Detecting Rogue Decryption in (Threshold) Encryption via Self-Incriminating Proofs

James Hsin-yu Chiang[1⋆], Bernardo David[2⋆⋆], Tore Kasper Frederiksen, Arup Mondal[4⋆⋆⋆†], and Esra Yeniaras[2⋆⋆†]

[1] Aarhus University
jachiang@cs.au.dk
[2] IT University of Copenhagen
bernardo@bmdavid.com, esye@itu.dk
[3] Zama
tore.frederiksen@zama.ai
[4] Ashoka University
arup.mondal_phd19@ashoka.edu.in

Abstract. Keeping decrypting parties accountable in public key encryption is notoriously hard since the secret key owner can decrypt any ciphertext. Threshold encryption aims to solve this issue by distributing the power to decrypt among a set of parties, who must interact via a decryption protocol. However, such parties can employ cryptographic tools such as Multiparty Computation (MPC) to decrypt arbitrary ciphertexts *without being detected*. We introduce the notion of (threshold) encryption with Self-Incriminating Proofs, where parties must *produce a self-incriminating proof of decryption when decrypting every ciphertext*. In the standard public key encryption case, the adversary could destroy these proofs, so we strengthen our notion to guarantee that the proofs are published on public ledger when decryption succeeds. This creates a decryption audit trail, which is useful in scenarios where decryption power is held by a single trusted party (*e.g.,* a Trusted Execution Environment) who must be kept accountable. In the threshold case, we ensure that at least one of the parties who execute the decryption protocol will learn a self-incriminating proof, even if they employ advanced tools such as MPC. The fact that a party learns the proof and may leak it at any moment functions as a deterrent for parties who do not wish to be identified as malicious decryptors (*e.g.,* a commercial operator of a service based on threshold encryption). We providing matching constructions of our notions under appropriate assumptions. In the threshold case, we build on recent results on Individual Cryptography (CRYPTO 2023).

1 Introduction

The now ubiquitous notion of public key encryption [29] gives full control over the privacy of a message encrypted under a given public key to the party who knows the corresponding secret key. In other words, whoever has the corresponding secret key has full discretion to decrypt ciphertexts generated under a given public key, learning the plaintext message and deriving whatever utility it may afford. More importantly, the custodian of the secret key may do so at any time without being detected. While this is unproblematic when the owner of a secret key stores it locally, it poses significant security risks when this is not the case.

Even though this may seem like a general issue for any kind of outsourced key management, the risks are significantly lower for the other main public key primitive of digital signatures. This is because a digital signature is in itself an unforgeable proof that a cryptographic operation has been carried out; thus, any use of an adversarially constructed signature will in itself provide detection of the misuse. Since signatures only provide value only when validated, this means that malicious signing will likely be observable by all honest verifiers.

Since holders of private signing keys will likely have secure storage and use the key as part of a larger protocol, it means that as long as the value of producing a "rogue" signature is less than that of following the protocol, then they are incentivized not to misuse any stored keys. As an example, consider a custodian company providing on-site storage of signing keys for cryptocurrency blockchains: Any rogue signature could drain a "single" user's wallet, but would also completely destroy the reputation of that

⋆ This work was supported by SUI Foundation.
⋆⋆ This work was supported by the Independent Research Fund Denmark (IRFD) grant number 0165-00079B.
⋆⋆⋆ This work was done while visiting the IT University of Copenhagen with supported by Mphasis F1 Foundation.
† This work was supported by CPH Fintech.

company. For decryption, simply learning the plaintext is often the end-goal, as the knowledge can be used in an undetectable manner. For example, consider the bids in a first-price auction; a rushing adversary knowing the bids from all other parties can win the auction by paying just above the highest honest bid.

In both the cases of digital signatures and of public key encryption, security can be greatly enhanced by splitting up the secret key among multiple parties via *threshold cryptography* [27, 28]. In this setting, no single party can decide to use the secret key. Instead, at least $t+1$ out of a total of n parties, must cooperate to successfully authorize any cryptographic operation using the secret key. Hence the committee holding shares of the key can enforce rules about when a cryptographic operation should be carried out. Thus, both key storage, and key-use authorization is secured through consensus by a certain amount of parties in the committee. However, for the decryption setting, this notion still requires that no more than t parties act maliciously. Otherwise, the same issue from before occurs, as a large enough set of corrupted parties in the decryption committee can easily decrypt any arbitrary ciphertext without detection.

It is notoriously hard to achieve accountability for both standard and threshold encryption schemes when the party (or parties) with the power to decrypt acts maliciously. Despite decades of research on public key cryptography, the majority of current encryption schemes still allow malicious parties with knowledge of the secret key to perform "rogue" (unauthorized) decryptions of arbitrary ciphertexts without being detected. Even if we construct schemes that readily allow detection of rogue decryptions through publicly available information, powerful cryptographic tools such as anonymous channels [18] and Multiparty Computation (MPC) [19, 36] can be employed by malicious parties to avoid detection. In this context, we ask the following question:

Is it possible to detect rogue decryptions in (threshold) public key encryption schemes even when secret key holders use cryptography to avoid detection?

We answer this question in the affirmative both in the standard setting where a single party holds the decryption key and in the threshold setting where the secret key is shared among multiple parties. We do so by proposing definitions and matching constructions of (threshold) public key encryption schemes where the decryption process intrinsically forces parties to produce proofs that decryption has happened. We call such proofs *Self-Incriminating Proofs* (SIP), as they reveal that a party (or committee of parties) has acted to decrypt a given ciphertext. In the threshold setting, our notion guarantees security even against adversaries who use cryptographic tools such as MPC to avoid producing a SIP. Hence, our threshold scheme thwarts so-called "cryptovirological" attacks [55].

Besides settling a long-standing theoretical question, our solution also finds practical applications in several real-world scenarios where public key cryptography is used. In general, any application of public key encryption where parties with the ability to decrypt must be kept accountable can benefit from our new notions and constructions. In particular, in cases where the secret key for a standard public key encryption is stored in a Trusted Execution Environment (TEE), our notion can help detect the TEE's malfunctioning (or malfeasance) if it performs rogue decryptions. On the other hand, in many applications of threshold encryption, clients submit ciphertexts to be decrypted by implicitly trusted committees that provide threshold decryption as a service. Our threshold notions can help detect committee malfeasance in such scenarios.

One concrete example of where accountability in decryption can bring utility is in the case of "encrypted mempools". In the blockchain literature, a mempool is the set of transactions that are known, but have not yet been written to the ledger. More specifically, these transactions are known to the miners/validators, who can mount "front-running" attacks to gain financial profit via so-called "miner extractable value" [23]. A common way to prevent this is to encrypt the transactions in mempools until they are written [51] using standard public key encryption executed by TEEs or threshold encryption executed as a service by committees. Thus, when using encrypted mempools there is a clear incentive for the TEE or the committee to misbehave and purposefully decrypt the pools to front-run. On the other hand, if these rogue decryptions can be detected, clients can react to the malfeasance appropriately, such as by boycotting the key management committee or by taking away collateral funs locked in a smart-contract. Thus, incentivizing the committee not to carry out rogue decryptions.

In other cases, the decrypting committee might not be incentivized to misbehave, but external parties might be incentivized to bribe them to maliciously decrypt and share the plaintext. This, for example, occurs when encryption is used as part of a protocol facilitating computation on private data from multiple clients. In the threshold setting, this can happen when using outsourced secure multi-party computation [44] with a scheme based on homomorphic encryption [8, 25]. For example, in the case of loan benchmarking; where clients are banks that input customers' confidential financial information [24], which, if revealed to competing banks, could give them a competitive advantage. In the blockchain

literature, we see this in systems realizing privacy-preserving smart contracts, such as Oasis [49] (based on TEEs) or Zama's fhEVM [22] (based on fully homomorphic encryption with threshold decryption). While knowledge of other client's private information could in itself be valuable, certain smart contracts are more vulnerable than others. Smart contracts handling personal information are particularly vulnerable, as a rogue decryption could expose sensitive personally identifying information protected by privacy regulations.

1.1 Our Contributions

We look at the issue of detecting rogue decryptions of arbitrary ciphertexts in standard and threshold public key encryption schemes as a problem on its own and introduce the notion of a "Self-Incriminating Proof" (SIP) as a central tool to solve it. A SIP proves, unequivocally, to any third party that a given ciphertext has been decrypted, while providing unforgeability properties ensuring that it cannot be generated unless the ciphertext is indeed decrypted. Starting from this concept, we define and construct the following new primitives: *Public* Self-Incriminating Proof (PSIPE), and *Threshold Encryption* with Self-Incriminating Proof (TSIPE). These new primitives enrich standard public key encryption with the guarantees that a SIP must be published on a public ledger (PSIPE) during the decryption process, and enrich threshold encryption with the guarantee that a SIP must be learned by one of the decrypting parties (TSIPE) if a ciphertext is decrypted. We summarize our main contributions as follows:

- Definitions, and two matching constructions, of the notion of Encryption with Public Self-Incriminating Proof (PSIPE), which guarantees that a SIP is published, in the context of non-threshold encryption.
- Definitions, and a matching construction, for the notion of Threshold Encryption with Self-Incriminating Proof (TSIPE), building on the Individual Cryptography paradigm introduced by Dziembowski *et al.* [30].

Both our PSIPE constructions require an underlying Proof-of-Stake (PoS) public ledger, which is assumed as a setup, and is required to guarantee that a SIP is published, *i.e.,* that all honest parties learn the SIP. While our constructions are based on any standard (non-threshold) public key encryption scheme, they can be trivially be generalized to the threshold setting by building on a threshold encryption scheme instead.

In the threshold setting, instead of assuming access to a PoS ledger to publish the SIP, we assume that rogue decryptions are desincentivized by the fact that at least one party in the decryption committee is guaranteed to learn a SIP. However, not guaranteeing that a SIP is published could potentially allow a set of parties to silently execute arbitrary MPC protocols that can be used to execute decryption *without* leaking the SIP. Hence, we construct a threshold decryption protocol with a *guarantee* that at least one of the parties who cooperate in decrypting a given ciphertext will learn a SIP, even if cryptographic techniques such as MPC are used by the rogue decryption committee members. Our TSIPE construction forces colluding malicious parties to disclose their private key shares to each other in order to do a rogue decryption *without* revealing a SIP to at least one of them. A private key share can then be used as a proof of malicious behavior, similar to a SIP. Even though a SIP or key shares will only be learned by colluding parties, as it is possible to incentivize these parties to share this information publicly and self-inncriminate the rogue committee (*e.g.* by offering financial incentives).

1.2 Technical Overview

We present a PSIPE construction, denoted $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$, based on a public ledger realized by a Proof-of-Stake blockchain, a signature scheme and a thresholdizable Batched IBE scheme (TBIBE). TBIBE is a special threshold version of an Identity Based Encryption (IBE) scheme introduced by Agrawal *et al.* [1] which allows public aggregation of any subset of identity secret keys. That is, multiple identities can be succinctly combined into a single decryption key, when a threshold of master key share holders agree. In the appendix, we present an alternative construction of PSIPE, denoted $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$, which uses an extractable witness encryption (eWE) scheme for a specific language instead of a TBIBE scheme. We show how to realize this eWE efficiently from standard assumptions using the underlying blockchain via techniques from [42] suitably combined with a publicly verifiable secret sharing scheme for anonymous committees [17] to force misbehaving parties to reveal themselves as proposed in [15]. We also present a construction of TSIPE based on a (regular) threshold encryption scheme, a commitment scheme, a non-interactive zero-knowledge proof, and an *MPC-hard function* (which is a function that cannot be feasibly computed by an MPC protocol [30]), which can be realized without the use of a public ledger.

In more detail, the core idea of $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ is to first slightly modify the protocol steps for validators (*i.e.* parties executing the PoS blockchain protocol) of the underlying PoS ledger so that each validator has a share of the secret master key for a TBIBE scheme, which can then be used to permit decryption of a ciphertext, after a SIP has been linked to a given ciphertext. More concretely, when encrypting a message msg, it is encrypted not only under a standard PKE, but also toward a random unique id under the TBIBE. During decryption, the decryptor first publishes, on the ledger, a signature on the doubly encrypted ciphertext, which will function as a SIP. Now, at the beginning of each new round on the ledger, the validators will recover all SIPs posted in the previous round. They then verify these SIPs against the appropriate public verification keys. They then construct a *single* secret key that can be used to decrypt *any* of the verified ciphertexts. This key is then published in the new block on the ledger. After the validators have verified the signature of the decryptor, they can use the newly derived TBIBE secret key to decrypt the outer ciphertext, and then use the standard PKE key to decrypt the inner message.

Similarly to $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$, the core idea of $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ is to use a signature on the ciphertext to decrypt, as a SIP. However, in $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ the message encryption is performed using witness encryption, where the witness required for the decryption is a proof that a SIP (signature) has been posted to a given smart contract on the public ledger. More concretely, the encryptor reads the current state of the ledger and then witness encrypts their message under the current state a public verification key s.t. a signature on the ciphertext *must* be included on the ledger in an appropriate contract, before decryption is permitted. Specifically, such a constraint is handled by the fact that each block is signed by a set of validators that can be linked back to any previous block. Hence, the decryptor is forced to publish a signature on the ciphertext to decrypt, to have the blockchain evolve to a new block that will be the witness they need to actually decrypt the message. We build on a Proof-of-Stake blockchain as it has been shown that it is possible to non-interactively verify whether a given blockchain state has evolved from a given previous state via an honest protocol execution [41]. We later also use the blockchain to realize the eWE scheme based on standard assumptions using the "eWE on Blockchains" construction from [42], using techniques from [15] to ensure that it cannot be abused to decrypt without generating a SIP.

Our threshold construction, $\mathbf{\Pi}_{\mathsf{TSIPE}}$ on the other hand does not directly require the use of a public ledger (but it can be used in conjunction with one in order to incentivize honesty of the threshold parties). Instead, it builds on top of regular threshold encryption, non-interactive zero-knowledge proofs, commitments and hash functions. The idea is to combine these in a clever manner s.t. any single party learning a sufficient amount of partial decryptions can efficiently produce a SIP that a given ciphertext was indeed decrypted. While this may on the surface sound like a weak result, it is important to note that at least one party will learn the proof and external incentives can encourage, even a malicious party, to share this. Still, the main technical issue is to ensure that it is not possible for a set of malicious parties to decrypt *without* producing a SIP *or* leaking their private key shares, e.g. using MPC. To achieve this, we start from the *distributed adversarial model* and the concept of *MPC-hard* function, both introduced in the recent work on Individual Cryptography [30]. In the distributed adversary model, each corrupted party is a sub-adversary that acts individually, without being controlled by a monolithic adversary that gets full control over the corrupted parties. Hence the sub-adversaries must organize to collaborate on an attack without the sub-adversary sharing their own internal state between each other. In our case, we assume that a sub-adversary is disincentivized from leaking their private decryption share (in their internal state) to other sub-adversaries[5]. Hence their only option for decryption without producing a SIP is to use MPC to compute a decryption, where they each input their key shares and selectively only open the actual decrypted output and not the SIP which is produced as part of the decryption process. We prevent this using the notion of an *MPC-hard* function, which is fast to compute in plain but slow to compute in MPC [6].

We observe that $\mathbf{\Pi}_{\mathsf{TSIPE}}$ should only be considered the first step in insuring that secure outsourcing of secret keys, as it does not consider *tracing* a SIP to a specific set of parties. A promising direction for identifying cheating parties is integrating recent research in traitor tracing [20, 46] or traceable secret sharing [14, 43] into our TSIPE construction, which we leave as future work.

[5] Observe that assuming access to a distributed tokenized ledger with Turing complete smart contracts, such as Ethereum, this can be incentivized through the folk-lore approach of an "ante-contract". Such a contract require each threshold party to "lock-in" a certain amount of tokens during key generation, which can only be retrieved in full by a quorum of the threshold parties agreeing, *or* partially retrieved through proving knowledge of the party's secret key share. Thus a party will lose some of their tokens if their secret key share is leaked *and* will publicly out themselves as having lost their key share.

[6] We highlight that this is unrelated to the notion of "time-lock puzzles" [50] which inherently focuses on timing the hardness of computing a specific function on a given *public* value.

1.3 Related Work

The concept of identifying malicious decryptions carried out by a key-holder was initially considered by Ryan [52] and dubbed *accountable encryption*. Later, it was more formally defined by Li *et al.* [47], who showed security definitions and protocols for the identification of malicious decryptions when the secret key is stored in a single TEE that always executes the decryption protocol correctly.

Using cryptography for malicious purposes was first considered by Young and Yung [55] in 1996. They studied the idea of "Cryptovirology", which consists of using cryptographic tools maliciously. Specifically, the work of [55] focuses on the malicious use of public-key encryption. In our case, we consider the use of distributed cryptographic protocols (*e.g.,* MPC) as a tool to subvert (threshold) encryption and investigate how to prevent such attacks.

Another approach to prevent leaking secrets has been studied extensively in the context of *traitor-tracing* [20,39,46]. Chor *et al.* [20] described traitor tracing as a method for providing personal decryption keys to users, such that there is a single encryption key corresponding to all the decryption keys, and any possible decryption key, even one that was generated by a coalition of corrupt users (traitors), identifies the personal keys that were used to generate it.

Recently, Goyal, Song, and Srinivasan [43] initiate the study of preventing leakage in the context of standard secret sharing schemes and introduced the notion of *traceable secret sharing*. In general, these are secret sharing schemes that allow leaked shares to be traced back to the share holders responsible for the leak. Boneh *et al.* [14] also introduced a new definitions for traceable secret sharing, and presented two efficient constructions of traceable secret sharing based on two classic secret sharing schemes, i.e., standard Shamir secret sharing [53] and a variant of Blakley secret sharing [11]. Another recent work [40] introduced verifiable secret sharing schemes with an associated mechanism that provably incentivizes parties who obtain a reconstructed secret to publicly identifying the parties who executed the reconstruction. However, these works only consider the case of reconstructing a secret but not the case where shares of a secret key are used to threshold decrypt a ciphertext without ever reconstructing the secret key. Moreover, they are not easily extended to the threshold decryption case.

Recently, Boneh *et al.* [13] initiated the study of traitor-tracing in the context of threshold decryption and showed several constructions for it. While this work allows for identifyingt the exact subset of parties who participated in a rogue threshold decryption, it requires unrestricted access to a *stateless decoder*, an oracle that allows for a polynomial number of decryption queries without keeping state between successive queries. If the colluding decryption committee members use a more sophisticated method to provide rogue decryptions (*e.g.* running the original threshold decryption protocol or decrypting via MPC) instead of providing such a stateless decoder, they can easily avoid detection. Our schemes work even against such clever attacks, at the cost of not individually identifying the specific colluding parties but only detecting decryption.

As mentioned in [30], none of above approaches [13,14,20,43,46,55] considers a distributed adversarial model [30]. On the other hand, we approach the problem of detecting when a distributed adversary performs (threshold) decryption but does not necessarily require the identification of individual sub-adversaries (*i.e.,* corrupted parties) who take part in a decryption process.

The notions of collusion-free [2, 4] and collusion-preserving [3] MPC address the setting where corrupted parties cannot collude. In other words, corrupted parties are not fully controlled by a monolithic adversary but instead must act individually according to their own strategies without coordination. These works investigate the construction of MPC protocols by leveraging the fact that corrupted parties do not communicate (or have no incentive to communicate). In our setting, we assume instead that corrupted parties act individually but can communicate and have an incentive to do so but are disincentivized from sharing their secret key shares with each other.

As previously mentioned, recently, Dziembowski *et al.* [30] defined the notion of *individual cryptography* in which they consider a distributed adversarial model. They construct two individual cryptographic primitives: (i) *proof of individual knowledge* (PoIK), a tool for proving that a given message is fully known to a single "individual" machine, *i.e.,* that the data is not shared among a group of parties; and (ii) *individual secret sharing* (ISS), a scheme for sharing a secret between a group of parties so that the parties do not know the secret as long as they do not reconstruct it, while reconstruction ensures that if the shareholders attempt to collude, one of them will learn the entire secret. Concurrently, Kelkar *et al.* [45] introduced the concept of *proof of complete knowledge* (PoCK) which is very similar to the notion of PoIK in [30]. A PoCK guarantees that a single party has complete knowledge of its secret. In particular, Kelkar *et al.* [45] showed a construction of PoCK that directly achieves a zero-knowledge property.

2 Preliminaries

Basic Notation. We denote the security parameter by $\lambda \in \mathbb{N}$. In threshold settings, we use n to denote the number of parties and t the corruption threshold such that $0 < t < n$. We use $[a, b]$ for $a, b \in \mathbb{Z}$, $a \leq b$, to denote $\{a, a+1, \ldots, b-1, b\}$. $[b]$ denotes the set $[1, b]$. We denote the concatenation of x and y by $(x\|y)$. Given a set $\mathcal{X}$, we denote by $x \xleftarrow{\$} \mathcal{X}$ the sampling of a value x from the uniform distribution on $\mathcal{X}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if it vanishes faster than any polynomial. We denote by $x = \mathsf{val}$ or $x \leftarrow \mathsf{val}$ the assignment of a value val to the variable x. We denote evaluating a PPT algorithm $\mathcal{A}$ that produces an output out from an input in with randomness $r \xleftarrow{\$} \{0,1\}^*$ as $\mathsf{out} \leftarrow \mathcal{A}(\mathsf{in}; r)$, omitting the randomness when it is obvious or not explicitly required. By $\mathcal{A}^{\mathcal{O}^{\mathsf{alg}}_{\mathsf{param}}}$ we denote that we run $\mathcal{A}$ with oracle access to $\mathcal{O}^{\mathsf{alg}}_{\mathsf{param}}$, *i.e.,* $\mathcal{O}$ executes alg with parameters param on inputs of $\mathcal{A}$'s and returns the corresponding outputs.

2.1 Building Blocks

Due to space constraints, we defer the building blocks preliminaries on *digital signature*, *commitment scheme*, *extractable witness encryption*, *threshold encryption*, *non-interactive zero-knowledge*, *pseudorandom functions*, to Appendix A.1, A.2, A.3, A.4, and A.5, and A.6, respectively.

Thresholdizable Batched IBE. In traditional Identity-Based Encryption (IBE) schemes, each message is encrypted for a specific identity, for each of which a private decryption key is associated. In order to decrypt, one must hold a decryption key for the identity under which the message has been encrypted. Private decryption keys are derived based on an identity, using a master secret key. If there are many distinct identities in play then an IBE scheme can quickly become inefficient. This is in particular true in the setting where the decryptors hold *multiple* identities; and hence multiple decryption keys. *Batched IBE* [1] overcomes this issue by allowing the aggregation of any subset of identities into a single private decryption key. Furthermore, batched IBE increases granularity further by introducing batching labels, s.t. decryption can be further limited to a set of identities associated with a specific label. A batched IBE scheme can again be generalized to the threshold setting, where initial key generation, along with derivation of a decryption key, based on a set of identities, needs to be carried out by a threshold of parties.

Below we sketch the algorithms for threshold batched IBE (TBIBE) based on the work of Agrewal *et al.* [1]. We refer the reader to Appendix A.7 for the detailed security definition of a TBIBE scheme.

Definition 1 (Thresholdizable Batched IBE). *A Thresholdizable Batched IBE scheme* TBIBE *is a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Digest}, \mathsf{ComputeKeyShare}, \mathsf{ComputeKeyAggregate})$*, which have the following syntax:*

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\mathsf{B}, 1^n)$*, the setup algorithm take as input a security parameter* 1^λ*, a batch size* $\mathsf{B} = \mathsf{B}(\lambda)$ *and number of threshold servers* $n = n(\lambda)$*, and It outputs public parameter* pp *which includes a description of the message space* $\mathcal{M}$*, identity space* $\mathcal{I}$*, batch label space* $\mathcal{B}$ *and ciphertext space* $\mathcal{C}$*.*
2. $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{msk}_i)\}_{i\in[n]}) \leftarrow \mathsf{KGen}(\mathsf{pp})$*, is a randomized, threshold algorithm that takes as input the public parameter* pp*, and a global public key* pk *along with a pair of partial master secret, and public, keys* $(\mathsf{msk}_i, \mathsf{pk}_i)$*, for each threshold server* P_i*.*
3. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}, \mathsf{id}; \mathsf{b})$*, is the encryption algorithm that takes as input a global public key* pk*, a message* $\mathsf{msg} \in \mathcal{M}$ *and an identity* $\mathsf{id} \in \mathcal{I}$ *and a batch label* $\mathsf{b} \in \mathcal{B}$*, and outputs a ciphertext* $\mathsf{ct} \in \mathcal{C}$*.*
4. $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\})$*, is a deterministic algorithm that takes as input the global public key* pk *and* B *number of identities* $\{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$ *where each* $\mathsf{id}_i \in \mathcal{I}$*, and outputs a digest* d *of the set of identities.*
5. $\mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, \mathsf{d}; \mathsf{b})$*, the key share computation algorithm takes as input the partial master secret key* msk_i*, a digest* d *and a batch label* $\mathsf{b} \in \mathcal{B}$*, and outputs a partial digest-batch, label-specific secret key* sk_i*.*
6. $\mathsf{sk} \leftarrow \mathsf{ComputeKeyAggregate}(\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}, \mathsf{d}; \mathsf{b})$*, the secret key aggregation algorithm that takes as input all pairs of partial public keys, and partial digest-batch label-specific secret keys* $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}$ *and a batch label* $\mathsf{b} \in \mathcal{B}$*, and outputs a digest-batch label-specific secret key* sk*. This can then be used to decrypt any ciphertext encrypted under* any *of the IDs used to derive* d*, which was used to construct the digest-batch, label-specific secret keys for batch label* b*.*
7. $\mathsf{msg} \leftarrow \mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, \mathsf{d}, \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}, \mathsf{id}; \mathsf{b})$*, the decryption algorithm takes as input a ciphertext* ct*, secret key* sk*, digest* d*,* B *number of identities* $\{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$*, an identity* id *and a batch label* $\mathsf{b} \in \mathcal{B}$*, and outputs a message* $\mathsf{msg} \in \mathcal{M}$ *if any of the identities were used to derive* sk*.*

A TBIBE *scheme must satisfy the properties of* ***Correctness*** *and* ***Security*** *which we formally describe in Appendix A.7.*

In the rest of that paper, we preclude the explicit batch label b in the algorithms defined above, as this will always be static, in our use-case.

