–474. IEEE (2014)

[91] Su, Z., Wang, Y., Xu, Q., Fei, M., Tian, Y.C., Zhang, N.: A secure charging scheme for electric vehicles with smart communities in energy blockchain. IEEE Internet of Things Journal 6(3), 4601–4613 (2018)

[92] Thukral, M.K.: Emergence of blockchain-technology application in peer-to-peer electrical-energy trading: A review. Clean Energy 5(1), 104–123 (2021)

[93] Tutorials, Z.A.: Incremental Merkle Tree. https://zokyo-auditing-tutorials.gitbook.io/zokyo-tutorials/tutorial-16-zero-knowledge-zk/definitions-and-essentials/incremental-merkle-tree, [Accessed 23-05-2024]

[94] Wan, Z.G., Deng, R.H., Lee, D., Li, Y.: Microbtc: efficient, flexible and fair micropayment for bitcoin using hash chains. Journal of Computer Science and Technology 34, 403–415 (2019)

[95] Wan, Z., Zhang, T., Liu, W., Wang, M., Zhu, L.: Decentralized privacy-preserving fair exchange scheme for v2g based on blockchain. IEEE Transactions on Dependable and Secure Computing 19(4), 2442–2456 (2021)

[96] Xu, Y., Safavi-Naini, R., Nguyen, K., Wang, H.: Traceable policy-based signatures and instantiation from lattices. Information Sciences 607, 1286–1310 (2022). https://doi.org/https://doi.org/10.1016/j.ins.2022.06.031, https://www.sciencedirect.com/science/article/pii/S0020025522006211

[97] Yue, X., Bi, X., Yang, H., Bai, S., He, Y.: Pap: A privacy-preserving authentication scheme with anonymous payment for v2g networks. Cryptology ePrint Archive (2023)

[98] Zhan, Y., Yi, B., Yang, Y., Shi, R., Dong, C., Huang, M.: A privilege-constrained sanitizable signature scheme for e-health systems. Journal of Systems Architecture 142, 102939 (2023)

[99] Zhao, K., Zhang, M., Lu, R., Shen, C.: A secure intra-regional-inter-regional peer-to-peer electricity trading system for electric vehicles. IEEE Transactions on Vehicular Technology 71(12), 12576–12587 (2022)

[100] Zhou, S., Lin, D.: Unlinkable randomizable signature and its application in group signature. vol. 2007, p. 213 (08 2007). https://doi.org/10.1007/978-3-540-79499-8_26