2.2 Proof-of-Stake (PoS) Blockchains

In PoS-based blockchains (public ledgers), each participant is associated with some "stake" in the system and a lottery mechanism ensures that each party succeeds in generating the next block with probability proportional to its stake in the system. To formally argue about executions of such protocols, we use the framework of Goyal *et al.* [41] which, in turn, builds on the analysis done in [32, 48]. The main property we are interested in is the possibility to non-interactively prove that a blockchain has "evolved", *i.e.* a given future state was obtained by honestly executing the protocol from a known past state. We remark that in [41] it is proven that there exist PoS blockchain protocols (*e.g.* [10, 26]) with this property.

More specifically, at the high level, we define a blockchain by the following algorithms:

Definition 2 (Blockchain Protocol [41]). *A blockchain protocol* Γ^V *consists of the following three polynomial-time algorithms* $(\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ *with a validity predicate* V*, are described as follows:*

- $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$*, the algorithm takes as input the security parameter* 1^λ *and outputs* st *which is the local state of the blockchain along with metadata.*
- $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$*, the algorithm takes as input the security parameter* 1^λ *and state* st*, and outputs the longest sequence of valid blocks* $\mathtt{B}$ *(with respect to* V*).*
- $\mathsf{Broadcast}(1^\lambda, m)$*, the algorithm takes as input the security parameter* 1^λ *and a message* m*, and broadcasts the message* m *over the network to all parties executing the blockchain protocol.*

Execution of $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$, with appropriate restrictions on adversary $\mathcal{A}$ and environment $\mathcal{Z}$, will yield certain compliant executions $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ where some security properties will hold with overwhelming probability. Existing works, such as [32,48], have converged toward a few security properties that characterizes blockchain protocols. These include: *Common Prefix* or *Chain Consistency*, *Chain Quality* and *Chain Growth*.

Definition 3 (Common Prefix). *Let* $\kappa \in \mathbb{N}$ *be the common prefix parameter. The chains* $\mathtt{B}_1$*,* $\mathtt{B}_2$ *possessed by two honest parties* P_1 *and* P_2 *satisfy* $\mathtt{B}_1^{\lceil\kappa} \preceq \mathtt{B}_2$*.*

Definition 4 (Chain Growth). *Let* $\tau \in (0, 1]$*,* $s \in \mathbb{N}$ *and let* $\mathtt{B}_1, \mathtt{B}_2$ *be as above, then* $\mathtt{len}(\mathtt{B}_2) - \mathtt{len}(\mathtt{B}_1) \geq \tau s$ *where* τ *is the speed coefficient.*

Definition 5 (Chain Quality). *Let* $\mu \in (0, 1]$ *and* $\kappa \in \mathbb{N}$*. Consider any set of consecutive blocks, of length at least* κ*, from an honest party's chain* $\mathtt{B}_1$*. The ratio of adversarial blocks in the set is* $1 - \mu$ *where* μ *is the quality coefficient.*

In this work, we propose two constructions PSIPE-TBIBE (Sec. 3.1) and PSIPE-eWE (Appendix B) which assume a secure blockchain protocol. Whilst the security properties above suffice to argue security of PSIPE-TBIBE, for the security proof of the PSIPE-eWE construction , additional formalism is required. More specifically, we need to be able to non-interactively prove that a blockchain has "evolved", *i.e.* a given future state has been obtained by honestly executing the protocol from a known, past state. This is achieved via the *distinguishable forking* property. At the high level, this property asserts that a sufficiently long sequence of blocks produced under honest protocol execution can consistently be *distinguished* from any fork generated adversarially. Moreover, the total stake committed to these sequences (known as their proof-of-stake fraction), which can be computed efficiently, serves as a distinguishing criterion. The relevant formalism for this has been introduced by Goyal and Goyal [41], but we defer the introduction of this to Appendix A.8 due to space constraints.

2.3 The Distributed Adversarial Model and MPC-hard Functions

In this section, we describe the distributed adversarial model, the notion of an MPC-hard function, along with a concrete construction, all of which are taken almost verbatim from [30], and are required for our TSIPE construction.

The *distributed adversary* [30] is a tuple $\mathcal{A}_1, \ldots, \mathcal{A}_a$ of poly-time interactive machines (also called the *sub-adversaries*) that can efficiently evaluate a cryptographic task via an MPC protocol or a similarly distributed manner. Dziembowski *et al.* [30] define the notion of an MPC-hard cryptographic task. Informally, a cryptographic task/function is MPC-hard if executing it securely in a distributed way takes a significant amount of time. This implies that if a cryptographic task is MPC-hard, then to run it efficiently, the parties need to execute it *individually*. In other words, by using an MPC-hard task, we want to enforce that the distributed adversary must run the cryptographic task locally. More concretely, if the adversaries manage to complete the cryptographic task within some specified time bound, then one of the adversaries, say $\mathcal{A}_j$, must know (or have "knowledge" of) the other parties' secret input to computation.

MPC-hard Functions and a (δ, Υ)-Distributed Adversary. The distributed adversary $\mathcal{A}_1, \ldots, \mathcal{A}_a$ is given access to a special oracle $\mathcal{O}_{\mathsf{Fun}}$ that allows evaluation of a fixed input-length function $\mathsf{Fun} : \{0,1\}^\alpha \to \{0,1\}^\beta$. The oracle accepts queries of the form $(x, \mathtt{mode})$, where $x \in \{0,1\}^\lambda$ and $\mathtt{mode} \in \{\mathsf{fast}, \mathsf{slow}\}$. If $\mathtt{mode} = \mathsf{fast}$, then a query is called *fast*, otherwise, the query is called *slow*. Let us give some intuition on these two modes:

1. The fast queries are Fun function evaluations that a sub-adversary $\mathcal{A}_j$ runs locally, in this case $\mathcal{A}_j$ has to know (or have "knowledge") the x entirely.
2. The slow queries model an evaluation of the Fun function using an MPC protocol. In particular, this means that the sub-adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_a$ can learn $\mathsf{Fun}(x)$ without knowing x.

Each query coming from any of the sub-adversaries $\mathcal{A}_j$ is answered to $\mathcal{A}_j$ with $\mathsf{Fun}(x)$ (we also say that $\mathcal{A}_j$ evaluated Fun on input x).

The execution of a protocol with a distributed adversary can now be divided into a *preprocessing* phase (where the adversary does not know the relevant input) followed by an *online* phase where relevant input is known. In the preprocessing phase, the sub-adversaries can send an arbitrary number of slow queries to the oracle $\mathcal{O}_{\mathsf{Fun}}$. In the online phase, however, the sub-adversaries are limited in the number of queries they can make to the oracle $\mathcal{O}_{\mathsf{Fun}}$. We observe that the notion of an independent preprocessing phase is common for MPC protocols [8, 25].

More specifically, we say $\mathcal{A}_1, \ldots, \mathcal{A}_a$ is a (δ, Υ)-distributed adversary relative to a function Fun if the total number of slow queries made by any sub-adversary $\mathcal{A}_j$ to $\mathcal{O}_{\mathsf{Fun}}$ in the *online* phase is bounded by Υ, for at most δ rounds. The total number of fast queries is only bounded by the time complexity of the adversaries (*i.e.,* it is polynomial in λ). The adversaries run in at most δ rounds, where each of them has the following form: (1) each sub-adversary $\mathcal{A}_j$ performs some local computation, at the end of which $\mathcal{A}_j$ outputs a string str_j, and (2) each str_j is delivered to every other sub-adversary.

Instantiating an MPC-hard Function. Dziembowski *et al.* [30] defined an MPC-hard function based on iterative hash function computation. They modeled the function Fun as the evaluation of a fixed input-length hash function $\mathcal{H} : \{0,1\}^\alpha \to \{0,1\}^\beta$ (and the access to an oracle $\mathcal{O}_{\mathsf{Fun}}$ as $\mathcal{O}_{\mathcal{H}}$). Since the evaluation of a hash function using MPC technology is conceivably much slower than using a regular CPU or even customized hardware, like an ASIC, we assume that the budget of the adversary for such queries is comparably small, *i.e.,* bounded by some parameter.

We reproduce the $\mathsf{scratch}$ function from [30] in Figure 1. The main idea behind $\mathsf{scratch}$ is that it forces a party to *sequentially* compute d times $\mathcal{H}$ on every block s_l of $s = (s_1 \| s_2 \| \ldots \| s_n)$. The $\mathsf{scratch}$ procedure takes as input two random $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ where each $|s_l| = \alpha - \beta - 2$ and $z \xleftarrow{\$} \{0,1\}^\beta$, and a nonce $w \in \{0,1\}^{\alpha-\beta-2}$, and then it sequentially computes d times hashes $\mathcal{H}$ on every block of s and finally outputs a value q (refer to Figure 1(a) in [30] for a schematic overview of $\mathsf{scratch}$ procedure.). Note that the scratch procedure computes $nd + 1$ hashes $\mathcal{H}$ in total.

For the purpose of our TSIPE construction (in Figure 10), we use the $\mathsf{scratch}$ function and the goal is finding a β number of nonces $w_1, \ldots, w_\beta \in \{0,1\}^{\alpha-\beta-2}$ such that the first ζ bits (where ζ can be a function of β) of each $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ are zero for all $i \in [1, \beta]$.

We refer the reader to Appendix A.9 for more details on $\mathsf{scratch}$, along with the security games for a distributed adversary.

3 Encryption with Public Self-Incriminating Proof

In this section, we start by introduce the notion of public key Encryption with Self-Incriminating Proof (PSIPE). This notion captures the fact that a decryptor who knows a secret key must produce a Self-Incriminating Proof (SIP) when they decrypt a ciphertext generated under the corresponding public key.

MPC-hard Function: scratch

Parameters: $\alpha, \beta, d, n \in \mathbb{N}$.
Building block: A hash function $\mathcal{H} : \{0,1\}^{\alpha} \to \{0,1\}^{\beta}$ with $\alpha \geq 2\beta$ is computed by accessing the special oracle $\mathcal{O}_{\mathcal{H}}$ (where $\mathcal{O}_{\mathcal{H}}$ allows for evaluating a fixed input-length hash function $\mathcal{H}$).
Input: $s \in \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \in \{0,1\}^{\beta}$, $w \in \{0,1\}^{\alpha-\beta-2}$.

$\mathsf{scratch}(s, z, w)$:

1. Parse s as $(s_1 \| s_2 \| \ldots \| s_n)$ where $|s_l| = \alpha - \beta - 2$ for all $l \in [n]$
2. For $k = 1$ to d:
   (a) For $l = 1$ to n:
      i. **If** $k = 1$ and $l = 1$ **then** compute: $q_l^k = \mathcal{H}(00 \| z \| w)$
      ii. **Else If** $k \neq 1$ and $l = 1$ **then** compute: $q_l^k = \mathcal{H}(10 \| s_n \| q_n^{k-1})$
      iii. **Else If** $l = 2$ **then** compute: $q_l^k = \mathcal{H}(01 \| s_1 \| q_1^k)$
      iv. **Else If** $l > 2$ **then** compute: $q_l^k = \mathcal{H}(01 \| s_{l-1} \| q_{l-1}^k)$
3. Output $q = \mathcal{H}(10 \| s_n \| q_n^d)$.

Fig. 1: Construction of a MPC-hard Function from [30].

We capture this property by requiring the decryption algorithm to output a valid SIP that can be verified by a new SIP-verification algorithm, Vf. We observe that a simple PSIPE notion where the adversary may choose never to output the SIP it has produced is not useful for applications, besides implying the strong notion of extractable witness encryption. Based on this observation, we introduce the notion of a public key Encryption with Public Self-Incriminating Proof (PSIPE), where the decryptor is forced to publish the SIP to successfully decrypt a ciphertext. Notably, this notion requires an underlying public ledger (where the proof is published). We show a PSIPE construction based on witness encryption but also describe how the public ledger, used during setup, allows us to realize the witness encryption needed via techniques from [42].

We observe that the notion of Self-Incriminating Proof Extractability as defined above has limited applicability since the adversary may just choose to erase the SIP that it is forced to produce during decryption. Instead we investigate a variation of this notion that moreover requires the SIP to be published. The notion of Encryption with *Public* Self-Incriminating Proof (PSIPE) is stronger than the former notion but provides a more meaningful guarantee. While it must be defined with respect to a public ledger (for publishing SIPs), this setup also helps realize it without resorting to strong building blocks. We define PSIPE in Definition 6, followed by formal security properties. We construct PSIPE in Figure 17, although we note that this construction is mostly of theoretical interest.

Intuitively, our primitive allows for creating ciphertexts that can only be decrypted by a party if the party has published a SIP on the public ledger. To capture the notion of a public ledger in our security games, we use the model of a PoS blockchain-based public ledger protocol execution introduced in [42] and recalled in 2.2. In this model, algorithms are defined *in the context of a blockchain* Γ^V, meaning that parties executing these algorithms are also part of an execution of an underlying blockchain protocol. This protocol is used to implement the public ledger we require, where parties that execute the protocol can read/write messages. Given such a protocol execution, it is possible to non-interactively verify if a future state $\tilde{\mathsf{B}}$ of the ledger has evolved from an initial state B, which is crucial for our definitions and constructions. This verification is done via the evolving blockchain predicate (defined in Sec. 2.2), *i.e.,* $\mathsf{evolved}(\mathsf{B}, \tilde{\mathsf{B}}) = 1$ iff $\tilde{\mathsf{B}}$ is obtained as a future state of executing the blockchain protocol starting from B.

We believe that a PoS blockchain is the minimal primitive we can use to fulfill our requirements since it exactly affords an incorruptible, public append-only database, whose updates are validated based on an NP-relation. All of these are features we require in our PSIPE solution.

Definition 6 (Encryption with Public Self-Incriminating Proof). *An encryption with public self-incriminating proof scheme* PSIPE *consists of the following* PPT *algorithms* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$ *in the context of a blockchain* Γ^V *with evolved predicate* evolved *(as in Definition 38), which have the following syntax:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$*, the key generation algorithm takes as input a security parameter* 1^λ *and outputs a key-pair* $(\mathsf{pk}, \mathsf{sk})$ *where* pk *is a public key and* sk *is a secret key.*
2. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*, the encryption algorithm takes as input the public key* pk *and a message* $m \in \{0,1\}^\lambda$*, and outputs a ciphertext* c*.*
3. $(\pi, m)/\perp \leftarrow \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$*, the decryption algorithm takes as input the public key* pk*, the secret key* sk *and a ciphertext* c*. It outputs a self-incriminating proof* π *and a plaintext message* m*.*

4. $1/0 \leftarrow \mathsf{Vf}(\mathsf{pk}, c, \pi)$, *the verification algorithm takes as input the public key* pk, *a ciphertext* c *and a self-incriminating proof* π. *It outputs 1 if* π *is a valid self-incriminating proof for* c, *otherwise, it outputs 0.*
5. $\pi/\perp \leftarrow \mathsf{ProofExt}(\mathsf{pk}, c)$, *the self-incriminating proof extraction algorithm takes as input the public key* pk *and a ciphertext* c. *It extracts the self-incriminating proof* π *of the corresponding ciphertext* c *from the underlying blockchain* Γ^V *and outputs* π, *otherwise output* $\perp$.

An encryption with public self-incriminating proof PSIPE *scheme must satisfy the following properties:* ***Correctness*** *(Definition 7),* ***Unforgeability*** *(Definition 8),* ***IND-CPA Security*** *(Definition 9) and* ***Public Self-Incriminating Proof*** *(Definition 10).*

Correctness: The notion of correctness ensures that for a ciphertext correctly generated under a given public key, decryption using the corresponding secret key will always output: (1) a valid self-incriminating proof, and (2) the original plaintext message.

Definition 7 (Correctness). *A scheme* $\mathsf{PSIPE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$ *is correct if for any security parameter* λ *and any message* $m \in \{0,1\}^\lambda$, *the following conditions hold for message decryption correctness and self-incriminating proof correctness, respectively:*

$$\Pr\left[\begin{matrix}\mathsf{Dec}(\mathsf{pk},\mathsf{sk},c)\\ =(\pi,m)\end{matrix}\middle|\begin{matrix}(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{KGen}(1^\lambda)\\ c\leftarrow\mathsf{Enc}(\mathsf{pk},m)\end{matrix}\right]=1 \qquad \Pr\left[\begin{matrix}\mathsf{Vf}(\mathsf{pk},c,\pi)=1\\ \mathsf{Dec}(\mathsf{pk},\mathsf{sk},c)=(\pi,m)\end{matrix}\middle|\begin{matrix}(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{KGen}(1^\lambda)\\ c\leftarrow\mathsf{Enc}(\mathsf{pk},m)\end{matrix}\right]=1$$

Unforgeability: We define a notion of unforgeability for the self-incriminating proofs produced by our primitive which is similar to the notion of existential unforgeability under chosen message attacks for signatures [38]. This type of unforgeability ensures that an adversary should not be able to generate self-incriminating proof for a ciphertext encrypted under a certain public key if they do not possess the corresponding secret key. This holds even if the adversary is the one generating the ciphertext, which is important to avoid falsely incriminating a decryptor. We formally define this notion in Definition 8 via a game $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}}$, which is presented in Figure 2.

$\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}}$

Game steps:

1. Initialize an empty list $\mathcal{Q} \leftarrow \emptyset$
2. $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
3. $(c',\pi') \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}}}(1^\lambda,\mathsf{pk})$

Oracle:

$\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}}(c)$:
1. $\mathcal{Q} \leftarrow \mathcal{Q} \cup c$
2. $(\pi, m) \leftarrow \mathsf{Dec}(\mathsf{pk},\mathsf{sk},c)$
3. **return** (π, m)

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}} = \Pr\left[\mathsf{Vf}(\mathsf{pk},c',\pi')=1 \wedge c' \notin \mathcal{Q}\right]$$

Fig. 2: Unforgeability Game for PSIPE scheme executed between a challenger and an adversary $\mathcal{A}$ given unlimited oracle access to $\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}}(c)$.

Definition 8 (Unforgeability). *A scheme* $\mathsf{PSIPE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$ *is unforgeable if for any security parameter* λ *and for all* PPT *adversaries* $\mathcal{A}$ *given unlimited oracle access to* $\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}}(\cdot)$, *advantage* $\mathsf{Adv}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}}$ *of the* $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}}$ *(in Figure 2) is negligible.*

IND-CPA Security: We define IND-CPA Security for Encryption with Self-Incriminating Proof in the standard manner. This notion is formally defined in Definition 9 via a game $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}}$, which is presented in Figure 3.

Definition 9 (IND-CPA Security). *A scheme* $\mathsf{PSIPE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$ *is IND-CPA secure if for any security parameter* λ *and for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, *advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}}$ *of* $\mathcal{A}$ *in* $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}}$ *(in Figure 3) is negligible.*

$\mathsf{Game}^{\text{IND-CPA}}_{\mathsf{PSIPE},\mathcal{A}}$

Game steps:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
2. $(\mathsf{st}, m_0, m_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pk})$
3. Sample $b \xleftarrow{\$} \{0,1\}$
4. $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
5. $b' \leftarrow \mathcal{A}_2(1^\lambda, \mathsf{st}, \mathsf{pk}, c_b)$
6. **return** b'

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{PSIPE},\mathcal{A}} = \left| \Pr\left[b = b'\right] - \frac{1}{2} \right|$$

Fig. 3: IND-CPA Security Game for PSIPE scheme.

Public Self-Incriminating Proof. We formalize the public self-incriminating proof property in the context of a blockchain Γ in Definition 10 via a game $\mathsf{Game}^{\text{PUB-SIP}}_{\mathsf{PSIPE},\mathcal{A},\Gamma}$, presented in Figure 4. For this notion, we need to ensure that the key pair has been generated correctly, rather than allowing the adversary to generate an arbitrary key pair. In order to capture this requirement in the simplest way possible, the challenger generates the key pair and gives it to the adversary. This can be captured by requiring key registration, *i.e.,* each party must publish their public key on the public ledger together with a zero-knowledge proof that the key has been generated by the correct key generation algorithm.

$\mathsf{Game}^{\text{PUB-SIP}}_{\mathsf{PSIPE},\mathcal{A},\Gamma}$

Game steps:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$; // Guarantee that the keys are generated correctly.
2. $\mathsf{VIEW} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk}), \mathcal{Z}, 1^\lambda)$; // Execute the blockchain protocol
3. $(\mathsf{st}_1, m_0, m_1) \leftarrow \mathcal{A}_2(1^\lambda, \mathsf{pk}, \mathsf{sk}, \mathsf{VIEW}_{\mathcal{A}_1})$; // Get adversary's view $\mathsf{VIEW}_{\mathcal{A}_1}$ from VIEW
4. Samples $b \xleftarrow{\$} \{0,1\}$
5. $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
6. $\mathsf{st}_2 \leftarrow \mathcal{A}_3(\mathsf{st}_1, c_b)$; // Allow the adversary to compute on c_b
7. $\mathsf{VIEW}' \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}_4(\mathsf{st}_2), \mathcal{Z}, 1^\lambda)$; // Execute the blockchain protocol
8. $\mathsf{VIEW}'_{\mathcal{A}_4} \leftarrow \mathsf{VIEW}'$; // Get adversary's view $\mathsf{VIEW}'_{\mathcal{A}_4}$ from VIEW'
9. $b' \leftarrow \mathcal{A}_5(\mathsf{st}_2, \mathsf{VIEW}'_{\mathcal{A}_4})$

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\text{PUB-SIP}}_{\mathsf{PSIPE},\mathcal{A},\Gamma} = \left| \Pr\left[b = b'\right] - \frac{1}{2} \right|$$

The extractor EXT's success probability in this game is:

$$\mathsf{Succ}^{\mathsf{PSIPE}}_{\mathsf{EXT},i} = \left| \Pr\left[\pi \leftarrow \mathsf{EXT}(\mathsf{pk}, c_b, \tilde{\mathsf{B}}_i) \mid \mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1\right] \right|$$

Where $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\tilde{\mathsf{B}}_i \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$ are executed on honest party P_i's view VIEW'_{P_i} obtained from VIEW'.

Fig. 4: Public Self-Incriminating Proof game for PSIPE scheme.

Definition 10 (Public Self-Incriminating Proof). *A scheme* PSIPE *in the context of a blockchain protocol* Γ *executed by* PPT *machines* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5)$ *and* $\mathcal{Z}$ *has the public self-incriminating proof property if for any security parameter* λ*, any polynomial* $p(\cdot)$*, any* $\mathcal{Z}$ *and any* $\mathcal{A}$ *with advantage* $\mathsf{Adv}^{\text{PUB-SIP}}_{\mathsf{PSIPE},\mathcal{A},\Gamma}$ *in* $\mathsf{Game}^{\text{PUB-SIP}}_{\mathsf{PSIPE},\mathcal{A},\Gamma}$*, there exists a polynomial* $q(\cdot)$ *and a* PPT *extractor* EXT *that outputs* $\pi \leftarrow \mathsf{EXT}(\mathsf{pk}, c_b, \tilde{\mathsf{B}}_i)$ *such that* $\mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1$ *for* $\tilde{\mathsf{B}}_i \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$ *obtained by any honest party* P_i *from* $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(1^\lambda)$ *executed on its* $\mathsf{VIEW}'_{P_i} \in \mathsf{VIEW}'$ *with success probability* $\mathsf{Succ}^{\mathsf{PSIPE}}_{\mathsf{EXT},i}$ *where*

$$\mathsf{Adv}^{\text{PUB-SIP}}_{\mathsf{PSIPE},\mathcal{A},\Gamma} \geq \frac{1}{p(\lambda)} \Rightarrow \mathsf{Succ}^{\mathsf{PSIPE}}_{\mathsf{EXT},i} \geq \frac{1}{q(\lambda)}$$

Remark 1 (On the requirement of Proof-of-Stake for PSIPE*).* The PSIPE definition requires a notion of consensus that allows for third parties to non-interactively verify that a message is agreed upon without participating in a protocol execution. This is the case because the decryption algorithm must make sure that an adversary has indeed published a SIP as part of the decryption process. However, that

step in decryption is inherently non-interactive, precluding the use of an ideal bulletin board, which requires interaction in order to verify that a message has been published. While non-interactive proofs that a message has been agreed upon can be obtained for classical byzantine agreement protocols, their guarantees under semi-synchrony or asynchrony are limited by fundamental impossibility results and their scalability is limited by round/communication complexity lower bounds. Hence, we base our definition on the weaker notion of blockchain-based consensus, which both circumvents such impossibility results over semi-synchronous networks and allow for large scale executions of the consensus protocol. We specifically base our definition on Proof-of-Stake (PoS) blockchains because in a Proof-of-Work (PoW) blockchain the adversary can always simulate a chain where it generates all blocks. Notice, however, that while the PoS blockchain model matches this requirement, it can also be obtained by alternative consensus protocols with similar non-interactive proofs of agreement, *e.g.,* HotStuff [54].

IND-CPA Security vs. Public Self-Incriminating Proofs. Notice that we do not formally require that a published SIP for a decrypted ciphertext c leaks no information about the message contained in c (*i.e.,* that IND-CPA Security holds even when the adversary has access to a SIP published when decrypting c). While this is a desirable property for many applications, we aim at defining a notion of PSIPE that is as general as possible, which includes potentially allowing for publishing a SIP that does reveal information about plaintext messages in decrypted ciphertexts. We hope that presenting such a definition will allow for obtaining potentially more efficient constructions based on weaker assumptions and simpler building blocks. Notice, however, that the PSIPE construction presented in this section does guarantee that a SIP leaks no information about the plaintext message in decrypted ciphertexts, since a SIP in this construction is simply a signature on a random string chosen independently from the message.

3.1 Practical PSIPE based on Threshold Batched IBE.

We propose two constructions $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ and $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ based on Threshold Batched IBE (TBIBE) and Extractable Witness Encryption (eWE) respectively. The former ($\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$) represents a practical protocol, but requires modifying a secure blockchain protocol that assumes the election of an honest majority validator committee in every round to perform consensus. The latter ($\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$) assumes nothing about the secure blockchain protocol, but is based on an extractable Witness Encryption scheme; concretely, the encryptor in $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ encrypts to a specific relation, such that decryption succeeds with a witness that is a valid blockchain transcript with a valid signature by the decryptor on the ciphertext instance finalized in the common-prefix.

We describe Protocol $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ in Fig. 5 and give an overview below:

$\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ Encryption. A well-formed $\mathsf{PSIPE\text{-}TBIBE}$ cipher-text consists of two layers of encryption.

1. The inner layer is a public key encryption of the plaintext towards the public key of the decryptor.
2. The outer layer is a TBIBE encryption towards a TBIBE public key.

The ciphertext instance is then defined as the $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$, where $\hat{\mathsf{c}}$ is the encryption ciphertext resulting from encryption steps (1) and (2), and id a freshly sampled TBIBE identity.

$\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ Decryption. The decryptor starts decryption by broadcasting a SIP and ciphertext $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$ to the blockchain. A SIP is simply a signature on $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$. For each block that is finalized, we require the blockchain validators to broadcast TBIBE decryption keys for all well-formed ciphertext id's in the block. Recall that in TBIBE, this can be performed in a batched manner; a single set of TBIBE keys will enable the decryption of each ciphertext in the entire block, restricting the additional message complexity imposed on the blockchain protocol by $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ to be only *linear* in the size of the validator committee size, and independent of the number of SIP's in each block.

$\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ Proof Extraction & Verification. Proof extraction for $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$ and decryptor public key pk simply involves parsing the blockchain for the appropriate SIP, *i.e.* a signature on ct that verifies under pk. Verification of a SIP from the decryptor with public key pk and ciphertext $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$ is also simply a signature verification on message ct and pk.

Blockchain protocol extension in $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$. We discuss the overhead of the required extension of the underlying blockchain protocol. As formalized in our scheme in Fig. 5, each elected validator committee must parse and verify newly finalized SIP messages (or signatures on $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$). Let $\mathcal{I}$ represents the set of id's corresponding to the batch of newly finalized valid SIP's. The committee of size n then posts n partial TBIBE decryption signatures which can be aggregegated ($\mathsf{TBIBE.ComputeKeyAggregate}$) to produce a key which that decrypts any TBIBE ciphertext encrypted to $\mathsf{id} \in \mathcal{I}$. Notice, that the blockchain

message complexity induced by this protocol extension is strictly linear in the size of the validator committee and independent of the batch size.

The overhead induced on the underlying blockchain protocol is related to frequency with which new validator committees are elected; each fresh set of validator parties requires the *resharing* of TBIBE master keys, inducing a $O(n^2)$ communication overhead. For a blockchain execution with committees that include thousands of validators which are frequently refreshed, this overhead may be prohibitive in practice. In contrast, for a blockchain protocol with smaller committees and infrequent committee elections or even static committee role assignments (permissioned blockchains), this overhead can be minimal.

Security of $\Pi_{\mathsf{PSIPE\text{-}TBIBE}}$. We first sketch arguments for unforgeability, IND-CPA and PSIPE security. It is easy to see that forgery of a SIP implies forging the signature scheme SIG. IND-CPA security follows from IND-CPA of encryption scheme PKE - distinguishing between PSIPE ciphertexts is reduced to distinguishing between the inner PKE encryptions. Finally, contradicting the public self incrimination property implies either (1) breaking common-prefix blockchain property or (2) security of TBIBE. (1) permits the adversary to remove a previously finalized SIP from the blockchain (causing proof extraction to fail) and (2) implies that the adversary can distinguish between ciphertexts without TBIBE decryption keys, permitting the adversary to forgo broadcasting a SIP, again causing proof extraction to fail. We formally capture the security of Protocol $\Pi_{\mathsf{PSIPE\text{-}TBIBE}}$ in Theorem 1 and defer the proof to Appendix C.1.

Construction of PSIPE Scheme: $\Pi_{\mathsf{PSIPE\text{-}TBIBE}}$

Parameters: A security parameter λ.

Building-blocks: A secure PoS blockchain protocol $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ (Sec. 2.2) satisfying the κ-common-prefix security (Def. 3). A IND-CPA secure public key encryption scheme $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$. An EUF-CMA secure signature scheme $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$. A Thresholdizable Batched IBE scheme $\mathsf{TBIBE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Digest}, \mathsf{ComputeKeyShare}, \mathsf{ComputeKeyAggregate}, \mathsf{Dec})$ (Definition 1).

- $\mathsf{KGen}(1^\lambda)$:
  1. Outputs $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KGen}(\lambda)$ and $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KGen}(\lambda)$.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$
  1. Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{pk}_{\mathsf{TBIBE}})$.
  2. Encrypt $\mathsf{c} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{msg})$.
  3. Sample $\mathsf{id} \leftarrow^{\$} \mathcal{I}$ and encrypt $\hat{\mathsf{c}} \leftarrow \mathsf{TBIBE.Enc}(\mathsf{pk}_{\mathsf{TBIBE}}, \mathsf{c}, \mathsf{id})$
  4. Output $\mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id})$.
- $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{ct})$
  1. Parse $\mathsf{pk} = (\mathsf{pk}_{\mathsf{SIG},}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]})$, $\mathsf{sk} = (\mathsf{sk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{PKE}})$.
  2. Post self-incriminating proof of decrypting ct:
     (a) Compute $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_{\mathsf{SIG}}, \mathsf{ct})$.
     (b) Run $\mathsf{Broadcast}(1^\lambda, (\mathsf{pk}_{\mathsf{SIG}}, \pi, \mathsf{ct}))$ in current blockchain round r.
  3. After blockchain round $r + \kappa$, retrieve decryption keys to decrypt message:
     (a) Run $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(\lambda)$ and $\tilde{\mathsf{B}} \leftarrow \mathsf{GetRecords}(\lambda, \tilde{\mathsf{st}})$.
     (b) For $(\mathsf{pk}_{\mathsf{SIG}}, \pi, \mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id}))$ in block r of $\tilde{\mathsf{B}}$: $\mathcal{J} \leftarrow \mathcal{J} \cup \{\mathsf{id}\}$ if $1 \leftarrow \mathsf{SIG.Vf}(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{ct}, \pi)$.
     (c) Compute $\mathsf{d} \leftarrow \mathsf{TBIBE.Digest}(\mathsf{pk}_{\mathsf{TBIBE}}, \mathcal{J})$. Parse all $(\mathsf{sk}^{\mathsf{d}}_{\mathsf{TBIBE},i}, \mathsf{d}_i)$ in block $r + \kappa$ of $\tilde{\mathsf{B}}$.
     (d) Compute $\mathsf{sk}^{\mathsf{d}}_{\mathsf{TBIBE}} \leftarrow \mathsf{TBIBE.ComputeKeyAggregate}(\{\mathsf{pk}_{\mathsf{TBIBE},i}, \mathsf{sk}^{\mathsf{d}}_{\mathsf{TBIBE},i}\}_{i\in[n]}, \mathsf{d})$.
     (e) Decrypt $\mathsf{c} \leftarrow \mathsf{TBIBE.Dec}(\hat{\mathsf{c}}, \mathsf{sk}^{\mathsf{d}}_{\mathsf{TBIBE}}, \mathsf{d}, \mathcal{J}, \mathsf{id})$.
     (f) Output $\mathsf{msg} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{c})$.
- $\mathsf{Vf}(\mathsf{pk}, \mathsf{ct} = (\hat{\mathsf{c}}, \mathsf{id}), \pi)$:
  1. Output $\mathsf{SIG.Vf}(\mathsf{pk}, \mathsf{ct}, \pi)$.
- $\mathsf{ProofExt}(\mathsf{pk}, \mathsf{ct})$:
  1. Run $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\tilde{\mathsf{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$ and parse for all $(\mathsf{pk}, \mathsf{ct}, \pi)$ in $\tilde{\mathsf{B}}$.
  2. Output π in first message $(\mathsf{pk}, \mathsf{ct}, \pi)$ for which $\mathsf{SIG.Vf}(\mathsf{pk}, \mathsf{ct}, \pi) = 1$ or $\perp$ otherwise.

Blockchain protocol Γ': Extends blockchain protocol Γ as follows.

- $\mathsf{Setup}(1^\lambda, 1^B, 1^n)$: At the beginning of Γ', the following setup procedure is run.
  1. $\mathsf{pp} \leftarrow \mathsf{TIBE.Setup}(1^\lambda, 1^B, 1^n)$
  2. $(\mathsf{pk}_{\mathsf{TBIBE}}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]}, \{\mathsf{msk}_{\mathsf{TBIBE},i}\}_{i\in[n]}) \leftarrow \mathsf{TBIBE.KGen}(\mathsf{pp})$
  3. Output $\mathsf{msk}_{\mathsf{TBIBE},i}$ to party P_i and $\mathsf{pk}_{\mathsf{TBIBE}}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]}$ to all parties.
- At the beginning of each round r of Γ', each party executes the following.
  1. $\mathsf{st} \leftarrow \mathsf{UpdateState}(\lambda)$ & $\tilde{\mathsf{B}} \leftarrow \mathsf{GetRecords}(\lambda, \mathsf{st})$
  2. Parse $\tilde{\mathsf{B}}$ for all $(\mathsf{pk}_{\mathsf{SIG},j}, \pi_j, \mathsf{ct}_j)$ messages posted in round $r - \kappa$.
  3. For each $(\mathsf{pk}_{\mathsf{SIG},j}, \pi_j, \mathsf{ct}_j)$ such that $\mathsf{SIG.Vf}(\mathsf{pk}_{\mathsf{SIG},j}, \pi_j, \mathsf{ct}_j) = 1$, $\mathcal{J} \leftarrow \mathcal{J} \cup \{\mathsf{id}_j\}$.
  4. Compute $\mathsf{d} \leftarrow \mathsf{TBIBE.Digest}(\mathsf{pk}_{\mathsf{TBIBE}}, \mathcal{J})$.
  5. Compute $\mathsf{sk}^{\mathsf{d}}_{\mathsf{TBIBE},i} \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_{\mathsf{TBIBE},i}, \mathsf{d})$.
  6. $\mathsf{Broadcast}(1^\lambda, (\mathsf{sk}^{\mathsf{d}}_{\mathsf{TBIBE},i}, \mathsf{d}))$.
  7. Reshare $\mathsf{msk}_{\mathsf{TBIBE},i}$ to the respective validator role in the next-round committee.

Fig. 5: $\Pi_{\mathsf{PSIPE\text{-}TBIBE}}$: Construction of PSIPE based on TBIBE.

Theorem 1. *Assuming that: (i) Γ is a secure blockchain protocol (Def. 3,4,5) (ii)* SIG *is a EUF-CMA secure signature scheme as per Definition 16. (iii)* PKE *is a IND-CPA secure signature scheme as per Definition 16. (iv)* TBIBE *is a secure (Fig. 16) thresholdizable batched IBE scheme (as in Definition 1), then our protocol* $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ *in Figure 5 is a secure encryption with public self-incriminating proof scheme* PSIPE *as per Definition 6.*

We refer to Appendix B for the complete description, and security treatment, of our alternative PSIPE protocol $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$, which realizes PSIPE with extractable Witness Encryption and without assuming any modifications of the underlying blockchain protocol. $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ requires a stricter formalism of blockchains from [41] defined in Appendix A.8 with distinguishable forking property (Def. 37) and evolved predicate (Def. 38).

4 Threshold Encryption with Self-Incriminating Proof

In this section, we introduce a novel primitive called "threshold encryption with self-incriminating proof" (TSIPE). As in the standard public key encryption case, any set of $t+1$ parties can perform an out-of-band attack to decrypt a ciphertext without being detected. However, since multiple parties need to cooperate in order to perform decryption, we do not necessarily need to force the SIP to be published. Instead, our notion of TSIPE guarantees that one of the parties involved in decryption learns a SIP; this party can then choose to leak the proof at any time. As discussed in the introduction this can be incentivized e.g. through an smart contract logic on a tokenized Turing complete public ledger, but this could also be motivated through more traditional means, *e.g.*, the court system.

4.1 Formal Syntax and Security Definitions

Definition 11 (Threshold Encryption with Self-Incriminating Proof). *A threshold encryption with self-incriminating proof scheme* TSIPE *consists of the following probabilistic polynomial-time* (PPT) *algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$ *with the following syntax:*

1. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$, *the key generation algorithm is executed by a trusted third party. It takes as input the security parameter λ, the number of parties n, and the threshold t and outputs $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$ where pk is the public key and sk_i is threshold decryption key share for party P_i. The trusted third party distributes sk_i to each party P_i before execution starts.*
2. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, *the encryption algorithm takes as input the public key pk and a message $m \in \{0,1\}^\lambda$. It outputs a ciphertext c.*
3. $\nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, c)$, *the partial decryption algorithm takes as input a secret key share sk_i and a ciphertext c, outputting a partial decryption ν_i.*
4. $(m, \pi)/\perp \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_i, c, \{\nu_i\}_{i\in T})$, *the partial decryption combining algorithm that takes as input the public key pk, a secret key share sk_i and the partial decryption $\{\nu_i\}_{i\in T}$ for a set T where $|T| \geq t+1$. It outputs a message m and a self-incriminating proof π, otherwise, it outputs $\perp$.*
5. $1/0 \leftarrow \mathsf{Vf}(\mathsf{pk}, c, \pi)$, *the verification algorithm takes as input the public key pk, the ciphertext c, and the self-incriminating proof π. It outputs 1 if π is a valid self-incriminating proof, otherwise outputs 0.*

A TSIPE *scheme must satisfy the following properties:* ***Correctness*** *(Definition 12),* ***Unforgeability*** *(Definition 13),* ***IND-CPA Security*** *(Definition 14), and* ***Self-Incriminating Proof Extractability*** *(Definition 15).*

Correctness The notion of correctness ensures that: (1) a set of at least t honestly generated partial decryptions always produce the original message and a correct self-incriminating proof, and (2) an honestly generated self-incriminating proof always verifies.

Definition 12 (Correctness). *A threshold encryption scheme self-incriminating proof scheme* $\mathsf{TSIPE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$ *is correct if for any security parameter λ, any $n, t \in \mathbb{N}$ where $0 < t < n$ and any message $m \in \{0,1\}^\lambda$, we have the following two following properties:*

- ***Combined decryption correctness:*** *for any $T \subseteq \{\mathsf{sk}_i\}_{i\in[n]}$ with $|T| \geq t+1$ and $i \in T$ and any message $m \in \{0,1\}^\lambda$ we have that,*

$$\Pr\left[\mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_i, c, \{\nu_i\}_{i\in T}) = (m, \pi) \;\middle|\; \begin{array}{r}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t) \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ \forall i \in T : \nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, c)\end{array}\right] = 1$$

- ***Self-incriminating proof correctness:*** *for any* $T \subseteq \{\mathsf{sk}_i\}_{i\in[n]}$ *with* $|T| \geq t+1$ *and* $i \in T$ *and any message* $m \in \{0,1\}^\lambda$ *we have that,*

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(\mathsf{pk}, c, \pi) = 1 \\ \mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_i, c, \{\nu_i\}_{i\in T}) = (m, \pi)\end{array}\middle|\begin{array}{r}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t) \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ \forall i \in T : \nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, c)\end{array}\right] = 1$$

Unforgeability. The notion of unforgeability ensures that an adversary should not be able to forge self-incriminating proofs for a ciphertext. In the threshold case, our notion of unforgeability holds only in the case where the party generating a ciphertext is not part of the decryption committee, *i.e.,* when the forger does not possess a secret key share. While we do not guarantee unforgeability against a member of the decryption committee, we argue that this notion is sufficient for many applications where threshold decryption is provided as a service for ciphertexts given as input by clients who are not in the decryption committee. In this case, a decryption committee member generating a "forged" SIP for an arbitrary ciphertext that they generate locally does not imply misbehavior.

We wish to guarantee security against two different attacks: 1. An adversary who is not part of the decryption committee and tries to forge a SIP against the decryption committee; 2. A subset T of the decryption committee with $|T| \leq t$ (*i.e.,* without the power to decrypt) who tries to forge a SIP against the decryption committee. The first scenario is captured by a variation of the game for Unforgeability against a monolithic adversary with has access to the public key and a decryption oracle that we have defined for the standard public key encryption scenario. In the context of threshold schemes, we define game $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$, presented in Figure 6, where a monolithic adversary has access to the public key and to a decryption oracle that will decrypt any ciphertext under any sufficiently large set of secret key shares, generating a SIP under any secret key share. The second scenario is formalized in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$, presented in Figure 7. In this game, a subset of the decryption committee has access to an oracle that generates decryption shares and another oracle that generates ciphertexts encrypting arbitrary messages while keeping the encryption randomness secret. This captures the guarantee that subsets of the decryption committee are not able to forge a SIP for a ciphertext that was generated by a third party but not yet decrypted.

Requiring that the adversaries have a negligible advantage in both aforementioned games captures the fact that we guarantee unforgeability only against an adversary who does not collude with the decryption committee (or is part of the decryption committee itself). We formalize self-incriminating proof unforgeability in Definition 13 via games $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$ and $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ presented in Figures 6 and 7, respectively.

$\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$

Game steps:

1. Initialize an empty list $\mathcal{Q} \leftarrow \emptyset$
2. The adversary $\mathcal{A}$ picks n and t.
3. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
4. $(c', \pi') \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}_{i\in[n]}}}(1^\lambda, \mathsf{pk})$

Oracle:

$\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}_{i\in[n]}}(j, T, c)$:

1. $\mathcal{Q} \leftarrow \mathcal{Q} \cup c$
2. For $i \in T$, $\nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, c)$
3. $(\pi, m) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_j, c, \{\nu_i\}_{i\in T})$
4. **return** (π, m)

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}} = \Pr\left[\mathsf{Vf}(\mathsf{pk}, c', \pi') = 1 \wedge c' \notin \mathcal{Q}\right]$$

Fig. 6: Self-Incriminating Proof Unforgeability Game for TSIPE with a monolithic adversary ($\mathcal{A}$) given pk and oracle access to $\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}_{i\in[n]}}(j, T, c)$.

Definition 13 (Unforgeability). *A threshold encryption scheme self-incriminating proof scheme* $\mathsf{TSIPE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$ *is unforgeable if for any security parameter* λ*, for any* $n, t \in \mathbb{N}$ *where* $0 < t < n$*, for all* PPT *monolithic adversaries* $\mathcal{A}$ *and for all* PPT (δ, Υ)*-distributed adversaries* $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$*, the advantage* $\mathsf{Adv}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$ *of* $\mathcal{A}$ *in* $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$ *(Figure 6) and the advantage* $\mathsf{Adv}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}}$ *of* $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ *in* $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ *(Figure 7) are negligible.*

$\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$

Game steps:

1. Initialize an empty list $\mathcal{Q} \leftarrow \emptyset$
2. The (δ, Υ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ picks n and t.
3. $\mathcal{A}_1, \ldots, \mathcal{A}_a$ chooses a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| = a \leq t$.
4. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
5. The sub-adversaries $\mathcal{A}_1^{\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_{i\in[n]\setminus\tilde{T}}}, \mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}}(1^\lambda, \mathsf{sk}_1), \ldots, \mathcal{A}_a^{\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_{i\in[n]\setminus\tilde{T}}}, \mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}}(1^\lambda, \mathsf{sk}_a)$ engage in an interactive protocol with access to oracles $\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_{i\in[n]\setminus\tilde{T}}}$ and $\mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}$, at the end of which, $\mathcal{A}_1$ outputs (c', π').

Oracles:

$\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_{i\in[n]\setminus\tilde{T}}}(i, c)$:

1. $\mathcal{Q} \leftarrow \mathcal{Q} \cup c$
2. $\nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, c)$
3. **return** ν_i

$\mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}(m)$:

1. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$
2. **return** c

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}} = \Pr\left[\mathsf{Vf}(\mathsf{pk}, c', \pi') = 1 \wedge c' \notin \mathcal{Q}\right]$$

Fig. 7: Self-Incriminating Proof Unforgeability Game for TSIPE with a (δ, Υ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ given oracle access to $\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_{i\in[n]\setminus\tilde{T}}}(i, c)$ and $\mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}(m)$.

IND-CPA Security: In Definition 14, we formalize IND-CPA Security for TSIPE in the usual manner via a game $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ between a challenger and a (δ, Υ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$, presented in Figure 8.

$\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$

Game steps:

1. The (δ, Υ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ picks n and t.
2. $\mathcal{A}_1, \ldots, \mathcal{A}_a$ chooses a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| = a \leq t$.
3. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
4. The sub-adversaries $\mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk}_1), \ldots, \mathcal{A}_a(1^\lambda, \mathsf{pk}, \mathsf{sk}_a)$ engage in an interactive protocol, at the end of which, $\mathcal{A}_1$ outputs (st, m_0, m_1).
5. Sample $b \xleftarrow{\$} \{0, 1\}$
6. Compute $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
7. The sub-adversaries $\mathcal{A}_1\left(1^\lambda, \mathsf{st}, \mathsf{pk}, \mathsf{sk}_1, c_b\right), \ldots, \mathcal{A}_a\left(1^\lambda, \mathsf{st}, \mathsf{pk}, \mathsf{sk}_a, c_b\right)$ engage in an interactive protocol, at the end of which, $\mathcal{A}_1$ outputs b'.
8. **return** b'

As there are only two outcomes, there is a $\frac{1}{2}$ probability that a random guess will be correct, the adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a} = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right|$$

Fig. 8: IND-CPA Security Game for TSIPE scheme.

Definition 14 (IND-CPA Security). *A threshold encryption scheme self-incriminating proof scheme* $\mathsf{TSIPE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$ *is IND-CPA secure if for any* $n, t \in \mathbb{N}$ *where* $0 < t < n$, *and for all* PPT (δ, Υ)*-distributed adversary* $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ *where* $|a| \leq t$, *the advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ *of* $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ *in* $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ *(in Figure 8) is negligible.*

Self-Incriminating Proof Extractability. We define self-incriminating proof extractability for Threshold Encryption with Self-Incriminating Proof, via a game $\mathsf{Game}^{\mathsf{SIP\text{-}SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ between a challenger and (δ, Υ)-distributed adversary $\mathcal{A}_1, \ldots, \mathcal{A}_a$. The game is presented in Figure 9.

Observe that unlike the work of Dziembowski *et al.* [30] described in Sec. 2.3 and App. A.9, we require an extractor kEXT_i to not only take as input the transcript of fast calls to $\mathcal{O}_{\mathcal{H}}$ for a party holding sk_i, but also the public setup parameters along with the secret key share sk_i itself. While on the surface this

$\mathsf{Game}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$

Game steps:

1. The (δ, Υ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ picks n and t.
2. $\mathcal{A}_1, \ldots, \mathcal{A}_a$ chooses a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| = a \geq t+1$.
3. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
4. The sub-adversaries $\mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk}_1), \ldots, \mathcal{A}_a(1^\lambda, \mathsf{pk}, \mathsf{sk}_a)$ engage in an interactive protocol with δ-bounded access to oracle $\mathcal{O}_\mathcal{H}$, at the end of which, $\mathcal{A}_1$ outputs (st, m_0, m_1).
5. Sample $b \xleftarrow{\$} \{0,1\}$
6. Compute $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
7. The sub-adversaries $\mathcal{A}_1\left(1^\lambda, \mathsf{st}, \mathsf{pk}, \mathsf{sk}_1, c_b\right), \ldots, \mathcal{A}_a\left(1^\lambda, \mathsf{st}, \mathsf{pk}, \mathsf{sk}_a, c_b\right)$ engage in an interactive protocol with δ-bounded access to oracle $\mathcal{O}_\mathcal{H}$, at the end of which, $\mathcal{A}_1$ outputs, b'.
8. **return** b'

As there are only two outcomes, there is a $\frac{1}{2}$ probability that a random guess will be correct, the adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a} = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right|$$

Define extractor $\pi_i \leftarrow \mathsf{kEXT}_i(\mathsf{pp}, c, \mathsf{sk}_i, \tau_i^{\mathsf{fast}})$ for $i \in [a]$ taking as input the public Setup values $\mathsf{pp} = (\lambda, n, t, \mathsf{pk})$, the ciphertext c, secret key share sk_i, transcript τ_i^{fast} of fast queries to oracle $\mathcal{O}_\mathcal{H}$ from sub-adversary $\mathcal{A}_i$, and outputs a SIP π_i with the following success probability:

$$\mathsf{Succ}^{\mathsf{TSIPE}}_{\mathsf{kEXT}} = \Pr\left[\exists i \in [a] \text{ s.t. } \pi_i \leftarrow \mathsf{kEXT}_i(\mathsf{pp}, c, \mathsf{sk}_i, \tau_i^{\mathsf{fast}}) \wedge \mathsf{Vf}(\mathsf{pk}, c_b, \pi_i) = 1\right]$$

Fig. 9: Self-Incriminating Proof Security Game for TSIPE scheme.

might seem like cheating, we point out that this does *not* make the extractor trivial. In particular, at least $t+1$ secret key shares should be required by any reasonable protocol in order to allow decryption. Thus, since each extractor is individual and unable to communicate with the other extractors, we are not giving it more power than any minimal party participating in the protocol. Furthermore, the notion is closely related to the idea of a knowledge extractor for soundness in zero-knowledge proofs. In practice, it is reasonable to assume that the Setup procedure is either carried out by a trusted third party or a distributed key generation protocol. Hence, parties will only be convinced of the correctness of the setup if there is a maliciously secure interactive protocol that has executed the setup, which will imply that the secret key shares can be extracted at the moment of setup.

Definition 15 (Self-Incriminating Proof Extractability). *A threshold encryption scheme self-incriminating proof scheme* $\mathsf{TSIPE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$ *has self-incriminating proof extractability if for any* λ *and* $n, t \in \mathbb{N}$ *where* $0 < t < n$*, there exist knowledge extractors* $\mathsf{kEXT}_1, \ldots, \mathsf{kEXT}_a$ *such that for every* PPT (δ, Υ)*-distributed adversary* $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ *where* $|a| \geq t+1$ *we have that,*

$$\mathsf{Succ}^{\mathsf{TSIPE}}_{\mathsf{kEXT}} \geq \mathsf{Adv}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a} - \mathsf{negl}(\lambda)$$

where $\mathsf{Adv}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ *is the advantage of* $\mathcal{A}_1, \ldots, \mathcal{A}_a$ *and* $\mathsf{Succ}^{\mathsf{TSIPE}}_{\mathsf{kEXT}}$ *is the extractors' success probability in the* $\mathsf{Game}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ *defined in Figure 9 and* τ_j^{fast} *is a transcript of the queries that the sub-adversary* $\mathcal{A}_j$ *has made to the oracle* $\mathcal{O}_\mathcal{H}$ *only in* $\mathtt{mode} = \mathsf{fast}$ *(defined in Sec. 2.3).*

4.2 Construction of TSIPE

In this section, we present a concrete construction of our threshold encryption with self-incriminating $\mathsf{TSIPE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$, which we call $\mathbf{\Pi}_{\mathsf{TSIPE}}$. The formal construction of our scheme $\mathbf{\Pi}_{\mathsf{TSIPE}}$ is described in Figure 10.

Overview of $\mathbf{\Pi}_{\mathsf{TSIPE}}$. We show a TSIPE construction, starting from a regular threshold encryption scheme and embedding the computation of an MPC-hard function in the threshold decryption process, using the input to this MPC-hard function to generate a SIP. The rationale is that we prevent the sub-adversaries from executing the threshold decryption process within MPC in order to obtain the message while discarding the SIP, which they cannot do as at least one sub-adversary must learn the input in order to evaluate the MPC-hard function. A brief sketch of our protocol $\mathbf{\Pi}_{\mathsf{TSIPE}}$ is described below:

To encrypt a message m, it samples two random $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^\beta$ and then search for β number of nonces $w_1, \ldots, w_\beta \in \{0,1\}^{\alpha-\beta-2}$ by computing $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ such that the first ζ bits of q_i are zero, for all $i \in [1, \beta]$ (as described in Step 2 in Figure 10), and we use its outputs to derive

Threshold Encryption with Self-Incriminating Proof Scheme: Π_{TSIPE}

Parameters: Security parameter λ, number of parties n and threshold t such that $0 < t < n$, value $\alpha, \beta, \zeta \in \mathbb{N}$ with $\alpha \geq 2\beta$ (and ζ can be a function of β).

Building-blocks: A threshold encryption scheme $\mathsf{TE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine})$, a commitment scheme $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Com})$, an MPC-hard function $\mathsf{scratch}$, a $\mathsf{NIZKPoK}$ proof system, and random oracles $\mathcal{H}_1 : \{0,1\}^{(n+\beta)\cdot(\alpha-\beta-2)+\beta^2+\beta} \to \{0,1\}^{\lambda}$, $\mathcal{H}_2 : \{0,1\}^{(n+\beta)\cdot(\alpha-\beta-2)+\beta^2+\beta} \to \{0,1\}^{\lambda}$, $\mathcal{H}_3 : \{0,1\}^{*} \to \{0,1\}^{\alpha-\beta-2}$,$\mathcal{H}_4 : \{0,1\}^{*} \to \{0,1\}^{\lambda}$. A pseudorandom function $\mathsf{PRF} : \{0,1\}^{(n+\beta)\cdot(\alpha-\beta-2)+\beta^2+\beta} \to \{0,1\}^{\lambda}$.

- **Setup Phase:** A trusted third party executes the $\mathsf{Setup}(1^{\lambda}, n, t)$ algorithm as follows:
  1. Run $\left(\mathsf{pk}_{\mathsf{TE}}, \{\mathsf{sk}_i^{\mathsf{TE}}\}_{i\in[n]}\right) \leftarrow \mathsf{TE.Setup}\left(1^{\lambda}, n, t\right)$ and $\mathsf{ck} \leftarrow \mathsf{CS.Setup}(1^{\lambda})$
  2. Compute $\mathsf{cm}_{\mathsf{sk}_i} = \mathsf{CS.Com}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i})$ where $\rho_{\mathsf{sk}_i} \leftarrow \mathcal{H}_4(\mathsf{sk}_i^{\mathsf{TE}})$ for $i \in [n]$.
  3. Set $\mathsf{pk} = (\mathsf{pk}_{\mathsf{TE}}, \mathsf{ck}, \{\mathsf{cm}_{\mathsf{sk}_i}\}_{i\in[n]})$
  4. Finally: (i) Output $\mathsf{sk}_i = (\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ to party P_i; and (ii) Output pk to all parties.
- $\mathsf{Enc}(\mathsf{pk}, m)$:
  1. Parse pk as $(\mathsf{pk}_{\mathsf{TE}}, \mathsf{ck}, \{\mathsf{cm}_{\mathsf{sk}_i}\}_{i\in[n]})$
  2. Sample random s and z as: $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^{\beta}$.
  3. Search for β pairs (w_i, q_i) by setting $\mathsf{cnt} = 1$, $i = 1$, $w_1 = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$ and doing:
     (a) While $i \leq \beta$ then do the following:
        i. Compute $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$
        ii. If the first ζ bits of q_i are 0, then record w_i, set $\mathsf{cnt} = \mathsf{cnt} + 1$, $i = i + 1$, $w_{i+1} = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$, and go to (a).
        iii. Else, set $\mathsf{cnt} = \mathsf{cnt} + 1$ and $w_i = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$, and go to (i).
  4. Set $w = (w_1\| \ldots \|w_\beta)$, $q = (q_1\| \ldots \|q_\beta)$, $\rho_1 = \mathcal{H}_1(s\|z\|w\|q)$, $\rho_2 = \mathcal{H}_2(s\|z\|w\|q)$.
  5. Compute $c_1 \leftarrow \mathsf{CS.Com}(\mathsf{ck}, (s\|z\|w\|q\|m); \rho_1)$, $c_2 \leftarrow \mathsf{TE.Enc}\left(\mathsf{pk}_{\mathsf{TE}}, s; \rho_2\right)$, $c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m$. Output $c = (c_1, c_2, c_3, z)$.
- $\mathsf{ParDec}(\mathsf{sk}_i, c)$:
  1. Parse c as (c_1, c_2, c_3, z) and sk_i as $(\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$.
  2. Compute partial decryption $\nu_i \leftarrow \mathsf{TE.ParDec}(\mathsf{sk}_i^{\mathsf{TE}}, c_2)$. Output ν_i.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_i, c, \{\nu_i\}_{i\in T})$:
  1. Parse pk as $(\mathsf{pk}_{\mathsf{TE}}, \mathsf{ck}, \{\mathsf{cm}_{\mathsf{sk}_i}\}_{i\in[n]})$, sk_i as $(\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ and c as (c_1, c_2, c_3, z).
  2. Decrypt c_2, obtaining $s \leftarrow \mathsf{TE.Combine}\left(\mathsf{pk}_{\mathsf{TE}}, \{\nu_i\}_{i\in T}\right)$.
  3. Search for β pairs (w_i, q_i) by setting $\mathsf{cnt} = 1$, $i = 1$, $w_1 = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$ and doing:
     (a) While $i \leq \beta$ then do the following:
        i. Compute $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$
        ii. If the first ζ bits of q_i are 0, then record w_i, set $\mathsf{cnt} = \mathsf{cnt} + 1$, $i = i + 1$, $w_{i+1} = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$, and go to (a).
        iii. Else, set $\mathsf{cnt} = \mathsf{cnt} + 1$ and $w_i = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$, and go to (i).
  4. Set $w = (w_1\| \ldots \|w_\beta)$, $q = (q_1\| \ldots \|q_\beta)$, $\rho_1 = \mathcal{H}_1(s\|z\|w\|q)$, $\rho_2 = \mathcal{H}_2(s\|z\|w\|q)$.
  5. Compute $m = c_3 \oplus \mathsf{PRF}(s\|z\|w\|q)$.
  6. Compute a self-incriminating proof as: [a]

$$\pi_i \leftarrow \mathsf{NIZKPoK}\left\{\left(s, z, w, q, \rho_1, \rho_2, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i}\right) \middle| \left(\vee_{j\in[n]} \mathsf{CS.Com}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i}) = \mathsf{cm}_{\mathsf{sk}_j}\right) \wedge c_1 = \mathsf{CS.Com}(\mathsf{ck}, (s\|z\|w\|q\|m); \rho_1) \wedge c_2 = \mathsf{TE.Enc}\left(\mathsf{pk}_{\mathsf{TE}}, s; \rho_2\right) \wedge c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m\right\}$$

  7. Output (m, π_i).
- $\mathsf{Vf}(\mathsf{pk}, c, \pi)$:
  1. Parse pk as $(\mathsf{pk}_{\mathsf{TE}}, \mathsf{ck}, \{\mathsf{cm}_{\mathsf{sk}_i}\}_{i\in[n]})$ and c as (c_1, c_2, c_3, z).
  2. Output 1 if π is a valid $\mathsf{NIZKPoK}$ for the statement above, otherwise output 0.

[a] Observe that *any* party with any sk_i for $i \in [n]$ can compute a valid proof $\pi \in \{\pi_1, \ldots, \pi_n\}$.

Fig. 10: Construction of Threshold Encryption with Self-Incriminating Proof

a one-time pad key as $\mathsf{PRF}(s\|z\|w\|q)$ using a pseudorandom function PRF where $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$. Our ciphertext $c = (c_1, c_2, c_3, z)$ is then composed by a commitment to c_1 to $s\|z\|w\|q\|m$ using randomness $\rho_1 = \mathcal{H}_1(s\|z\|w\|q)$, an encryption c_2 of s using randomness $\rho_2 = \mathcal{H}_2(s\|z\|w\|q)$ with the underlying threshold encryption scheme, $c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m$, akin to the technique of [31] but using the MPC-hard function to obtain the extra values w, q needed to derive the key to encrypt message m via the PRF. Note that z is revealed in the ciphertext in order to allow for SIP extractability (described under TSIPE SIP extractability proof in Appendix C.2).

In order to decrypt $c = (c_1, c_2, c_3, z)$, a set of $t+1$ or more parties first threshold decrypt c_2 and output s, then the sub-adversaries must first compute $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$ by computing an MPC-hard function $\mathsf{scratch}(s, z, \cdot)$, which requires at least one of them to learn (s, z). Finally, we can retrieve the message as $m = c_3 \oplus \mathsf{PRF}(s\|z\|w\|q)$.

Now notice that we can generate a SIP as a zero-knowledge proof showing knowledge of both the message m and all the randomness $s, z, w, q, \rho_1, \rho_2$ used in generating the ciphertext $c = (c_1, c_2, c_3, z)$, and of a secret key share for the underlying threshold encryption scheme, without revealing which secret

key share is used. A party simply verifies that SIP is a valid zero-knowledge proof for the statement above with respect to ciphertext $c = (c_1, c_2, c_3, z)$.

Detecting Key Share Leakage in the Distributed Adversary Model. We analyze the security of $\mathbf{\Pi}_{\mathsf{TSIPE}}$ in the Distributed Adversary model, where it is assumed that multiple independent malicious parties collaborate via an interactive protocol in order to break the TSIPE security guarantees. Hence, we focus on the worst case where each malicious party keeps their decryption key share secret while executing an arbitrary interactive protocol to achieve decryption without generating a SIP. However, malicious parties could still send their shares to a single party who locally performs decryption. In order to make it possible to generate a SIP in this case, we set $\mathsf{cm}_{\mathsf{sk}_i} = \mathsf{CS.Com}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i})$ such that $\rho_{\mathsf{sk}_i} \leftarrow \mathcal{H}_4(\mathsf{sk}_i^{\mathsf{TE}})$ where $\mathcal{H}_4 : \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle. Since $\mathsf{sk}_i^{\mathsf{TE}}$ has enough min-entropy (as it is a secret share) and $\mathcal{H}_4$ is a random oracle, ρ_{sk_i} is indistinguishable from a uniformly random string of same length to a PPT adversary and the commitment $\mathsf{cm}_{\mathsf{sk}_i}$ remains computationally binding and hiding. Generating $\mathsf{cm}_{\mathsf{sk}_i}$ in this manner allows any party who learns sk_i to prove (potentially in zero knowledge) that they have an opening to $\mathsf{cm}_{\mathsf{sk}_i}$, *i.e.,* proving that sk_i has leaked. Combining $t+1$ such proofs for different sk_i gives a SIP that any ciphertext generated under the corresponding pk may be decrypted. We observe that a commitment $\mathsf{cm}_{\mathsf{sk}_i}$ can also be used in external mechanisms to disincentivize parties from sharing their key shares, *e.g.,* through the use of a smart contract where a proof of knowledge of sk_i can be used to non-interactively extract value from party i.

Security Analysis. We formally state the security of $\mathbf{\Pi}_{\mathsf{TSIPE}}$ in Theorem 2, which is proven in Appendix C.2.

Theorem 2. *Assuming that: (i)* TE *is an IND-CPA threshold encryption scheme as per Definition 25, (ii)* CS *is a secure commitment scheme as per Definition 19, (iii)* $\mathsf{scratch}$ *is a correct and secure MPC-hard function as per Figure 1, (iv)* $\mathsf{NIZKPoK}$ *is a secure non-interactive zero-knowledge proof of knowledge system as per Definition 28, and (v)* $\mathcal{H}_1$, $\mathcal{H}_2$, *and* $\mathcal{H}_3$ *are random oracles. Let* $d, \alpha, \beta, \zeta \in \mathbb{N}$ *with* $\alpha \geq 2\beta$ *and* $\beta * (\beta - \zeta) \geq 2\lambda$ *(where* ζ *can be a function of* β*),* $s \in \{0,1\}^{n\cdot(\alpha-\beta-2)}$*. Also, let* $\eta = \beta \cdot (nd+1) \cdot 2^{\zeta+1}$*,* $\Upsilon \leq \beta \cdot 2^{\zeta-3}$ *and* $\delta \leq d-1$*. Then our protocol* $\mathbf{\Pi}_{\mathsf{TSIPE}}$ *is a secure threshold encryption with self-incriminating proof scheme* TSIPE *as per Definition 11 with* η*-bounded parties against a* (δ, Υ)*-distributed adversary* $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ *(defined in Sec. 2.3).*

References

1. Agarwal, A., Fernando, R., Pinkas, B.: Efficiently-thresholdizable batched identity based encryption, with applications. Cryptology ePrint Archive, Report 2024/1575 (2024), https://eprint.iacr.org/2024/1575
2. Alwen, J., Katz, J., Lindell, Y., Persiano, G., shelat, a., Visconti, I.: Collusion-free multiparty computation in the mediated model. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 524–540. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009). https://doi.org/10.1007/978-3-642-03356-8_31
3. Alwen, J., Katz, J., Maurer, U., Zikas, V.: Collusion-preserving computation. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 124–143. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_9
4. Alwen, J., Shelat, A., Visconti, I.: Collusion-free protocols in the mediated model. In: Wagner, D.A. (ed.) Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5157, pp. 497–514. Springer (2008). https://doi.org/10.1007/978-3-540-85174-5_28, https://doi.org/10.1007/978-3-540-85174-5_28
5. Barker, E.: NIST SP 800-57 Part 1 Rev. 5 - Recommendation for Key Management: Part 1 – General. National Institute of Standards and Technology (May 2020), https://csrc.nist.gov/pubs/sp/800/57/pt1/r4/final
6. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) Advances in Cryptology – CRYPTO'92. Lecture Notes in Computer Science, vol. 740, pp. 390–420. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1993). https://doi.org/10.1007/3-540-48071-4_28
7. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: De Santis, A. (ed.) Advances in Cryptology – EUROCRYPT'94. Lecture Notes in Computer Science, vol. 950, pp. 92–111. Springer Berlin Heidelberg, Germany, Perugia, Italy (May 9–12, 1995). https://doi.org/10.1007/BFb0053428
8. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 169–188. Springer Berlin Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011). https://doi.org/10.1007/978-3-642-20465-4_11

9. Benhamouda, F., Halevi, S., Krawczyk, H., Miao, A., Rabin, T.: Threshold cryptography as a service (in the multiserver and YOSO models). In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022: 29th Conference on Computer and Communications Security. pp. 323–336. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). https://doi.org/10.1145/3548606.3559397
10. Bentov, I., Pass, R., Shi, E.: Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919 (2016), https://eprint.iacr.org/2016/919
11. Blakley, G.R.: Safeguarding cryptographic keys. In: 1979 International Workshop on Managing Requirements Knowledge, MARK 1979, New York, NY, USA, June 4-7, 1979. pp. 313–318. IEEE (1979). https://doi.org/10.1109/MARK.1979.8817296, https://doi.org/10.1109/MARK.1979.8817296
12. Blum, M., Santis, A.D., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput. **20**(6), 1084–1118 (1991). https://doi.org/10.1137/0220068, https://doi.org/10.1137/0220068
13. Boneh, D., Partap, A., Rotem, L.: Accountability for misbehavior in threshold decryption via threshold traitor tracing. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part VII. Lecture Notes in Computer Science, vol. 14926, pp. 317–351. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68394-7_11
14. Boneh, D., Partap, A., Rotem, L.: Traceable secret sharing: Strong security and efficient constructions. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part V. Lecture Notes in Computer Science, vol. 14924, pp. 221–256. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68388-6_9
15. Brorsson, J., David, B., Gentile, L., Pagnin, E., Wagner, P.S.: PAPR: Publicly auditable privacy revocation for anonymous credentials. In: Rosulek, M. (ed.) Topics in Cryptology – CT-RSA 2023. Lecture Notes in Computer Science, vol. 13871, pp. 163–190. Springer, Cham, Switzerland, San Francisco, CA, USA (Apr 24–27, 2023). https://doi.org/10.1007/978-3-031-30872-7_7
16. Campanelli, M., David, B., Khoshakhlagh, H., Konring, A., Nielsen, J.B.: Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13793, pp. 151–180. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22969-5_6
17. Cascudo, I., David, B., Garms, L., Konring, A.: YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part I. Lecture Notes in Computer Science, vol. 13791, pp. 651–680. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22963-3_22
18. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981). https://doi.org/10.1145/358549.358563, https://doi.org/10.1145/358549.358563
19. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing. pp. 11–19. ACM Press, Chicago, IL, USA (May 2–4, 1988). https://doi.org/10.1145/62212.62214
20. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: Desmedt, Y. (ed.) Advances in Cryptology – CRYPTO'94. Lecture Notes in Computer Science, vol. 839, pp. 257–270. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1994). https://doi.org/10.1007/3-540-48658-5_25
21. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. **33**(1), 167–226 (2003). https://doi.org/10.1137/S0097539702403773, https://doi.org/10.1137/S0097539702403773
22. Dahl, M., Joye, M., Danjou, C., Rotaru, D., Demmler, D., Smart, N., Frederiksen, T., Ivanov, P., Thibault, L.T.: fhevm - confidential evm smart contracts using fully homomorphic encryption. Tech. rep., Zama (2023), https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf
23. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040, https://doi.org/10.1109/SP40000.2020.00040
24. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. In: Grossklags, J., Preneel, B. (eds.) FC 2016: 20th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 9603, pp. 169–187. Springer Berlin Heidelberg, Germany, Christ Church, Barbados (Feb 22–26, 2016). https://doi.org/10.1007/978-3-662-54970-4_10
25. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_38
26. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 66–98. Springer, Cham, Switzerland, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78375-8_3

27. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: Pomerance, C. (ed.) Advances in Cryptology – CRYPTO'87. Lecture Notes in Computer Science, vol. 293, pp. 120–127. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1988). https://doi.org/10.1007/3-540-48184-2_-8
28. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 307–315. Springer, New York, USA, Santa Barbara, CA, USA (Aug 20–24, 1990). https://doi.org/10.1007/0-387-34805-0_28
29. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
30. Dziembowski, S., Faust, S., Lizurej, T.: Individual cryptography. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 547–579. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38545-2_18
31. Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In: Nyberg, K. (ed.) Advances in Cryptology – EUROCRYPT'98. Lecture Notes in Computer Science, vol. 1403, pp. 32–46. Springer Berlin Heidelberg, Germany, Espoo, Finland (May 31 – Jun 4, 1998). https://doi.org/10.1007/BFb0054115
32. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 281–310. Springer Berlin Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46803-6_10
33. Garg, S., Gentry, C., Halevi, S., Wichs, D.: On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 518–535. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014). https://doi.org/10.1007/978-3-662-44371-2_29
34. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing. pp. 467–476. ACM Press, Palo Alto, CA, USA (Jun 1–4, 2013). https://doi.org/10.1145/2488608.2488667
35. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: you only speak once - secure MPC with stateless ephemeral roles. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12826, pp. 64–93. Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_3, https://doi.org/10.1007/978-3-030-84245-1_3
36. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing. pp. 218–229. ACM Press, New York City, NY, USA (May 25–27, 1987). https://doi.org/10.1145/28395.28420
37. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run Turing machines on encrypted data. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 536–553. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40084-1_30
38. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988). https://doi.org/10.1137/0217017, https://doi.org/10.1137/0217017
39. Gong, J., Luo, J., Wee, H.: Traitor tracing with $N^{1/3}$-size ciphertexts and $O(1)$-size keys from k-Lin. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part III. Lecture Notes in Computer Science, vol. 14006, pp. 637–668. Springer, Cham, Switzerland, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30620-4_21
40. Gong, T., Kate, A., Maji, H.K., Nguyen, H.H.: Disincentivize collusion in verifiable secret sharing. In: Fehr, S., Fouque, P. (eds.) Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part V. Lecture Notes in Computer Science, vol. 15605, pp. 34–64. Springer (2025). https://doi.org/10.1007/978-3-031-91092-0_2, https://doi.org/10.1007/978-3-031-91092-0_2
41. Goyal, R., Goyal, V.: Overcoming cryptographic impossibility results using blockchains. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017: 15th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 10677, pp. 529–561. Springer, Cham, Switzerland, Baltimore, MD, USA (Nov 12–15, 2017). https://doi.org/10.1007/978-3-319-70500-2_18
42. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 13177, pp. 252–282. Springer, Cham, Switzerland, Virtual Event (Mar 8–11, 2022). https://doi.org/10.1007/978-3-030-97121-2_10
43. Goyal, V., Song, Y., Srinivasan, A.: Traceable secret sharing and applications. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 718–747. Springer, Cham, Switzerland, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84252-9_24

44. Jakobsen, T.P., Nielsen, J.B., Orlandi, C.: A framework for outsourcing of secure computation. In: Ahn, G., Oprea, A., Safavi-Naini, R. (eds.) Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014. pp. 81–92. ACM (2014). https://doi.org/10.1145/2664168.2664170, https://doi.org/10.1145/2664168.2664170
45. Kelkar, M., Babel, K., Daian, P., Austgen, J., Buterin, V., Juels, A.: Complete knowledge: Preventing encumbrance of cryptographic secrets. Cryptology ePrint Archive, Report 2023/044 (2023), https://eprint.iacr.org/2023/044
46. Kiayias, A., Tang, Q.: How to keep a secret: leakage deterring public-key cryptosystems. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013: 20th Conference on Computer and Communications Security. pp. 943–954. ACM Press, Berlin, Germany (Nov 4–8, 2013). https://doi.org/10.1145/2508859.2516691
47. Li, R., Li, Y., Wang, Q., Duan, S., Wang, Q., Ryan, M.: Accountable decryption made formal and practical. IACR Cryptol. ePrint Arch. p. 1519 (2023), https://eprint.iacr.org/2023/1519
48. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 643–673 (2017). https://doi.org/10.1007/978-3-319-56614-6_22, https://doi.org/10.1007/978-3-319-56614-6_22
49. Project, O.P.: The oasis blockchain platform. Tech. rep., Oasis Protocol Foundation (2020)
50. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., USA (1996)
51. Rondelet, A., Kilbourn, Q.: Threshold encrypted mempools: Limitations and considerations. arXiv preprint arXiv:2307.10878 (2023)
52. Ryan, M.D.: Making decryption accountable. In: Stajano, F., Anderson, J., Christianson, B., Matyás, V. (eds.) Security Protocols XXV - 25th International Workshop, Cambridge, UK, March 20-22, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10476, pp. 93–98. Springer (2017). https://doi.org/10.1007/978-3-319-71075-4_11, https://doi.org/10.1007/978-3-319-71075-4_11
53. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
54. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: Robinson, P., Ellen, F. (eds.) 38th ACM Symposium Annual on Principles of Distributed Computing. pp. 347–356. Association for Computing Machinery, Toronto, ON, Canada (Jul 29 – Aug 2, 2019). https://doi.org/10.1145/3293611.3331591
55. Young, A.L., Yung, M.: Cryptovirology: Extortion-based security threats and countermeasures. In: 1996 IEEE Symposium on Security and Privacy. pp. 129–140. IEEE Computer Society Press, Oakland, CA, USA (1996). https://doi.org/10.1109/SECPRI.1996.502676

Supplementary Material

A Additional Preliminaries

In this section, we provide our additional technical preliminaries.

A.1 Digital Signatures

Here, we describe the general syntax for digital signature in Definition 16.

Definition 16 (Digital Signature). *A digital signature scheme is a tuple of* PPT *algorithms* $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *defined as follows:*

1. $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{KGen}(1^\lambda)$ *is a randomized key generation algorithm that takes as input the security parameter* 1^λ *and outputs a key-pair* $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}})$*;*
2. $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{SIG}}, m)$*, the signing algorithm takes an input a a secret key* $\mathsf{sk}_{\mathsf{SIG}}$ *and a message* $m \in \{0,1\}^\lambda$*, outputting a signature* σ*;*
3. $1/0 \leftarrow \mathsf{Vf}(\mathsf{pk}_{\mathsf{SIG}}, m, \sigma)$*, the verification algorithm outputs* 1 *if* σ *is a valid signature on* m *generated with* $\mathsf{sk}_{\mathsf{SIG}}$*, and outputs* 0 *otherwise.*

A signature scheme SIG *must satisfy the standard notions of* ***correctness*** *(Definition 17) and* ***unforgeability*** *(Definition 18) (i.e., existentially unforgeable against adaptive chosen message attacks (EUF-CMA) [38]) described below.*

Definition 17 (Correctness). *A signature scheme* $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *is correct if for any security parameter* λ *and any message* $m \in \{0,1\}^\lambda$*, we have that*

$$\Pr\left[\mathsf{Vf}(\mathsf{pk}_{\mathsf{SIG}}, m, \sigma) = 1 \,\middle|\, \begin{array}{l}(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{KGen}(1^\lambda \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}, m)\end{array}\right] = 1$$

Definition 18 (Unforgeability). *A signature scheme* $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *is unforgeable if for any security parameter* λ *and for all* PPT *adversaries* $\mathcal{A}$*, advantage* $\mathsf{Adv}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}}$ *of the* $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}}$ *(in Figure 11) is negligible.*

$\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}}$

Game steps:

1. Initialize an empty list $\mathcal{Q} \leftarrow \emptyset$
2. $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{KGen}(1^\lambda)$
3. $(m', \sigma') \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}_{\mathsf{sk}_{\mathsf{SIG}}}}(1^\lambda, \mathsf{pk}_{\mathsf{SIG}})$

Oracle:

$\mathcal{O}^{\mathsf{Sign}}_{\mathsf{sk}_{\mathsf{SIG}}}(m)$:

1. $\mathcal{Q} \leftarrow \mathcal{Q} \cup m$
2. $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}, m)$
3. **return** σ

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}} = \Pr\left[\mathsf{Vf}(\mathsf{pk}_{\mathsf{SIG}}, m', \sigma') = 1 \wedge m' \notin \mathcal{Q}\right]$$

Fig. 11: Unforgeability Game for SIG scheme executed between a challenger and an adversary $\mathcal{A}$ given unlimited oracle access to $\mathcal{O}^{\mathsf{Sign}}_{\mathsf{sk}_{\mathsf{SIG}}}(m)$.

A.2 Commitments

Here, we recall the syntax for a commitment scheme.

Definition 19 (Commitment Scheme). *A commitment scheme* CS *consists of the tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Com})$ *defined as follows:*

1. $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$*, is a randomized algorithm that takes as input the security parameter* 1^λ *and outputs a commitment key* ck*. The commitment key* ck *defines a message space* $\mathcal{M}$ *and a randomizer space* $\mathcal{R}$*.*
2. $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{ck}, \mathsf{s}; \rho)$*, the commitment algorithm takes as inputs the commitment key* ck*, an input message* $\mathsf{s} \in \mathcal{M}$ *and randomness* $\rho \in \mathcal{R}$*, and outputs a commitment* cm*.*

A commitment scheme CS *must satisfy the standard properties of* ***binding*** *(Definition 20) and* ***hiding*** *(Definition 21) described below.*

Definition 20 (Binding). *A commitment scheme* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Com})$ *is binding if for any security parameter* λ*, if no* PPT *adversary can come up with two pairs* (s, ρ)*,* (s', ρ') *such that* $\mathsf{s} \neq \mathsf{s}'$ *and* $\mathsf{Com}(\mathsf{ck}, \mathsf{s}; \rho) = \mathsf{Com}(\mathsf{ck}, \mathsf{s}'; \rho')$ *for* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$*.*

Definition 21 (Hiding). *A commitment scheme* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Com})$ *is hiding if for any security parameter* λ*, for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, advantage* $\mathsf{Adv}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$ *of the* $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$ *(in Figure 12) is negligible.*

$\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$

Game steps:

1. $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(\mathsf{st}, \mathsf{s}_0, \mathsf{s}_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{ck})$
3. Sample $b \xleftarrow{\$} \{0, 1\}$
4. $\mathsf{cm}_b \leftarrow \mathsf{Com}(\mathsf{ck}, \mathsf{s}_b; \rho)$
5. $b' \leftarrow \mathcal{A}_2(1^\lambda, \mathsf{st}, \mathsf{cm}_b)$
6. **return** b'

As there are only two outcomes, there is a $\frac{1}{2}$ probability that a random guess will be correct, the adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}} = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right|$$

Fig. 12: Hiding Game for CS scheme.

A.3 Extractable Witness Encryption: Syntax and Security

Here, we recall the syntax and security definition of extractable witness encryption following broadly the definitions of [34, 37] as presented in [33].

Definition 22 (Extractable Witness Encryption). *Let $\mathcal{L}_{\mathsf{eWE}}$ be an* **NP** *language with witness relation $\mathcal{R}_{\mathsf{eWE}}$. An extractable witness encryption scheme* eWE *for language $\mathcal{L}_{\mathsf{eWE}}$ with message space $\mathcal{M} \subseteq \{0,1\}^*$ consists of the following two polynomial-time algorithms* $(\mathsf{Enc}, \mathsf{Dec})$ *are as follows:*

1. $c \leftarrow \mathsf{Enc}(1^\lambda, \mathcal{L}_{\mathsf{eWE}}, \mathsf{inst}, m)$*, the encryption algorithm takes as input a security parameter 1^λ, the language $\mathcal{L}_{\mathsf{eWE}}$, an unbounded-length string* inst*, and a message $m \in \mathcal{M}$, and outputs a ciphertext c.*
2. $m/\perp \leftarrow \mathsf{Dec}(c, \mathsf{wit})$*, the decryption algorithm takes as input a ciphertext c and an unbounded-length string (witness)* wit*, and outputs a message m, otherwise output $\perp$.*

A scheme eWE *must satisfy the following properties:* ***Correctness*** *(Definition 23) and* ***Extractable Security*** *(Definition 24) described below.*

Definition 23 (Correctness). *An extractable witness encryption scheme* $\mathsf{eWE} = (\mathsf{Enc}, \mathsf{Dec})$ *for language $\mathcal{L}_{\mathsf{eWE}}$ is correct if for any security parameter λ, for any message $m \in \mathcal{M}$, and for any* $\mathsf{inst} \in \mathcal{L}_{\mathsf{eWE}}$ *such that $\mathcal{R}_{\mathsf{eWE}} \in (\mathsf{inst}, \mathsf{wit})$ holds, we have that,*

$$\Pr\left[\mathsf{Dec}(\mathsf{wit}, c) = m \mid \mathsf{Enc}(1^\lambda, \mathcal{L}_{\mathsf{eWE}}, \mathsf{inst}, m)\right] = 1$$

Extractable Security. An extractable witness encryption scheme is said to be extractable secure if an adversary can learn some non-trivial information about the encrypted message only if it knows a *witness* for the instance used during encryption. We define the formal extractable security in Definition 24.

Definition 24 (Extractable Security). *An extractable witness encryption scheme* $\mathsf{eWE} = (\mathsf{Enc}, \mathsf{Dec})$ *for language $\mathcal{L}_{\mathsf{eWE}}$ with witness relation $\mathcal{R}_{\mathsf{eWE}}$ is extractable secure if for any security parameter λ and for all* PPT *adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and polynomial $p(\cdot)$, there exists a* PPT *extractor* EXT *and polynomial $q(\cdot)$ such that for every pair of messages $m_0, m_1 \in \mathcal{M}$ and for any* $\mathsf{inst} \in \mathcal{L}_{\mathsf{eWE}}$*,*

$$\mathsf{Adv}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}} \geq \frac{1}{2} + \frac{1}{p(\lambda)}$$
$$\Rightarrow \mathsf{Succ}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathsf{EXT}} \geq \frac{1}{q(\lambda)}$$

where $\mathsf{Adv}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$ *is advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in* $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$ *(in Figure 13) and* $\mathsf{Succ}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathsf{EXT}}$ *is the success probability of the extractor* EXT *for extracting the witness* wit *such that* $(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{eWE}}$*.*

A.4 Threshold Encryption

We here outline the formal algorithms and definitions we assume for threshold decryption.

Definition 25 (Threshold Encryption). *A threshold encryption scheme* TE *consists of the tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine})$ *with the following requirements:*

1. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$*, is a randomized algorithm that takes as input the security parameter 1^λ, the number of shares n and the threshold t. It computes key parameters threshold encryption where* pk *is the public encryption key and* sk_i *is threshold decryption key share for party P_i.*
2. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}; \rho)$*, is a randomized algorithm that takes as input the public key* pk*, a message* msg *and randomness ρ, outputting a ciphertext* ct*. If ρ is not explicitly written, it is assumed that it is sampled uniformly at random.*
3. $\mu_i \leftarrow \mathsf{ParDec}(\mathsf{pk}, \mathsf{sk}_i, \mathsf{ct})$*, the partial decryption algorithm takes as input the public key* pk*, the secret key share* sk_i *and a ciphertext* ct*, producing a partial decryption μ_i.*
4. $\mathsf{msg}/\perp \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{ct}, \{\mu_i\}_{i\in T})$*, the partial decryption combine algorithm that takes as input the public key* pk*, the partial decryption $\{\mu_i\}_{i\in T}$ for a set T where $|T| \geq t+1$. It outputs the original message* msg *which* ct *encrypts, otherwise output $\perp$.*

$\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$

Game steps:

1. $(\mathsf{st}, \mathsf{inst}, m_0, m_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathcal{L}_{\mathsf{eWE}})$
2. Sample $b \xleftarrow{\$} \{0,1\}$
3. $c_b \leftarrow \mathsf{Enc}(1^\lambda, \mathcal{L}_{\mathsf{eWE}}, \mathsf{inst}, m_b)$
4. $b' \leftarrow \mathcal{A}_2(1^\lambda, \mathsf{st}, c_b)$
5. **return** b'

As there are only two outcomes, there is a $\frac{1}{2}$ probability that a random guess will be correct, the adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}} = \mathsf{Pr}\left[b = b'\right]$$

The extractor EXT's success probability is:

$$\mathsf{Succ}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathsf{EXT}} = \mathsf{Pr}\left[(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{eWE}} \middle| \mathsf{wit} \leftarrow \mathsf{EXT}^{\mathcal{A}(\cdot)}(1^\lambda, \mathcal{L}_{\mathsf{eWE}}, \mathsf{inst}, m_0, m_1)\right]$$

Fig. 13: Extractable Security Game for eWE scheme.

We require a threshold encryption scheme TE *satisfy the standard properties:* ***Correctness*** *(Definition 26) and* ***IND-CPA Security*** *(Definition 27) below.*

Definition 26 (Correctness). *A threshold encryption scheme* $\mathsf{TE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine})$ *is correct for any security parameter* λ *with correctly generated keys* $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$ *and for any* $\mathsf{msg} \in \{0,1\}^\lambda$ *(from the permissible domain) where* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$*, we have:*

$$\forall T \subset [n] \text{ with } |T| \geq t+1 \ : \mathsf{Pr}\left[\mathsf{Combine}\left(\mathsf{pk}, \mathsf{ct}, \{\mathsf{ParDec}(\mathsf{pk}, \mathsf{sk}_i, \mathsf{ct})\}_{i\in T}\right) = \mathsf{msg}\right] = 1$$

Definition 27 (IND-CPA Security). *A threshold encryption scheme* $\mathsf{TE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine})$ *is IND-CPA secure if for any* n *and* t *where* $0 < t < n$*, and for all* PPT *adversaries* $\mathcal{A}$*, advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$ *of the* $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$ *(in Figure 14) is negligible.*

$\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$

Game steps:

1. The adversary $\mathcal{A}$ picks n and t.
2. $\mathcal{A}$ chooses a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| \leq t$.
3. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
4. $(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\tilde{T}})$
5. Sample $b \xleftarrow{\$} \{0,1\}$
6. $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
7. $b' \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\tilde{T}}, c_b, \{\mu_{b,i}\}_{i\in\tilde{T}})$
8. **return** b'

As there are only two outcomes, there is a $\frac{1}{2}$ probability that a random guess will be correct, the adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}} = \left|\mathsf{Pr}\left[b = b'\right] - \frac{1}{2}\right|$$

Fig. 14: IND-CPA Security Game for TE scheme.

A.5 Non-interactive Zero-Knowledge

We here outline the formal algorithms and definitions we assume for non-interactive zero-knowledge (NIZK) [12] proof system.

Definition 28 (Non-Interactive Zero-Knowledge Proof). *A non-interactive zero-knowledge proof system* NIZK *for an* $\mathbf{NP}$*-language* $\mathcal{L}_{\mathsf{NIZK}}$ *with witness relation* $\mathcal{R}_{\mathsf{NIZK}}$ *is a tuple of* PPT *algorithms* $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *such that:*

1. $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$*, is a randomized algorithm that takes as input the security parameter* 1^λ *and outputs a common random string* crs*.*
2. $\pi \leftarrow \mathsf{P}(\mathsf{crs}, \mathsf{inst}, \mathsf{wit})$*, the prover algorithm takes as input a common random string* crs*, a statement* $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ *and a witness* wit *and outputs a proof* π*.*
3. $1/0 \leftarrow \mathsf{V}(\mathsf{crs}, \mathsf{inst}, \pi)$*, the verifier algorithm takes as input a common random string* crs*, a statement* $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ *and a proof* π*. It outputs 1 if it accepts the proof* π*, otherwise outputs 0.*

We require a NIZK proof system $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *must satisfy the following properties:* ***Completeness****,* ***Soundness****, and* ***Zero-Knowledge*** *described below. A NIZK proof system satisfying all these properties is called a* ***secure NIZK*** *[12] proof system.*

Definition 29 (Completeness). *A NIZK proof system* $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for an* $\mathbf{NP}$*-language* $\mathcal{L}_{\mathsf{NIZK}}$ *with witness relation* $\mathcal{R}_{\mathsf{NIZK}}$ *is correct if for any security parameter* λ*, and for any* $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ *such that* $\mathcal{R}_{\mathsf{NIZK}} \in (\mathsf{inst}, \mathsf{wit})$ *holds, we have that,*

$$\mathsf{Pr}\left[\mathsf{V}(\mathsf{crs}, \mathsf{inst}, \mathsf{P}(\mathsf{crs}, \mathsf{inst}, \mathsf{wit})) \mid \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)\right] = 1$$

Definition 30 (Soundness). *A NIZK proof system* $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for an* $\mathbf{NP}$*-language* $\mathcal{L}_{\mathsf{NIZK}}$ *with witness relation* $\mathcal{R}_{\mathsf{NIZK}}$ *is sound if for any security parameter* λ*, for all* PPT *provers* P^* *and for any* $\mathsf{inst} \notin \mathcal{L}_{\mathsf{NIZK}}$*, then there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that,*

$$\mathsf{Pr}\left[\mathsf{V}(\mathsf{crs}, \mathsf{inst}, \pi) = 1 \middle| \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) \end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

Definition 31 (Zero-Knowledge). *A NIZK proof system* $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for an* $\mathbf{NP}$*-language* $\mathcal{L}_{\mathsf{NIZK}}$ *with witness relation* $\mathcal{R}_{\mathsf{NIZK}}$ *is zero-knowledge if for any security parameter* λ *there exists a* PPT *simulator* $\mathcal{S}$ *such that for every* $\mathcal{R}_{\mathsf{NIZK}} \in (\mathsf{inst}, \mathsf{wit})$*, the following distribution ensembles are computationally indistinguishable,*

$$\left\{(\mathsf{crs}, \pi) \middle| \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, \mathsf{inst}, \mathsf{wit}) \end{matrix}\right\}_{\lambda \in \mathsf{N}} \approx \left\{(\mathsf{crs}, \pi) \middle| \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi \leftarrow \mathcal{S}(1^\lambda, \mathsf{crs}, \mathsf{inst}) \end{matrix}\right\}_{\lambda \in \mathsf{N}}$$

NIZK Proof-of-Knowledge (NIZKPoK) [6]. A proof-of-knowledge is an additional property which a NIZK proof system can have. We say that a zero-knowledge proof system is a NIZKPoK if an adversary must "know" a witness wit to compute a proof for $(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{NIZK}}$. More formally, a NIZK proof system is said to be an NIZKPoK for the relation $\mathcal{R}_{\mathsf{NIZK}}$, if the following are satisfied:

- **Completeness:** On common input statement $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$, if the honest prover P gets as private witness wit such that $(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{NIZK}}$, then the verifier V always accepts.
- **Soundness:** The soundness for proofs-of-knowledge is formalized by defining a prover P^* which outputs an accepted proof π and demonstrating an efficient algorithm EXT called the knowledge extractor which can interact with P^* and output a witness wit such that $(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{NIZK}}$. Depending on the proof construction, the extractor may need to rewind P^* (a rewinding extractor) or inspect P^*'s internal state (a non-black box extractor).
- **Zero-Knowledge:** A proof-of-knowledge is zero-knowledge if the proof π reveals nothing about the witness wit. Formally, this is established by an efficient algorithm $\mathcal{S}$ called the simulator which is given any statement $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ and the ability to program the random oracle to give specified responses, can output simulated proofs π' which is indistinguishable from real proofs such that the verifier V accepts the proofs π'.

Throughout our paper, we write $\mathsf{NIZKPoK}\{\mathsf{wit} \mid (\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{NIZK}}\}$ to denote a generic non-interactive zero-knowledge proof of knowledge for relation $\mathcal{R}_{\mathsf{NIZK}}$.

$\mathsf{Game}^{\mathsf{Cor}}_{\mathsf{TBIBE},\mathcal{A}}(1^\lambda, \mathsf{B}, n, \mathsf{Corr})$

Game steps:

1. The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\mathsf{B}, 1^n)$ and gives pp to $\mathcal{A}$.
2. The challenger runs $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{msk}_i)\}_{i\in[n]}) \leftarrow \mathsf{KGen}(\mathsf{pp})$ and gives $(\mathsf{pk}, \{\mathsf{msk}_i\}_{i\in\mathsf{Corr}}, \{\mathsf{pk}_i\}_{i\in[n]})$ to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ picks a subset of corrupt parties $\mathsf{Corr} \subset [n]$ such that $|\mathsf{Corr}| \leq \lfloor \frac{n-1}{2} \rfloor$.
4. The challenger generates a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}, \mathsf{id}, \mathsf{b})$, a digest $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\})$ and a partial digest-batch label-specific secret key $\mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, \mathsf{d}, \mathsf{b})$ for all $i \in [n] \setminus \mathsf{Corr}$.
5. The adversary $\mathcal{A}$ is given access to the partial digest-batch label-specific secret key generation oracle of the honest parties for arbitrary inputs of the adversary's choice, *i.e.,* $\mathsf{ComputeKeyShare}(\mathsf{msk}_i, \cdot, \cdot)$ for all $i \in [n] \setminus \mathsf{Corr}$.
6. The adversary $\mathcal{A}$ generates a partial digest-batch label-specific secret key sk_i for all $i \in \mathsf{Corr}$, *i.e.,* $\mathsf{sk}_i \leftarrow \mathcal{A}(\{\mathsf{msk}_j\}_{j\in\mathsf{Corr}}, \{\mathsf{pk}_j\}_{j\in[n]\setminus\mathsf{Corr}}, \mathsf{pk}, \mathsf{msg}, \mathsf{id}, \mathsf{b}, \mathsf{ct}, \mathcal{J})$.
7. The challenger computes $\mathsf{sk} \leftarrow \mathsf{ComputeKeyAggregate}(\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}, \mathsf{d}, \mathsf{b})$.
8. The challenger decrypts ct as $\mathsf{msg}' \leftarrow \mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, \mathsf{d}, \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}, \mathsf{id}, \mathsf{b})$
9. **return** 1 iff $\mathsf{msg}' \neq \mathsf{msg}$, otherwise **return** 0

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{Cor}}_{\mathsf{TBIBE},\mathcal{A}} = \Pr\left[\mathsf{msg} \neq \mathsf{msg}'\right]$$

Fig. 15: Correctness Game for the Thresholdizable Batched IBE scheme.

A.6 Pseudorandom Function

We recall the definition of a pseudorandom function (PRF) family.

Definition 32 (Pseudorandom Function Family). *A pseudorandom function (PRF) family is a family of polynomial-time computable functions* $\mathsf{PRF} : \mathcal{X} \to \mathcal{Y}$ *(where the sets* $(\mathcal{X}, \mathcal{Y})$ *are all parameterized by the security parameter* $\lambda \in \mathbb{N}$*, and where each function* $\mathsf{PRF}(\cdot)$*, such that for any PPT adversary* $\mathcal{A}$*, any random function* $f \xleftarrow{\$} \mathsf{Funcs}(\mathcal{X}, \mathcal{Y})$*, we have*

$$\left|\Pr\left[\mathcal{A}^{\mathsf{PRF}(\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

A.7 Thresholdizable Batched IBE

We define the syntax of a Thresholdizable Batched Identity-Based Encryption (TBIBE) scheme in Sec. 2.1 and recall here the security definition of TBIBE scheme from [1].

Definition 33 (Correctness). *A Thresholdizable Batched IBE scheme* $\mathsf{TIBE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Digest}, \mathsf{ComputeKeyShare}, \mathsf{ComputeKeyAggregate})$ *is correct for any security parameter* $\lambda \in \mathbb{N}, \mathsf{B} \in \mathbb{N}, n \in \mathbb{N}, \mathsf{msg} \in \mathcal{M}, \mathsf{b} \in \mathcal{B}, \mathsf{id} \in \mathcal{I}, \mathcal{J} \subseteq \mathcal{I}$ *such that* $|\mathcal{J}| = \mathsf{B}$ *and* $\mathsf{id} \in \mathcal{J}$*,* $\mathsf{Corr} \subset [n]$ *such that* $|\mathsf{Corr}| \leq \lfloor \frac{n-1}{2} \rfloor$ *and for any unbounded adversary* $\mathcal{A}$*:*

$$\mathsf{Adv}^{\mathsf{Cor}}_{\mathsf{TBIBE},\mathcal{A}} = 0$$

for $\mathsf{Adv}^{\mathsf{Cor}}_{\mathsf{TBIBE},\mathcal{A}}$ *defined in Figure 15.*

Definition 34 (Security). *A Thresholdizable Batched IBE scheme* $\mathsf{TIBE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Digest}, \mathsf{ComputeKeyShare}, \mathsf{ComputeKeyAggregate})$ *is secure for any security parameter* $\lambda \in \mathbb{N}, \mathsf{B} \in \mathbb{N}, n \in \mathbb{N}$*, for all* PPT *adversaries* $\mathcal{A}$*, for all* $\mathsf{Corr} \subset [n]$ *such that* $|\mathsf{Corr}| \leq \lfloor \frac{n-1}{2} \rfloor$*, there exists some negligible function* $\mathsf{negl}(\lambda)$ *such that the following holds:*

$$\mathsf{Adv}^{\mathsf{Sec}}_{\mathsf{TBIBE},\mathcal{A}} < \mathsf{negl}(\lambda)$$

where the TBIBE *security game* $\mathsf{Sec}^{\mathcal{A},b}_{\mathsf{TBIBE}}$ *is defined in Figure 16.*

A.8 Extended Formalization of Proof-of-Stake (PoS) Blockchains

In this section, we give an overview of the full framework from [41] for arguing about PoS blockchain protocol security. We recall, in almost verbatim form, the overview given in [16] of the blockchain execution model from [41].

$\mathsf{Game}^{\mathsf{Sec}}_{\mathsf{TBIBE},\mathcal{A}}(1^\lambda, \mathsf{B}, n, \mathsf{Corr})$

Game steps:

Setup:

1. The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\mathsf{B}, 1^n)$ and $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{msk}_i)\}_{i\in[n]}) \leftarrow \mathsf{KGen}(\mathsf{pp})$.
2. The challenger gives $(\mathsf{pp}, \mathsf{pk}, \{\mathsf{msk}_i\}_{i\in\mathsf{Corr}}, \{\mathsf{pk}_i\}_{i\in[n]})$ to the adversary $\mathcal{A}$.

Pre-challenge queries: $\mathcal{A}$ may issue an arbitrary number of these:

1. $\mathcal{A}$ sends identities $\{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$ along with a batch label b to the challenger.
2. If a key computation query has already been made with batch label b, the challenger halts the game. Otherwise, the challenger does the following:
   (a) Compute $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\})$ and $\mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, \mathsf{d}, \mathsf{b})$ for all $i \in [n] \setminus \mathsf{Corr}$.
   (b) The challenger gives $\{\mathsf{sk}_i\}_{i\in[n]\setminus\mathsf{Corr}}$ to the adversary $\mathcal{A}$.

Challenge round: Once during the game, $\mathcal{A}$ may decide that the current round is the *challenge round*. The challenge round proceeds as follows:

1. The adversary $\mathcal{A}$ sends two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}$ and an identity-batch label pair $(\mathsf{id}^*, \mathsf{b}^*)$ on which it wishes to be challenged.
2. If key computation query $(\{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}, \mathsf{b})$ has already been made with batch label $\mathsf{b} = \mathsf{b}^*$ and $\mathsf{id}^* \in \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$, the challenger halts the game.
3. Otherwise, the challenger samples $b \xleftarrow{\$} \{0,1\}$ and computes $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}_b, \mathsf{id}^*, \mathsf{b}^*)$ and gives ct to the adversary $\mathcal{A}$.

Post-challenge queries: After the challenger round, $\mathcal{A}$ may again issue an arbitrary number of *key computation queries*, with the additional restriction that $\mathcal{A}$ cannot query $(\{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}, \mathsf{b}^*)$ with $\mathsf{id}^* \in \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$:

1. $\mathcal{A}$ sends identities $\{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$ along with a batch label b to the challenger.
2. If a key computation query has already been made with batch label b **or if** $\mathsf{b} = \mathsf{b}^*$ **and** $\mathsf{id}^* \in \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\}$, the challenger halts the game.
3. Otherwise, the challenger does the following:
   (a) Compute $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pk}, \{\mathsf{id}_1, \ldots, \mathsf{id}_\mathsf{B}\})$ and $\mathsf{sk}_i \leftarrow \mathsf{ComputeKeyShare}(\mathsf{msk}_i, \mathsf{d}, \mathsf{b})$ for all $i \in [n] \setminus \mathsf{Corr}$.
   (b) The challenger gives $\{\mathsf{sk}_i\}_{i\in[n]\setminus\mathsf{Corr}}$ to the adversary $\mathcal{A}$.

Output: At any point in time, $\mathcal{A}$ can decide to halt and output a bit $b' \in \{0,1\}$. The game then halts with the same output b'.

The adversary's advantage in this game is:

$$\mathsf{Adv}^{\mathsf{Sec}}_{\mathsf{TBIBE},\mathcal{A}} = \left|\mathsf{Pr}\left[b = b'\right] - \frac{1}{2}\right|$$

Fig. 16: Security Game $\mathsf{Sec}^{\mathcal{A},b}_{\mathsf{TBIBE}}$ for the Thresholdizable Batched IBE scheme.

Blockchain Structure. A genesis block $B_0 = \{(\mathsf{SIG.pk}_1, \mathsf{aux}_1, \mathsf{stake}_1), \ldots, (\mathsf{SIG.pk}_n, \mathsf{aux}_n, \mathsf{stake}_n), \mathsf{aux}\}$ associates each party P_i to a public signature verification key $\mathsf{SIG.pk}_i$, an amount of stake stake_i and auxiliary information aux_i (*i.e.,* any other relevant information required by the blockchain protocol, such as verifiable random function public keys). A blockchain $\mathtt{B}$ relative to a genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence such that $B_i = (\mathcal{H}(B_{i-1}), \mathsf{d}, \mathsf{aux}))$ where $\mathcal{H}(B_{i-1})$ is a collision-resistant hash of the previous block, d is data and aux is auxiliary information required by the blockchain protocol. We denote by $\mathtt{B}^{\lceil\ell}$ the chain (sequence of blocks) $\mathtt{B}$ where the last ℓ blocks have been removed and if $\ell \geq |\mathtt{B}|$ then $\mathtt{B}^{\lceil\ell} = \epsilon$ (empty symbol). Also, if $\mathtt{B}_1$ is a prefix of $\mathtt{B}_2$ we write $\mathtt{B}_1 \preceq \mathtt{B}_2$. Each party participating in the protocol has public identity P_i and most messages will be a transaction of the following form: $m = (P_i, P_j, q, \mathsf{aux})$ where P_i transfers q coins to P_j along with some optional, auxiliary information aux.

Blockchain Protocol Execution. Recalling the three polynomial-time algorithms $(\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ from Definition 2, we can, at the high level, view the execution of a blockchain protocol Γ^V as follows: The participant in the protocol runs the $\mathsf{UpdateState}^V$ algorithm to get the latest blockchain state, the $\mathsf{GetRecords}$ algorithm is used to extract an ordered

sequence of blocks encoded in the blockchain state variable and the $\mathsf{Broadcast}$ algorithm is used by a party when it wants to post a new message on the blockchain if accepted by V.

The blockchain protocol Γ^V execution is directed by the environment $\mathcal{Z}$ which classifies parties as either *honest* or *corrupt*, and is also responsible for providing inputs/records to all parties in each round. All honest parties execute Γ^V on input 1^λ with an empty local state st, and all corrupted parties are controlled by the adversary $\mathcal{A}$ who also controls network including delivery of messages between all parties. The execution of the protocol proceeds as follows (and the following description is mostly taken from [16, 41]).

- The execution proceeds in rounds that model time steps. In each round r, all the honest parties potentially receive a message(s) m from the environment $\mathcal{Z}$ and potentially receive incoming network messages delivered by the adversary $\mathcal{A}$. The honest parties may perform any computation, broadcast messages (using $\mathsf{Broadcast}$ algorithm), and/or update their local states.
- The adversary $\mathcal{A}$ is responsible for delivering all messages sent by honest parties to all other parties. $\mathcal{A}$ cannot modify messages broadcast by honest parties but may delay and reorder messages on the network.
- At any point $\mathcal{Z}$ can communicate with adversary $\mathcal{A}$ or use $\mathsf{GetRecords}$ to retrieve a view of the local state of any party participating in the protocol.

The joint view of all parties (*i.e.,* all inputs, random coins, and messages received) in the above protocol execution can be denoted by the random variable $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$. Note that the joint view of all parties fully determines the execution. We define the view of the party P_i as $\mathsf{VIEW}_{P_i}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$ and the view of the adversary $\mathcal{A}$ as $\mathsf{VIEW}_{\mathcal{A}}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$. If it is clear from the context which execution the argument is referring to, then we just write VIEW_i. We assume that it is possible to take a snapshot *i.e.,* a view of the protocol after the first r rounds have been executed. We denote that by $\mathsf{VIEW}^r \leftarrow \mathsf{EXEC}_r^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$. Furthermore, we can resume the execution starts with this view and continue until round $\tilde{r}$ resulting in the full view including round $\tilde{r}$ denoted by $\mathsf{VIEW}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma^V}_{(\mathsf{VIEW}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$.

Defining stake *and* $\mathsf{u\text{-}stakefrac}$. We denote the stake of party P_i as $\mathsf{stake}_i = \mathsf{stake}(\mathtt{B}, i)$ which takes as input a local blockchain $\mathtt{B}$ and a party P_i and outputs a number representing the stake of party P_i as per the blockchain $\mathtt{B}$. Here, $\mathsf{stake}(\cdot, \cdot)$ is a polynomial time algorithm that takes as inputs the blockchain $\mathtt{B}$ and a party's public identity and outputs a rational value.

Let an adversary $\mathcal{A}$ that controls all parties with public identities in the set $\mathcal{X}$, its sum of stake controlled by the adversary as per blockchain $\mathtt{B}$ can computed as $\mathsf{stake}_{\mathcal{A}}(\mathtt{B}) = \sum_{j \in \mathcal{X}} \mathsf{stake}(\mathtt{B}, j)$, and the total stake held by all parties can be computed as $\mathsf{stake}_{\mathsf{total}}(\mathtt{B}) = \sum_i \mathsf{stake}(\mathtt{B}, j)$. We compute the adversaries relative stake ratio as $\mathsf{stake\text{-}ratio}_{\mathcal{A}}(\mathtt{B}) = \frac{\mathsf{stake}_{\mathcal{A}}(\mathtt{B})}{\mathsf{stake}_{\mathsf{total}}(\mathtt{B})}$. Also, we will simply write $\mathsf{stake}_{\mathcal{A}}$, $\mathsf{stake}_{\mathsf{total}}$, and $\mathsf{stake\text{-}ratio}_{\mathcal{A}}$ whenever $\mathtt{B}$ is clear from context.

We also consider the PoS-fraction $\mathsf{u\text{-}stakefrac}(\mathtt{B}, \ell)$ as the amount of unique stake whose proof is provided in the last ℓ mined blocks. More precisely, let $\mathbb{M}$ be the index i corresponding to miners P_i of the last ℓ blocks in $\mathtt{B}$ then we compute the PoS-fraction as follows,

$$\mathsf{u\text{-}stakefrac}(\mathtt{B}, \ell) = \frac{\sum_{i \in \mathbb{M}} \mathsf{stake}(\mathtt{B}, i)}{\mathsf{stake}_{\mathsf{total}}}$$

A note on corruption. For simplicity in the above execution we restrict the environment to only allow static corruption while the execution described in [48] supports adaptive corruption with erasures.

A note on admissible environments. Pass et al. [48] specifies a set of restrictions on $\mathcal{A}$ and $\mathcal{Z}$ such that only compliant executions are considered and argues that certain security properties hold with overwhelming probability for these executions. An example of such a restriction is that $\mathcal{A}$ should deliver network messages to honest parties within Δ rounds.

Blockchain Setup and Key Knowledge. As in [26], we assume that the genesis block is generated by an initialization functionality $\mathcal{F}_{\mathtt{INIT}}$ that registers all parties' keys. Moreover, we assume that primitives specified in separate functionalities in [26] as incorporated into $\mathcal{F}_{\mathtt{INIT}}$. $\mathcal{F}_{\mathtt{INIT}}$ is executed by the environment $\mathcal{Z}$ as defined below and is parameterized by a stake distribution associating each party P_i to an initial stake stake_i. Upon being activated by P_i for the first time, $\mathcal{F}_{\mathtt{INIT}}$ generates a signature key pair $(\mathsf{SIG.pk}_i, \mathsf{SIG.sk}_i)$ and auxiliary information aux_i, and sending $(\mathsf{SIG.pk}_i, \mathsf{SIG.sk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$ to P_i as

response. After all parties have activated $\mathcal{F}_{\text{INIT}}$, it responds to requests for a genesis block by providing $B_0 = \{(\mathsf{SIG.pk}_1, \mathsf{aux}_1, \mathsf{stake}_1), \ldots, (\mathsf{SIG.pk}_n, \mathsf{aux}_n, \mathsf{stake}_n), \mathsf{aux}\}$, where aux is generated according to the underlying blockchain consensus protocol.

Since $\mathcal{F}_{\text{INIT}}$ generates keys for all parties, we capture the fact that even corrupted parties have registered public keys and auxiliary information such that they know the corresponding secret keys.

Blockchain Properties.

Stake Contribution Property. At a high level, the sufficient stake contribution property states that after sufficiently many rounds, the total amount of proof-of-stake in mining the ℓ most recent blocks is at least β fraction of the total stake in the system.

Definition 35 (Sufficient Stake Contribution). *Let* $\mathsf{suf\text{-}stake\text{-}contr}$ *be the predicate such that* $\mathsf{suf\text{-}stake\text{-}contr}^{\ell}(\mathsf{VIEW}, \beta) = 1$ *iff for every round* $r \geq \ell$*, and each party* i *in* VIEW *such that* i *is honest at round* r *with blockchain* $\mathtt{B}$*, we have* $\mathsf{u\text{-}stakefrac}(\mathtt{B}, \ell) > \beta$*. A blockchain protocol* Γ *has* $(\beta(\cdot), \ell_0(\cdot))$*-sufficient stake contribution property with adversary* sA *in environment* $\mathcal{Z}$*, if there is a negligible function* $\mathsf{negl}(\cdot)$ *such that for any* $\lambda \in \mathbb{N}, \ell \geq \ell_0$*, it holds that,*

$$\Pr\left[\mathsf{suf\text{-}stake\text{-}contr}^{\ell}(\mathsf{VIEW}, \beta(\lambda)) = 1 \mid \mathsf{VIEW} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})\right] \geq 1 - \mathsf{negl}(\lambda)$$

Bounded Forking Property. Roughly speaking, the bounded forking property requires that no efficient adversary can create a sufficiently long fork so that its total amount of proof of stake is higher than a certain threshold. In more detail, it states that for property parameters α, ℓ_1, ℓ_2, the proof-of-stake fraction in the last ℓ_2 blocks in any adversarially created fork of length at least $\ell_1 + \ell_2$ should not be more than α.

Definition 36 (Bounded Stake Forking). *Let* $\mathsf{bd\text{-}stake\text{-}fork}$ *be the predicate such that* $\mathsf{bd\text{-}stake\text{-}fork}^{(\ell_1,\ell_2)}(\mathsf{VIEW}, \alpha) = 1$ *iff for any round* $r \geq \tilde{r}$ *and any pair of parties* i, j *in* VIEW *such that* i *is honest at round* r *with blockchain* $\mathtt{B}$ *and* j *is corrupt in round* $\tilde{r}$ *with blockchain* $\tilde{\mathtt{B}}$*, if there exists* $\ell' \geq \ell_1 + \ell_2$ *such that* $\tilde{\mathtt{B}}^{\lceil \ell'} \preceq \mathtt{B}$ *and for all* $\tilde{\ell} < \ell'$*,* $\tilde{\mathtt{B}}^{\lceil \ell'} \not\preceq \mathtt{B}$ *then* $\mathsf{u\text{-}stakefrac}(\tilde{\mathtt{B}}, \ell' - \ell_1) \leq \alpha$*. A blockchain protocol* Γ *has* $(\alpha(\cdot), \ell_1(\cdot), \ell_2(\cdot))$*-bounded forking property with adversary* sA *in environment* $\mathcal{Z}$*, if there is a negligible function* $\mathsf{negl}(\cdot)$ *and* $\delta(\cdot)$ *such that for any* $\lambda \in \mathbb{N}, \ell \geq \ell_1(\lambda), \tilde{\ell} \geq \ell_2(\lambda)$*, it holds that,*

$$\Pr\left[\mathsf{bd\text{-}stake\text{-}fork}^{(\ell,\tilde{\ell})}(\mathsf{VIEW}, \alpha(\lambda) + \delta(\lambda)) = 1 \mid \mathsf{VIEW} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})\right] \geq 1 - \mathsf{negl}(\lambda)$$

Distinguishable Forking. At a high level, distinguishable forking asserts that a sufficiently long sequence of blocks produced under honest protocol execution can consistently be *distinguished* from any fork generated adversarially. Formally, this concept can be defined as follows:

Definition 37 (Distinguishable Forking). *A blockchain protocol* Γ *satisfies* $(\alpha(\cdot), \beta(\cdot), \ell_1(\cdot), \ell_2(\cdot))$*-distinguishable forking with adversary* $\mathcal{A}$ *in environment* $\mathcal{Z}$*, if there is a negligible function* $\mathsf{negl}(\cdot)$ *and* $\delta(\cdot)$ *such that for any* $\lambda \in \mathbb{N}, \ell \geq \ell_1(\lambda), \tilde{\ell} \geq \ell_2(\lambda)$*, it holds that,*

$$\Pr\left[\begin{array}{l|l} \alpha(\lambda) + \delta(\lambda) < \beta(\lambda) \wedge & \\ \mathsf{suf\text{-}stake\text{-}contr}^{\tilde{\ell}}(\mathsf{VIEW}, \beta(\lambda)) = 1 \wedge & \mathsf{VIEW} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \mathsf{bd\text{-}stake\text{-}fork}^{(\ell,\tilde{\ell})}(\mathsf{VIEW}, \alpha(\lambda) + \delta(\lambda)) = 1 & \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Evolving Blockchains. To define Encryption with Public Self-Incriminating Proofs scheme (in Definition 6), we need to be able to non-interactively verify that a blockchain has evolved from a previous state such that the current state includes a certain message. In particular, we want to make sure that the initial chain $\mathtt{B}$ has "correctly" evolved into the final chain $\tilde{\mathtt{B}}$. A sufficiently long chain in an honest execution can be distinguished from a fork generated by the adversary by looking at the combined amount of stake proven in such a sequence of blocks. We encapsulate this property in a predicate called $\mathsf{evolved}(\cdot, \cdot)$ defined as follows.

Definition 38. *Let* $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ *be a blockchain protocol with validity predicate* V *and where the* $(\alpha, \beta, \ell_1, \ell_2)$*-distinguishable forking property (formally defined in Definition 37). Also, let* $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^{\lambda}, \mathsf{st})$ *and* $\tilde{\mathtt{B}} \leftarrow \mathsf{GetRecords}(1^{\lambda}, \tilde{\mathsf{st}})$*. We define an evolved predicate as a polynomial time function* $\mathsf{evolved}$ *that takes as input blockchains* $\mathtt{B}$ *and* $\tilde{\mathtt{B}}$ *and returns an indicator bit s.t.* $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1$ *if and only if all the following properties are satisfied: (i)* $V(\mathtt{B}) = V(\tilde{\mathtt{B}}) = 1$*, (ii)* $\mathtt{B}$ *and* $\tilde{\mathtt{B}}$ *are consistent i.e.,* $\mathtt{B}^{\lceil \kappa} \preceq \tilde{\mathtt{B}}$ *where* κ *is the common prefix parameter, and (iii) Let* $\ell' = |\tilde{\mathtt{B}}| - |\mathtt{B}|$ *then it holds that* $\ell' \geq \ell_1 + \ell_2$ *and* $\mathsf{u\text{-}stakefrac}(\tilde{\mathtt{B}}, \ell' - \ell_1) \geq \beta$*.*

NP-Relation for Proof Inclusion on an Evolving Blockchains Assume a blockchain protocol $\Gamma = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ with validity predicate V. We define a relation $\mathcal{R}_{\Gamma^V}$ that captures the fact that a valid signature σ on a reference message d generated under pk is included in the common prefix of a blockchain $\tilde{\mathtt{B}}$ that has evolved from an initial blockchain $\mathtt{B}$ via a valid execution of the protocol. This relation is formalized in Definition 39 below.

Definition 39 (NP-Relation for Proof Inclusion). *Let $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol with validity predicate V with the $(\alpha, \beta, \ell_1, \ell_2)$-distinguishable forking property (as in Definition 37) and associated predicate $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) \in \{0,1\}$ (as in Definition 38). Let $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be an EUF-CMA secure signature scheme and $d \in \{0,1\}^*$ is a reference message. We define relation $\mathcal{R}_{\Gamma^V}$ as follows:*

$$\mathcal{R}_{\Gamma^V} : \left\{ ((\underbrace{\mathsf{pk}, d, \mathtt{B}}_{\text{inst}}), (\underbrace{\sigma, \tilde{\mathtt{B}}}_{\text{wit}})) \;\middle|\; \begin{array}{l} 1 \leftarrow \mathsf{SIG.Vf}(\mathsf{pk}, d, \sigma) \wedge \mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1 \\ \wedge\, (\mathsf{pk}, d, \sigma) \in B^* \wedge B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2} \wedge \mathsf{u\text{-}stakefrac}(\tilde{\mathtt{B}}, \ell_2) \geq \beta \end{array} \right\}$$

Let $\mathcal{L}_{\Gamma^V}$ be the language specified by the relation $\mathcal{R}_{\Gamma^V}$. This language is in **NP** because verification of blockchains and signatures are polynomial time algorithms, as are the verification of the additional chain predicates in Definition 39.

A.9 Extended details on MPC-hardness

Security Games for Distributed Adversary. To formalize "knowledge" in the distributed adversary attack scenario, Dziembowski *et al.* [30] used the concept of a "knowledge extractor" proposed in [7]. Bellare and Rogaway [7] consider an adversary $\mathcal{A}$ with access to a function Fun (they assume that the Fun is a hash function; which is modeled as a random oracle). It is assumed that if an adversary $\mathcal{A}$ evaluated Fun on some input x, then $\mathcal{A}$ knows the input x and the corresponding output $\mathsf{Fun}(x)$. Technically, the (input, output) pairs are later given to an algorithm kEXT called "knowledge extractor". If kEXT outputs some message s, then we assume that "$\mathcal{A}$ knows s" (since $\mathcal{A}$ could have computed s by observing the oracle queries and corresponding replies).

Now, we define the information each party received as a result of the fast oracle queries at the end of the execution of a protocol: We define the local transcript of a party $\mathcal{A}_j$ to be the sequence τ_j of function inputs that $\mathcal{O}_{\mathsf{Fun}}$ received from $\mathcal{A}_j$ (in the same order in which they were received). Let τ_j^{fast} be the sub-sequence of τ_j containing only the inputs corresponding to fast queries (call it a local fast-function transcript of a party $\mathcal{A}_j$). A knowledge extractor kEXT is a deterministic poly-time machine that takes τ_j^{fast} as input and produces as output a finite set $\mathsf{kEXT}(\tau_j^{\mathsf{fast}}) \subset \{0,1\}^*$.

In the distributed adversary settings, Dziembowski *et al.* [30] use the concept of a knowledge extractor but slightly adjust it in the following way:

1. There is a knowledge extractor kEXT_j for each of the sub-adversaries $\mathcal{A}_j$.
2. Each such knowledge extractor kEXT_j takes as input the transcript of queries τ_j^{fast} that $\mathcal{A}_j$ has made to the oracle $\mathcal{O}_{\mathsf{Fun}}$ only in $\mathtt{mode} = \mathsf{fast}$. Queries made by $\mathcal{A}_j$ in $\mathtt{mode} = \mathsf{slow}$ (recall that these queries model MPC evaluations of Fun with a potentially unknown input) are not given to kEXT_j.
3. Finally, we say that an adversary $\mathcal{A}_p$ individually knows a secret s if there exists an efficient knowledge extractor kEXT_p such that $\mathsf{s} \in \mathsf{kEXT}_p(\tau_j^{\mathsf{fast}})$.

Detailed considerations of scratch We use the MPC-hard function $\mathsf{scratch}$ in our constructions, our goal is to extract the inputs to $\mathsf{scratch}$ from one of the transcripts τ_j^{fast} of fast queries made to $\mathcal{O}_{\mathcal{H}}$ by one of the sub-adversaries $\mathcal{A}_j$. To argue about this extractability, we derive Lemma 1 below from the proof of Theorem 1 in [30].

Now, before proceeding with the formal Lemma 1, let us comment on another parameter in the lemma statement. We define a party P as a η-bounded party if we bound the total number of fast queries made by the party P to the oracle $\mathcal{O}_{\mathcal{H}}$ in the online phase by η. Note that each computation of the $\mathsf{scratch}$ procedure requires $(nd+1)$ hashes $\mathcal{H}$. For a party, observe that each $\mathsf{scratch}$ attempt succeeds with probability $2^{-\zeta}$ (by "succeeding" we mean finding a value that starts with ζ zeros). Since the party needs to be successful β times, the party needs, on average, $\lfloor (nd+1) \cdot 2^{\zeta} \cdot \beta/(nd+1) \rfloor = 2^{\zeta} \cdot \beta$ $\mathsf{scratch}$ attempts. We set η to be the double of this parameter, *i.e.*, $\eta = \beta \cdot (nd+1) \cdot 2^{\zeta+1}$, to make the probability that the party is successful (refer to Definition 2 in [30] for the formal statement) less than β times exponentially small. Note that this budget allows the η-bounded party P to evaluate $\mathsf{scratch}$ $\lfloor \eta/(nd+1) \rfloor = \lfloor (nd+1) \cdot 2^{\zeta+1} \cdot \beta/(nd+1) \rfloor = 2^{\zeta+1} \cdot \beta$ times.

Setting the parameters of scratch appropriately we can de-facto prevent MPC attacks up a certain amount of time. Since key rolling/rotation within a few months to a few years [5] is the de-facto industry standard this allow us to heuristically prevent MPC-based attacks. We furthermore note that a second motivation on the heuristic security of MPC-hard functions is that the price of executing an MPC computation over a very long time-period will not be cheap in computation resources and thus may in many situations, be higher than the value of what parties might learn by executing an MPC protocol.

Lemma 1 (Derived from [30]). *Let* scratch *be the function defined in Figure 1 with parameters* $d, n \in \mathbb{N}$*, and also let* $\alpha, \beta, \zeta \in \mathbb{N}$ *be arbitrary parameters with* $\alpha \geq 2\beta$ *(where* ζ *can be a function of* β*) and a special oracle* $\mathcal{O}_{\mathcal{H}}$ *that allows for evaluating a fixed input-length hash function* $\mathcal{H} : \{0,1\}^{\alpha} \to \{0,1\}^{\beta}$*. Let* $\mathcal{A}_1, \ldots, \mathcal{A}_a$ *be a* (δ, Υ)*-distributed adversary where* $\delta \leq d-1$ *and* $\Upsilon \leq \beta \cdot 2^{\zeta-3}$*. Let* $\eta = \beta \cdot (nd+1) \cdot 2^{\zeta+1}$ *and* P *be a* η*-bounded party. For random* $s \in \{0,1\}^{n\cdot(\alpha-\beta-2)}$ *and* $z \in \{0,1\}^{\beta}$*, the following holds:*

1. *The* η*-bounded party* P *can compute* β *number of nonces* $w_1, \ldots, w_\beta \in \{0,1\}^{\alpha-\beta-2}$ *by accessing the oracle* $\mathcal{O}_{\mathcal{H}}$ *such that for all* $i \in [\beta]$ *the* $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ *has its first* ζ *bits equal to 0 except with negligible probability over the choice of* s, z*.*
2. *If* (δ, Υ)*-distributed adversary* $\mathcal{A}_1, \ldots, \mathcal{A}_a$ *can compute* β *number of nonces* $w_1, \ldots, w_\beta \in \{0,1\}^{\alpha-\beta-2}$ *such that for all* $i \in [1, \beta]$ *the* $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ *has its first* ζ *bits equal to 0 with access to an oracle* $\mathcal{O}_{\mathcal{H}}$*, then except with negligible probability over the choice of* s, z*, there exists an extractor* $\mathsf{kEXT}(\tau_j^{\mathsf{fast}})$ *that outputs* s *for at least one* $j \in [1, a]$*, where* τ_j^{fast} *is transcript of the fast hash queries made to* $\mathcal{O}_{\mathcal{H}}$ *by the sub-adversaries* $\mathcal{A}_j$*.*

Proof (Sketch). This lemma follows from Theorem 1 of [30], which proves the security of the Proof of Individual Knowledge scheme introduced in that work. Point 1 follows from the completeness of the Proof of Individual Knowledge. Intuitively, an η-bounded party for $\eta = \beta \cdot (nd+1) \cdot 2^{\zeta+1}$ can compute scratch a sufficient number of times to find such β number of nonces $w_1, \ldots, w_\beta$ by accessing the oracle $\mathcal{O}_{\mathcal{H}}$, except with negligible probability. Point 2 follows from the soundness of the Proof of Individual Knowledge, more specifically from the existence of an extractor that successfully extracts a malicious prover's witness from τ_j^{fast} for a sub-adversary $\mathcal{A}_j$ given a (δ, Υ)-distributed adversary $\mathcal{A}_1, \ldots, \mathcal{A}_a$ where $\delta \leq d-1$ and $\Upsilon \leq \beta \cdot 2^{\zeta-3}$ with access to $\mathcal{O}_{\mathcal{H}}$.

We refer to reader to Sec. 4 in [30] for a more detailed description, correctness, and security (MPC-hardness) of the scratch procedure.

B PSIPE based on Extractable Witness Encryption

In this section, we present a concrete construction of encryption with public self-incriminating proof $\mathsf{PSIPE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$, which we call $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$. Our construction is described formally in Figure 17. We require as setup a blockchain protocol $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ with validity predicate V (discussed in Sec. 2.2). We realize the notion of PSIPE from a signature scheme $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ (in Definition 16) and an extractable witness encryption scheme $\mathsf{eWE} = (\mathsf{Enc}, \mathsf{Dec})$ (in Definition 22) for the language $\mathcal{L}_{\Gamma^V}$ specified by the relation $\mathcal{R}_{\Gamma^V}$ (in Definition 39).

Although our construction works based on any eWE scheme, in Sec. B.1, we observe that since we assume a PoS blockchain as a setup, we can realize such a scheme under standard assumptions via the *extractable Witness Encryption on Blockchain (eWEB)* notion of [42] using techniques from [42] and [15].

Overview of our $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$. At a high level, the core idea is to force the decryptor to produce and publish on the blockchain a signature π on a reference signing message d for the message m, and the signature must be valid under a given public key pk in order to decrypt a ciphertext c.

To achieve this, we encrypt $msg = m \| d$ using a eWE scheme with respect to a statement $(\mathsf{pk}, d, \mathtt{B})$, where d is the reference signing message d for the message m, pk is the prescribed public key and B denotes the current state of the blockchain. This outputs a ciphertext $\hat{c}$ and finally defining our output PSIPE ciphertext as $c = (\hat{c}, d)$ and publish it on the blockchain.

To decrypt ciphertext $c = (\hat{c}, d)$, the decryptor must first generate a signature π on d, and then $\hat{c}$ can only be decrypted as $msg = m \| d$ by a party who has a witness $(\pi, \tilde{\mathtt{B}})$ where $\tilde{\mathtt{B}}$ denotes a future valid state of the blockchain that evolved from B containing π such that π verifies as a valid signature on d under pk. Therefore, the decryptor is forced to publish π in order to obtain a $\tilde{\mathtt{B}}$ that allows it to decrypt $\hat{c}$.

At the time of SIP verification, to verify that the SIP π is valid and ensure the consistency of $c = (\hat{c}, d)$, the verifier first extracts the witness $(\pi, \tilde{\mathtt{B}})$. Then, using this witness, the verifier decrypts $\hat{c}$ to obtain

Construction of PSIPE Scheme: $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$

Parameters: A security parameter λ.

Building-blocks: A blockchain protocol $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ (Definition 2) with validity predicate V, the $(\alpha, \beta, \ell_1, \ell_2)$-distinguishable forking property (Definition 37) and associated predicate $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) \in \{0,1\}$ (Definition 38). An extractable witness encryption scheme $\mathsf{eWE} = (\mathsf{Enc}, \mathsf{Dec})$ for the language $\mathcal{L}_{\Gamma V}$ specified by the relation $\mathcal{R}_{\Gamma V}$ (Definition 39). An EUF-CMA secure signature scheme $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$.

- $\mathsf{KGen}(1^\lambda)$:
  1. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$.
  2. Output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$:
  1. Run $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$.
  2. Sample a random $d \in \{0,1\}^\lambda$ and set $msg = m \| d$.
  3. $\hat{c} \leftarrow \mathsf{eWE.Enc}(1^\lambda, \mathcal{L}_{\Gamma V}, (\mathsf{pk}, d, \mathtt{B}), msg)$, where $(\mathsf{pk}, d, \mathtt{B}) \in \mathcal{L}_{\Gamma V}$.
  4. Publish the ciphertext on the blockchain Γ by $\mathsf{Broadcast}(1^\lambda, c = (\hat{c}, d))$
  5. Output $c = (\hat{c}, d)$.
- $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c = (\hat{c}, d))$:
  1. Compute the self-incriminating proof as $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, d)$.
  2. Publish π on the blockchain Γ by executing $\mathsf{Broadcast}(1^\lambda, (\mathsf{pk}, d, \pi))$.
  3. Run $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\tilde{\mathtt{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$ until the message (pk, d, π) appears in a block $B^* \in \tilde{\mathtt{B}}$ of blockchain $\tilde{\mathtt{B}}$ such that the chain extends B^* by $\ell_1 + \ell_2$ block.
  4. Decrypt $\hat{c}$ using π and $\tilde{\mathtt{B}}$, obtaining $msg \leftarrow \mathsf{eWE.Dec}(\hat{c}, (\pi, \tilde{\mathtt{B}}))$.
  5. Parse msg as $m \| d$ and output (m, π).
- $\mathsf{Vf}(\mathsf{pk}, c = (\hat{c}, d), \pi)$:
  1. Run $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\tilde{\mathtt{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$.
  2. If c is *not* the first ciphertext with d to appear on the ledger, then return 0.
  3. Decrypt $\hat{c}$ using π and $\tilde{\mathtt{B}}$, obtaining $msg' \leftarrow \mathsf{eWE.Dec}(\hat{c}, (\pi, \tilde{\mathtt{B}}))$[a].
  4. Parse msg' as $m' \| d'$ and, if $d' \neq d$, then return 0.
  5. Output $\mathsf{SIG.Vf}(\mathsf{pk}, d, \pi)$.
- $\mathsf{ProofExt}(\mathsf{pk}, c = (\hat{c}, d))$:
  1. Run $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\tilde{\mathtt{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$.
  2. Find a block $B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2}$ containing a record $(\mathsf{pk}, d, \pi) \in B^*$ such that $1 \leftarrow \mathsf{SIG.Vf}(\mathsf{pk}, d, \pi)$ and output π.
  3. Otherwise, if no such record exists in any block $B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2}$, output $\perp$.

[a] As observed in Remark 2, it is trivial to add an extra layer of encryption that prevents leaking the plaintext message when a SIP is produced.

Fig. 17: Construction of PSIPE based on eWE: $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$.

$msg = m' | d'$. Finally, it verifies the SIP π and checks whether d' is consistent with d and c is the first ciphertext with d to appear on the ledger.

We build on a Proof-of-Stake blockchain as it has been shown that it is possible to non-interactively verify whether a blockchain $\tilde{\mathtt{B}}$ evolved from a previous blockchain $\mathtt{B}$ via an honest protocol execution [41] and ensure that the protocol cannot be abused to decrypt a ciphertext without publishing a SIP.

Remark 2 (Stronger Security $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$). In our proposed protocol $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ shown in Figure 17, once a valid decryptor decrypts a ciphertext, anyone within the blockchain system can subsequently decrypt it and retrieve the original message. This limitation reduces the practical applicability of the construction. However, we can easily address this issue using the approach described below, ensuring that *only* the intended decryptor is able to obtain the original message.

To encrypt a message m, an encryptor first encrypts the message as $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ using an IND-CCA2 secure encryption scheme (*e.g.,* [21]), and then it encrypt $msg = \mathsf{ct} \| d$ using a eWE scheme with respect to a statement $(\mathsf{pk}, d, \mathtt{B})$, where d is the reference signing message d for the message m, pk is the prescribed public key and $\mathtt{B}$ denotes the current state of the blockchain. This outputs a ciphertext $\hat{c}$ and and our PSIPE ciphertext is then defined as $c = (\hat{c}, d)$ and publish it on the blockchain.

To decrypt ciphertext $c = (\hat{c}, d)$, the decryptor must first generate a signature π on d, and then $\hat{c}$ can only be decrypted as $msg = \mathsf{ct} \| d$ by a party who has a witness $(\pi, \tilde{\mathtt{B}})$ where $\tilde{\mathtt{B}}$ denotes a future valid state

of the blockchain that evolved from B containing π such that π verifies as a valid signature on d under pk. Therefore, the decryptor is forced to publish π in order to obtain a $\tilde{\mathsf{B}}$ that allows it to decrypt $\hat{c}$. Finally, *only* the decryptor can decrypt the ciphertext ct and obtain the original message as $m \leftarrow \mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$. The SIP verification and proof extraction process is same as described in Figure 17.

B.1 Instantiating $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ from Standard Assumptions

Our construction of $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ in Figure 17 can be instantiated from any extractable witness encryption (eWE) scheme. However, known eWE constructions with support for the language we need are based on non-standard and very strong assumptions (*e.g.* iO). Hence, in order to obtain a concrete instantiation of $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ from standard assumptions, we take advantage of the fact that we already use a blockchain-based public ledger in $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ and employ the notion of extractable Witness Encryption on a Blockchain (eWEB) [42].

It has been shown in [42] that a flavor of (extractable) witness encryption can be realized using a Proof-of-Stake (PoS) blockchain ledger as setup, which we already do in $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$. This notion is called extractable Witness Encryption on a Blockchain (eWEB) and provides the same functionality as a regular extractable WE scheme, provided that the parties executing the eWEB scheme have access to the underlying PoS ledger. The main idea of the eWEB construction of [42] is to use dynamic proactive secret sharing to store the encrypted message in such a way that it can be re-shared towards new committees as parties join and leave the PoS blockchain protocol execution. When a party who knows a witness to the instance under which a ciphertext was generated wants to decrypt it, they publish a NIZKPoK of that witness, which allows the committee to verify whether the party indeed knows the witness (also allowed the simulator to extract this witness). Extractable privacy for eWEB holds given that the majority of the committee is honest, and thus refuses to help a party reconstruct an encrypted message unless it publishes such a valid NIZKPoK.

In order to meaningfully employ such an eWEB scheme in instantiating our $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ construction, we must prevent dynamic proactive secret sharing committees from leaking messages encrypted under eWEB without being detected, which would circumvent the need to publish a self-incriminating proof. Notice that we cannot prevent such a committee from leaking a message, but in our case it is sufficient that this leakage is detected in public if it happens. We achieve this property via the techniques of [15] by storing each message encrypted under eWEB as shares held by dynamic *anonymous* committees chosen at random. Since each committee is anonymous, even an adaptive adversary does not know which parties to corrupt to take control of a committee (as in the YOSO model [35]). We observe that this construction can be instantiated with the efficient publicly verifiable secret sharing scheme for random anonymous committees presented in [17], which also allows for the secret to be periodically re-shared towards a newly selected dynamic anonymous committee.

The key observation of [15], is that since each secret message is held by a different anonymous committee chosen at random, adversarial committee members cannot leak the secret without communicating in public (*e.g.,* announcing their shares, or their willingness to leak shares). Hence, we can modify the eWEB construction of [42] to employ such publicly verifiable secret sharing with randomly chosen dynamic anonymous committees to store messages encrypted under eWEB, instead of using standard dynamic proactive secret sharing. Notice that this only modifies the encryption step of the construction from [42], requiring encryptors to use this alternative secret sharing scheme, while the decryption remains the same. As observed in [15], instead of requiring a new committee to hold shares of each encrypted message, this solution can be instantiated by threshold encrypting under a public key whose corresponding secret key shares are held by randomly chosen anonymous committees, who re-share this secret key towards new committees whenever a decryption happens or when parties leave the protocol execution (as in YOSO threshold encryption scheme [9]).

Notice that even when randomly chosen dynamic anonymous committees are employed, an attacker may still offer to bribe committee members to leak their shares. Such a bribe proposition can be publicized by the attacker, who is then contacted privately by each opportunistic committee member. This sort of attack can be thwarted in our setting by choosing larger committees in way that providing such bribes to sufficiently many committee members becomes economically infeasible. Analysing such incentive structures is beyond the scope of this work. Providing such an analysis as well as alternative constructions of eWEB that offer better resilience against such attacks is left for future works.

B.2 Security Analysis

We formally state the security of $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ in Theorem 3.

Theorem 3. *Assuming that: (i) Γ is a blockchain protocol (as in Definition 2) with validity predicate V, the $(\alpha, \beta, \ell_1, \ell_2)$-distinguishable forking property (as in Definition 37) and associated predicate $\mathsf{evolved}(\mathsf{B}, \tilde{\mathsf{B}}) \in \{0,1\}$ (as in Definition 38), (ii) eWE is an extractable witness encryption scheme (as in Definition 22) for the language $\mathcal{L}_{\Gamma^V}$ specified by the relation $\mathcal{R}_{\Gamma^V}$ (Definition 39), (iii) SIG is a EUF-CMA secure signature scheme as per Definition 16. Then our protocol $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ in Figure 17 is a secure encryption with public self-incriminating proof scheme PSIPE as per Definition 6.*

Proof. We prove the Theorem 3 by showing a proof for Correctness, Unforgeability, IND-CPA Security, and Public Self-Incriminating Proof properties of our scheme $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ as follows:

Correctness. The correctness of $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ is immediate and can be proven by the correctness of the underlying primitives.

Fix λ, ℓ_1, ℓ_2, and β and a correct blockchain protocol Γ with validity predicate V as described in Sec. 2.2. Let $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}.\mathsf{KGen}(1^\lambda)$ as $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$ and for any message $m \in \{0,1\}^\lambda$, we encrypt the message by $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}.\mathsf{Enc}(\mathsf{pk}, m)$ as $\hat{c} \leftarrow \mathsf{eWE.Enc}(1^\lambda, \mathcal{L}_{\Gamma^V}, (\mathsf{pk}, d, \mathsf{B}), msg = m \| d)$ where d is a reference signing message for the message m, and $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\mathsf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$. Finally, it publish the PSIPE ciphertext $c = (\hat{c}, d)$ on the blockchain Γ by executing $\mathsf{Broadcast}(1^\lambda, c = (\hat{c}, d))$ and output $c = (\hat{c}, d)$.

For decrypting a ciphertext $c = (\hat{c}, d)$ by $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}.\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$, a decryptor first needs to generate a self-incriminating proof as $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, d)$, and then run the $\mathsf{Broadcast}$ algorithm to post (pk, d, π) on the blockchain Γ. Let $\tilde{\mathsf{st}}$ be the local state of the decryptor after message (pk, d, π) is posted on the blockchain and it is extended by $\ell_1 + \ell_2$ blocks. At this point, it holds that $\mathsf{evolved}(\mathsf{B}, \tilde{\mathsf{B}}) = 1$ and that there exists a block $B^* \in \tilde{\mathsf{B}}^{\lceil \ell_1 + \ell_2}$ such that $(\mathsf{pk}, d, \sigma) \in B^*$, so that with all but negligible probability, $\tilde{\mathsf{B}}$ and π can be used as the witness to decrypt ciphertexts $\hat{c}$ as $msg = m' \| d' \leftarrow \mathsf{eWE.Dec}(\hat{c}, (\pi, \tilde{\mathsf{B}}))$ where $((\mathsf{pk}, d, \mathsf{B}), (\pi, \tilde{\mathsf{B}})) \in \mathcal{R}_{\Gamma^V}$. Therefore, $msg = m' \| d' \leftarrow \mathsf{eWE.Dec}(c_2, (\pi, \tilde{\mathsf{B}}))$ follows from the correctness of the extractable witness encryption scheme as per Definition 22, $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, d)$ follows from the correctness of the signature scheme as per Definition 16 and finally check the consistency of d' with d and check c is the first ciphertext with d to appear on the ledger. Therefore, $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}eWE}}$ satisfies the correctness condition.

Unforgeability. Assume, for the sake of contradiction, that we have an adversary $\mathcal{A}_{\mathsf{PSIPE}}$ that wins the game $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$ with non-negligible advantage when executing PSIPE from Figure 17. We then show how to use $\mathcal{A}_{\mathsf{PSIPE}}$ to construct another adversary $\mathcal{A}_{\mathsf{SIG}}$ with black-box access to $\mathcal{A}_{\mathsf{PSIPE}}$ which breaks the unforgeability of the signature scheme SIG, *i.e.,* $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG}, \mathcal{A}_{\mathsf{SIG}}}$ (defined in Figure 11), with asymptotically similar advantage.

We construct an adversary $\mathcal{A}_{\mathsf{SIG}}$, who is talking with the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG}, \mathcal{A}_{\mathsf{SIG}}}$ and an internal copy of $\mathcal{A}_{\mathsf{PSIPE}}$ for which it simulates $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$. The adversary $\mathcal{A}_{\mathsf{SIG}}$ proceeds as follows:

1. $\mathcal{A}_{\mathsf{SIG}}$ receives $(1^\lambda, \mathsf{pk})$ from the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG}, \mathcal{A}_{\mathsf{SIG}}}$ and forwards this to $\mathcal{A}_{\mathsf{PSIPE}}$, pretending to be the challenger of the $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$ game.
2. Playing the role of challenger in $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$ then $\mathcal{A}_{\mathsf{SIG}}$ receives back from $\mathcal{A}_{\mathsf{PSIPE}}$ the value (c', π') which has a non-negligible advantage in winning $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$.
3. Letting $c' = (\hat{c}, d)$ then $\mathcal{A}_{\mathsf{SIG}}$ returns $(m', \sigma') = (d, \pi')$ to the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG}, \mathcal{A}_{\mathsf{SIG}}}$.

First observe that $(\mathsf{pk}, \cdot) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$, hence the pair $(1^\lambda, \mathsf{pk})$ that $\mathcal{A}_{\mathsf{SIG}}$ receives from $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG}, \mathcal{A}_{\mathsf{SIG}}}$ is similarly distributed to the pair that $\mathcal{A}_{\mathsf{PSIPE}}$ receive from the *real* challenger in the $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$ game. Now see that $c' = (\hat{c}, d)$ for which $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, d)$ with $(\sigma = \pi) = \pi'$ with non-negligible probability. Hence (m', σ') will be a valid output with similar probability.

IND-CPA Security. Assume by contradiction that there exists an adversary $\mathcal{A}_{\mathsf{PSIPE}}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$ for our PSIPE. We will show that this $\mathcal{A}_{\mathsf{PSIPE}}$ can be used to construct adversaries breaking the extractable security of the underlying extractable witness encryption scheme eWE or unforgeability of the underlying signature scheme SIG.

We construct an adversary $\mathcal{A}$ who is talking with the challenger of $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE}, \mathcal{A}}$ and $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG}, \mathcal{A}}$, and an internal copy of $\mathcal{A}_{\mathsf{PSIPE}}$ for which it simulates $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE}, \mathcal{A}_{\mathsf{PSIPE}}}$. Throughout this reduction, $\mathcal{A}$ acts as $\mathcal{Z}$ in the execution of the blockchain protocol Γ, which it simulates towards $\mathcal{A}_{\mathsf{PSIPE}}$ following the same steps as the real protocol. The adversary $\mathcal{A}$ proceeds as follows:

1. $\mathcal{A}$ receives $(1^\lambda, \mathsf{pk})$ from the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}}$ and forwards $(1^\lambda, \mathsf{pk})$ to $\mathcal{A}_{\mathsf{PSIPE}}$, acting as the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$.
2. $\mathcal{A}$ receives a tuple (m_0, m_1) from $\mathcal{A}_{\mathsf{PSIPE}}$ and sets $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B}) \in \mathcal{L}_{\Gamma^V}$ where $d \xleftarrow{\$} \{0,1\}^\lambda$, and $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$. $\mathcal{A}$ forwards the tuple $(\cdot, \mathsf{inst}, m_0, m_1)$ to the challenger of $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$.
3. $\mathcal{A}$ receives the challenge ciphertext c'_b from the challenger of $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$. Then $\mathcal{A}$ forwards $c_b = (\hat{c}, d)$ to $\mathcal{A}_{\mathsf{PSIPE}}$, where $d = d$ and $\hat{c} = c'_b$.
4. $\mathcal{A}$ receives a guess b' from $\mathcal{A}_{\mathsf{PSIPE}}$.
5. $\mathcal{A}$ executes the eWE extractor (as defined in the extractable security property) and obtains $\mathsf{wit} = (\pi, \tilde{\mathtt{B}}) \leftarrow \mathsf{EXT}^{\mathcal{A}_{\mathsf{PSIPE}}(\cdot)}(1^\lambda, \mathcal{L}_{\Gamma^V}, \mathsf{inst}, m_0, m_1)$.
6. Finally, $\mathcal{A}$ forwards the guess b' to the challenger of $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$ and $(m', \sigma') = (d, \pi)$ to the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}}$.

Notice that $\mathcal{A}$ simulates $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ exactly as in a real execution. Now assume that $\mathcal{A}_{\mathsf{PSIPE}}$ has non-negligible advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$, then $\mathcal{A}$ is able to distinguish extractable witness encryption ciphertexts c_0, c_1 generated under the statement $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B})$ such that $\mathsf{inst} \in \mathcal{L}_{\Gamma^V}$ from messages m_0, m_1. Hence, $\mathcal{A}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$, which means given extractable security for the eWE scheme there is an extractor EXT that obtains $\mathsf{wit} = (\pi, \tilde{\mathtt{B}})$ from $\mathcal{A}_{\mathsf{PSIPE}}$, where $\mathsf{SIG.Vf}(\mathsf{pk}, d, \pi) = 1$. Notice that π is a valid signature forgery on d, since $\mathcal{A}_{\mathsf{PSIPE}}$ does not have the signing key corresponding to pk. Hence, if $\mathcal{A}_{\mathsf{PSIPE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ and extractable security holds for the eWE scheme, then $\mathcal{A}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}}$. On the other hand, if $\mathcal{A}_{\mathsf{PSIPE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ but EXT fails to output such $(\pi, \tilde{\mathtt{B}})$ with non-negligible probability, we contradict the extractable security property of the eWE scheme. Hence, given that eWE has extractable security and that SIG is EUF-CMA secure, we have that $\mathcal{A}_{\mathsf{PSIPE}}$ can only have negligible advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}}$.

Public Self-Incriminating Proof. Assume by contradiction that an adversary $\mathcal{A}_{\mathsf{PSIPE}}$ exists which can win the $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ game with a non-negligible advantage $\mathsf{Adv}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ while extractor $\mathsf{EXT}(\mathsf{pk}, c_b, \tilde{\mathtt{B}})$ obtains π such that $\mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1$ with probability $\mathsf{Succ}^{\mathsf{PSIPE}}_{\mathsf{EXT}} < \mathsf{Adv}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma} - \mathsf{negl}(\lambda)$. We argue that if this is the case, then it must be because either $\mathcal{A}_{\mathsf{PSIPE}}$ breaks the underlying blockchain protocol's distinguishable forking and common prefix properties, or the adversary breaks the extractable security of the extractable witness encryption scheme eWE. This leads to two distinct cases:

1. Assuming that the extractable witness encryption scheme is secure, if a valid π does not appear on the common prefix of an honest party's blockchain but $\mathcal{A}_{\mathsf{PSIPE}}$ is able to produce $\tilde{\mathtt{B}}$ such that $1 \leftarrow \mathsf{SIG.Vf}(\mathsf{pk}, d, \pi) \wedge \mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1 \wedge (\mathsf{pk}, d, \pi) \in B^* \wedge B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2} \wedge \mathsf{u\text{-}stakefrac}(\tilde{\mathtt{B}}, \ell_2) \geq \beta$ (*i.e.,* $((\pi, \tilde{\mathtt{B}}), \mathsf{inst}) \in \mathcal{R}_{\Gamma^V}$ where $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B})$ and $\mathsf{inst} \in \mathcal{L}_{\Gamma^V}$), then $\mathcal{A}_{\mathsf{PSIPE}}$ is breaking the distinguishable forking and common prefix properties of the blockchain protocol Γ.
2. Assuming that the blockchain protocol is secure, if the adversary $\mathcal{A}_{\mathsf{PSIPE}}$ distinguishes the ciphertext c_0, c_1 with a $\mathtt{B}$ such that $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B}) \in \mathcal{L}_{\Gamma^V}$ and a valid $\tilde{\mathtt{B}}$ evolved from $\mathtt{B}$ containing a valid π, then there exists an extractor EXT can extract the π or $\mathcal{A}_{\mathsf{PSIPE}}$ breaks the extractable security of the extractable witness encryption scheme.

We first reason about case 1. Notice that if $\mathcal{A}_{\mathsf{PSIPE}}$ successfully distinguishes the ciphertext c_0, c_1 without allowing for the extractor to obtain a valid SIP π and without violating the extractable security of the extractable witness encryption scheme, $\mathcal{A}_{\mathsf{PSIPE}}$ must obtain a valid witness for the following relation:

$$\mathcal{R}_{\Gamma^V} : \left\{ ((\underbrace{\mathsf{pk}, d, \mathtt{B}}_{\mathsf{inst}}), (\underbrace{\sigma, \tilde{\mathtt{B}}}_{\mathsf{wit}})) \middle| \begin{array}{l} 1 \leftarrow \mathsf{SIG.Vf}(\mathsf{pk}, d, \sigma) \wedge \mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1 \\ \wedge (\mathsf{pk}, d, \sigma) \in B^* \wedge B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2} \wedge \mathsf{u\text{-}stakefrac}(\tilde{\mathtt{B}}, \ell_2) \geq \beta \end{array} \right\}$$

However, while obtaining a valid π is trivial for $\mathcal{A}_{\mathsf{PSIPE}}$ since it holds sk, $\mathcal{A}_{\mathsf{PSIPE}}$ must obtain a $\tilde{\mathtt{B}}$ that satisfies the relation without resulting in a blockchain execution where a valid SIP π is present in the common prefix of every honest party's blockchain at the moment of decryption. In order to do so, $\mathcal{A}_{\mathsf{PSIPE}}$ must produce a valid execution of the blockchain starting from the initial blockchain $\mathtt{B}$ used to generate the ciphertext c_b and arriving at an evolved blockchain $\tilde{\mathtt{B}}$ such that $1 \leftarrow \mathsf{SIG.Vf}(\mathsf{pk}, d, \pi) \wedge \mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1 \wedge (\mathsf{pk}, d, \pi) \in B^* \wedge B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2} \wedge \mathsf{u\text{-}stakefrac}(\tilde{\mathtt{B}}, \ell_2) \geq \beta$ while preventing π from appearing in the common

prefix of an honest party's blockchain given the view of $\mathcal{A}_{\mathsf{PSIPE}}$. However, since this contradicts the common prefix property for the underlying blockchain protocol, $\mathcal{A}_{\mathsf{PSIPE}}$ can only hope to produce $\tilde{\mathtt{B}}$ locally, without actually executing the blockchain protocol, in such a way that π never appears in the common prefix of the blockchain (*i.e.,* so that the extractor fails when run on the blockchain obtained by honest parties). However, this would violate the distinguishable forking property of the blockchain protocol, since $\mathcal{A}_{\mathsf{PSIPE}}$ would need to obtain a $\tilde{\mathtt{B}}$ that does not contain a valid π but that satisfies $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1$. Hence, we conclude that $\mathcal{A}_{\mathsf{PSIPE}}$ is not able to produce a blockchain execution such that $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B})$ and $\mathsf{inst} \in \mathcal{L}_{\Gamma^V}$ without allowing the extractor to obtain a valid π, except with the negligible probability that $\mathcal{A}_{\mathsf{PSIPE}}$ breaks the distinguishable forking or common prefix properties of the underlying blockchain protocol Γ^V. This leaves us with case 2, where $\mathcal{A}_{\mathsf{PSIPE}}$ is able to distinguish extractable witness encryption ciphertexts c_0, c_1 for $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B}) \in \mathcal{L}_{\Gamma^V}$ without producing a valid witness $(\pi, \tilde{\mathtt{B}})$, which is ruled out by this reasoning.

To tackle case 2, we consider an adversary $\mathcal{A}_{\mathsf{PSIPE}} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5)$ that has non-negligible advantage in $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ but does not publish π on the blockchain, as that is ruled out by case 1. We construct an adversary $\mathcal{A}_{\mathsf{eWE}}$, using black-box access to $\mathcal{A}_{\mathsf{PSIPE}}$, which breaks the extractable security of eWE. Throughout this reduction, $\mathcal{A}_{\mathsf{eWE}}$ acts as $\mathcal{Z}$ in the execution of the blockchain protocol Γ, which it simulates towards $\mathcal{A}_{\mathsf{PSIPE}}$ following the same steps as the real protocol. Specifically, $\mathcal{A}_{\mathsf{eWE}}$ proceeds as follows:

1. $\mathcal{A}_{\mathsf{eWE}}$ starts $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ game acting as the challenger towards $\mathcal{A}_{\mathsf{PSIPE}}$, simulating an execution of the blockchain protocol $\mathsf{EXEC}^{\Gamma}(\mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk}), \mathcal{Z}, 1^\lambda)$ where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$. $\mathcal{A}_{\mathsf{eWE}}$ proceeds exactly as $\mathcal{Z}$ would until $\mathcal{A}_1$ stops and obtains a VIEW of the execution.
2. $\mathcal{A}_{\mathsf{eWE}}$ executes $\mathcal{A}_2(1^\lambda, \mathsf{pk}, \mathsf{sk}, \mathsf{VIEW}_{\mathcal{A}_1})$ where $\mathsf{VIEW}_{\mathcal{A}_1} \in \mathsf{VIEW}$ to obtain $(\mathsf{st}_1, m_0, m_1)$.
3. $\mathcal{A}_{\mathsf{eWE}}$ sets $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B})$, where $d \xleftarrow{\$} \{0,1\}^\lambda$, and $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$, given $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$ obtained from $\mathcal{A}_2$'s view $\mathsf{VIEW}'_{\mathcal{A}_2} \leftarrow \mathsf{VIEW}'$ of the simulated blockchain protocol execution. $\mathcal{A}_{\mathsf{eWE}}$ sends $(\cdot, \mathsf{inst}, m_0, m_1)$ to the challenger of the $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}_{\mathsf{eWE}}}$ game.
4. When $\mathcal{A}_{\mathsf{eWE}}$ receives c_b^{eWE} from the challenger of $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}_{\mathsf{eWE}}}$, it sets $c_b = (c_b^{\mathsf{eWE}}, d)$ and resumes $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$. $\mathcal{A}_{\mathsf{WE}}$ executes $\mathsf{st}_2 \leftarrow \mathcal{A}_3(\mathsf{st}_1, c_b)$ and resumes the blockchain protocol execution $\mathsf{EXEC}^{\Gamma}(\mathcal{A}_4(\mathsf{st}_2, c_b), \mathcal{Z}, 1^\lambda)$ acting exactly as $\mathcal{Z}$ until $\mathcal{A}_4$ stops, obtaining VIEW'.
5. $\mathcal{A}_{\mathsf{eWE}}$ executes $\mathcal{A}_5(\mathsf{st}_2, \mathsf{VIEW}'_{\mathcal{A}_4})$ where $\mathsf{VIEW}'_{\mathcal{A}_4} \leftarrow \mathsf{VIEW}'$, obtaining output b', which it returns to the challenger in the $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$ game.

Notice that $\mathcal{A}_{\mathsf{eWE}}$ simulates $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ and the blockchain execution towards $\mathcal{A}_{\mathsf{PSIPE}}$ exactly as in a real execution. Now assume that $\mathcal{A}_{\mathsf{PSIPE}}$ has non-negligible advantage $\mathsf{Adv}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ in $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$, then $\mathcal{A}_{\mathsf{eWE}}$ is able to distinguish extractable witness encryption ciphertexts c_0, c_1 generated under the statement $\mathsf{inst} = (\mathsf{pk}, d, \mathtt{B}) \in \mathcal{L}_{\Gamma^V}$ from messages m_0, m_1. Hence, $\mathcal{A}_{\mathsf{eWE}}$ has non-negligible advantage $\mathsf{Adv}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$ in $\mathsf{Game}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathcal{A}}$, which means given extractable security for the eWE scheme there is an extractor $\mathsf{EXT}^{\mathcal{A}_{\mathsf{PSIPE}}(\cdot)}(1^\lambda, \mathcal{L}_{\Gamma^V}, \mathsf{inst}, m_0, m_1)$ that obtains $\mathsf{wit} = (\pi, \tilde{\mathtt{B}})$ from $\mathcal{A}_{\mathsf{PSIPE}}$ with probability $\mathsf{Succ}^{\mathsf{EXT\text{-}SEC}}_{\mathsf{eWE},\mathsf{EXT}}$, where$(\mathsf{wit}, \mathsf{inst}) \in \mathcal{R}_{\Gamma^V}$. Otherwise, if $\mathcal{A}_{\mathsf{PSIPE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ while the extractor $\mathsf{EXT}(\mathsf{pk}, c_b, \tilde{\mathtt{B}})$ obtains SIP π such that $\mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1$ with negligible probability $\mathsf{Succ}^{\mathsf{PSIPE}}_{\mathsf{EXT}}$ (*i.e.,* fails to output such SIP π with non-negligible probability), we contradict the extractable security property of the eWE scheme. Hence, given that the blockchain protocol is secure and eWE has extractable security, we have that $\mathcal{A}_{\mathsf{PSIPE}}$ can only have negligible advantage $\mathsf{Adv}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ in $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ while EXT fails to output SIP π.

C Security Proofs

C.1 Proof of Theorem 1 ($\Pi_{\mathsf{PSIPE\text{-}TBIBE}}$ Security)

Proof. We prove the Theorem 1 by showing a proof for Correctness, Unforgeability, IND-CPA Security, and Public Self-Incriminating Proof properties of our scheme $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ as follows:

Correctness. The correctness of our $\mathbf{\Pi}_{\mathsf{PSIPE\text{-}TBIBE}}$ scheme follows immediately from the correctness guarantees of the underlying blockchain protocol, public key encryption, signature scheme and thresholdizable batched IBE.

A well-formed $\mathsf{PSIPE\text{-}TBIBE}$ ciphertext consists of a PKE and TBIBE encryption layers. Whilst the decryptor holds the sk to decrypt the inner PKE layer, the outer layer requires TBIBE decryption keys

for the id corresponding to the ciphertext instance. Given a valid SIP for ciphertext encrypted to id in the common-prefix of the blockchain, the blockchain protocol (as extended in Fig. 5) ensures that TBIBE decryption keys for id will be broadcast by a blockchain validator committee holding sharings of the TBIBE master keys, permitting correct recovery of the plaintext by the decryptor, and successful public SIP recovery during proof extraction.

Unforgeability. We construct an adversary $\mathcal{A}_{\mathsf{SIG}}$ from $\mathcal{A}_{\mathsf{PSIPE}}$ with non-negligible advantage to win $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ to break unforgeability of the signature scheme SIG. This adversary $\mathcal{A}_{\mathsf{SIG}}$ proceeds as follows:

1. $\mathcal{A}_{\mathsf{SIG}}$ receives $(1^\lambda, \mathsf{pk})$ from the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}_{\mathsf{SIG}}}$ and forwards this to $\mathcal{A}_{\mathsf{PSIPE}}$, acting as the challenger of the $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ game.
2. Playing the role of challenger in $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$, $\mathcal{A}_{\mathsf{SIG}}$ then receives back from $\mathcal{A}_{\mathsf{PSIPE}}$ the tuple (ct, π) which has a non-negligible advantage in winning $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$.
3. Letting $\mathsf{ct} = (\hat{c}, d)$ then $\mathcal{A}_{\mathsf{SIG}}$ returns $(m, \sigma) = (\mathsf{ct}, \pi)$ to the challenger of $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}_{\mathsf{SIG}}}$.

First observe that $(\mathsf{pk}, \cdot) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$, hence the pair $(1^\lambda, \mathsf{pk})$ that $\mathcal{A}_{\mathsf{SIG}}$ receives from $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIG},\mathcal{A}_{\mathsf{SIG}}}$ is similarly distributed to the pair that $\mathcal{A}_{\mathsf{PSIPE}}$ receive from the *real* challenger in the $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ game. Now see that $\mathsf{ct} = (\hat{c}, \mathsf{id})$ and $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{pk}, \mathsf{sk}, \mathsf{ct})$ are returned from $\mathcal{A}_{\mathsf{PSIPE}}$ with non-negligible probability. Hence $(m = \mathsf{ct}, \sigma = \pi)$ will be a verifying message, signature pair with similar probability.

IND-CPA. We construct an adversary $\mathcal{A}_{\mathsf{Enc}}$ from $\mathcal{A}_{\mathsf{PSIPE}}$ with non-negligible advantage to win $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ to break IND-CPA security of PKE. This adversary $\mathcal{A}_{\mathsf{Enc}}$ proceeds as follows:

1. $\mathcal{A}_{\mathsf{PKE}}$ receives $\mathsf{pk}_{\mathsf{PKE}}$ from the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}}$, and generates $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KGen}(\lambda)$ and $(\mathsf{pk}_{\mathsf{TBIBE}}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]}, \{\mathsf{msk}_{\mathsf{TBIBE},i}\}_{i\in[n]}) \leftarrow \mathsf{TBIBE.KGen}(\mathsf{pp})$ and forwards $\mathsf{pk} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{pk}_{\mathsf{SIG}})$, $\mathsf{pk}_{\mathsf{TBIBE}}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]}$ and $\mathsf{msk}_{\mathsf{TBIBE},i\in\mathcal{A}}$ to $\mathcal{A}_{\mathsf{PSIPE}}$, acting as the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$.
2. $\mathcal{A}_{\mathsf{PKE}}$ receives a tuple (m_0, m_1) from $\mathcal{A}_{\mathsf{PSIPE}}$ and forwards this plaintext pair to the challenger of $\mathsf{Game}_{\mathsf{TBIBE},\mathcal{A}}$.
3. $\mathcal{A}_{\mathsf{PKE}}$ receives a ciphertext c from $\mathcal{A}_{\mathsf{PKE}}$, samples $\mathsf{id} \xleftarrow{\$} \mathcal{I}$ and encrypts $\hat{\mathsf{c}} \leftarrow \mathsf{TBIBE.Enc}(\mathsf{pk}_{\mathsf{TBIBE}}, \mathsf{c}, \mathsf{id})$. $\mathcal{A}_{\mathsf{PKE}}$ forwards this to $\mathcal{A}_{\mathsf{PSIPE}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$.
4. Finally, $\mathcal{A}_{\mathsf{PKE}}$ receives a bit from $\mathcal{A}_{\mathsf{PSIPE}}$ which it forwards to the challenger in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}_{\mathsf{PKE}}}$.

Observe that the same plaintext messages (m_0, m_1) are encrypted in both black-box $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ and $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}_{\mathsf{PKE}}}$ instances. Thus, the advantage of $\mathcal{A}_{\mathsf{PSIPE}}$ winning $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ directly translates to an advantage of $\mathcal{A}_{\mathsf{PKE}}$ winning $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE},\mathcal{A}_{\mathsf{PKE}}}$.

Public Self Incrimination. Assume by contradiction that an adversary $\mathcal{A}_{\mathsf{PSIPE}}$ exists which can win the $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ game with a non-negligible advantage $\mathsf{Adv}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ while extractor $\mathsf{EXT}(\mathsf{pk}, c_b, \tilde{\mathsf{B}})$ obtains π such that $\mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1$ with probability $\mathsf{Succ}^{\mathsf{PSIPE}}_{\mathsf{EXT}} < \mathsf{Adv}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma} - \mathsf{negl}(\lambda)$. We argue that if this is the case, then it must be because (1) $\mathcal{A}_{\mathsf{PSIPE}}$ breaks the underlying blockchain protocol's common prefix properties, or (2) the adversary breaks IND-CPA security of TBIBE. This leads to two distinct cases:

1. Assuming TBIBE is secure, if a valid π does not appear on the common prefix of an honest party's blockchain, then $\mathcal{A}_{\mathsf{PSIPE}}$ is breaking the common prefix properties of the blockchain protocol Γ.
2. Assuming the blockchain protocol Γ is secure, if a valid π does not appear on the common prefix of an honest party's blockchain, then $\mathcal{A}_{\mathsf{PSIPE}}$ is breaking the security of TBIBE.

Consider case 1. Note that if $\mathcal{A}_{\mathsf{PSIPE}}$ distinguishes ciphertexts c_0, c_1 without allowing the extractor to obtain a valid SIP π and without breaking security of the TBIBE scheme, this implies that $\mathcal{A}_{\mathsf{PSIPE}}$ must have obtained valid decryption TBIBE decryption keys for a ciphertext $c_{b\in\{0,1\}}$; however, note that parties executing blockchain protocol Γ only produce these keys when a valid SIP π occurs in the common prefix of the blockchain (Fig. 5). Thus, if the same SIP π is not observed by the extractor later on, the common prefix property of the blockchain must have been violated.

Consider case 2. If $\mathcal{A}_{\mathsf{PSIPE}}$ distinguishes ciphertexts c_0, c_1 without violating the common-prefix property of the blockchain, then no valid SIP π is ever finalized in the common-prefix of the blockchain, which

would otherwise imply a successful SIP extraction. We can then construct an adversary $\mathcal{A}_{\mathsf{TBIBE}}$ from $\mathcal{A}_{\mathsf{PSIPE}}$ winning $\mathsf{Game}^{\mathsf{PUB\text{-}SIP}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}},\Gamma}$ without obtaining any TBIBE decryption keys for any ciphertext instances as follows.

1. $\mathcal{A}_{\mathsf{TBIBE}}$ in $\mathsf{Game}^{\mathsf{TBIBE}}_{\mathcal{A}_{\mathsf{TBIBE}}}$ acts as the challenger towards $\mathsf{Game}^{\mathsf{PSIPE}}_{\mathsf{PUB\text{-}SIP},\mathcal{A}_{\mathsf{PSIPE}}}$: it generates $(\mathsf{pk}_{\mathsf{PSIPE}}, \mathsf{sk}_{\mathsf{PSIPE}}) \leftarrow \mathsf{PSIPE.KGen}(\lambda)$ and receives $(\mathsf{pk}_{\mathsf{TBIBE}}, \{\mathsf{msk}_{\mathsf{TBIBE},i}\}_{i\in\mathcal{A}}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]})$ from the $\mathsf{Game}^{\mathsf{TBIBE}}_{\mathcal{A}_{\mathsf{TBIBE}}}$ challenger. It forwards $\mathsf{pk}_{\mathsf{PSIPE}} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{pk}_{\mathsf{SIG}})$ and $(\mathsf{pk}_{\mathsf{TBIBE}}, \{\mathsf{msk}_{\mathsf{TBIBE},i}\}_{i\in\mathcal{A}}, \{\mathsf{pk}_{\mathsf{TBIBE},i}\}_{i\in[n]})$ to the adversary $\mathcal{A}_{\mathsf{PSIPE}}$.
2. $\mathcal{A}_{\mathsf{TBIBE}}$ obtains (m_0, m_1) from $\mathcal{A}_{\mathsf{PSIPE}}$ and encrypts $\mathsf{c}_b \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, c_b)$ for $b \in \{0,1\}$. $\mathcal{A}_{\mathsf{TBIBE}}$ forwards $(m_0, m_1) = (\mathsf{c}_0, \mathsf{c}_1)$ to the challenger in $\mathsf{Game}^{\mathsf{TBIBE}}_{\mathsf{TBIBE},\mathcal{A}_{\mathsf{TBIBE}}}$.
3. $\mathcal{A}_{\mathsf{TBIBE}}$ forwards ct_b from the challenger in $\mathsf{Game}^{\mathsf{TBIBE}}_{\mathcal{A}_{\mathsf{TBIBE}}}$ to $\mathcal{A}_{\mathsf{PSIPE}}$.
4. $\mathcal{A}_{\mathsf{TBIBE}}$ outputs the bit which it obtains from $\mathcal{A}_{\mathsf{PSIPE}}$ in $\mathsf{Game}^{\mathsf{PSIPE}}_{\mathsf{PUB\text{-}SIP},\mathcal{A}_{\mathsf{PSIPE}}}$.

Notice that $\mathcal{A}_{\mathsf{TBIBE}}$ simulates $\mathsf{Game}^{\mathsf{PSIPE}}_{\mathsf{PUB\text{-}SIP},\mathcal{A}_{\mathsf{PSIPE}}}$ perfectly towards $\mathcal{A}_{\mathsf{PSIPE}}$. The ciphertext challenges from (simulated) and real challengers in $\mathsf{Game}^{\mathsf{PSIPE}}_{\mathsf{PUB\text{-}SIP},\mathcal{A}_{\mathsf{PSIPE}}}$ and $\mathsf{Game}^{\mathsf{TBIBE}}_{\mathcal{A}_{\mathsf{TBIBE}}}$ respectively are "consistent" in the underlying plaintext message; the ciphertext challenge from $\mathsf{Game}^{\mathsf{TBIBE}}_{\mathcal{A}_{\mathsf{TBIBE}}}$ simply includes another PKE encryption layer on the underlying plaintext that is consistent with the (m_0, m_1) chosen by $\mathcal{A}_{\mathsf{PSIPE}}$. Thus, any advantage of $\mathcal{A}_{\mathsf{PSIPE}}$ directly translates to the same advantage of $\mathcal{A}_{\mathsf{TBIBE}}$. □

C.2 Proof of Theorem 2 ($\mathbf{\Pi}_{\mathsf{TSIPE}}$ Security)

Proof. We prove Theorem 2 by showing a proof for Correctness, Unforgeability, IND-CPA Security and Self-Incriminating Proof Extractability properties of our scheme $\mathbf{\Pi}_{\mathsf{TSIPE}}$ as follows:

Correctness. The correctness of our $\mathbf{\Pi}_{\mathsf{TSIPE}}$ protocol is immediate and can be proven by the correctness of the underlying primitives' threshold encryption, MPC-hard function, commitment scheme and NIZK.

Fix a security parameter $\lambda \in \mathbb{N}$, value $\alpha, \beta, \zeta \in \mathbb{N}$ with $\alpha \geq 2\beta$ and $\beta * (\beta - \zeta) \geq 2\lambda$ (where ζ can be a function of β) and parameters (n, t) such that $0 < t < n$. The setup algorithm $\mathbf{\Pi}_{\mathsf{TSIPE}}.\mathsf{Setup}(1^\lambda, n, t)$ works as follows: a trusted third party who computes a public key and secret shares for a threshold encryption scheme $\left(\mathsf{pk}_{\mathsf{TE}}, \{\mathsf{sk}_i^{\mathsf{TE}}\}_{i\in[n]}\right) \leftarrow \mathsf{TE.Setup}\left(1^\lambda, n, t\right)$ and public parameters for a commitment scheme $\mathsf{ck} \leftarrow \mathsf{CS.Setup}(1^\lambda)$. For all $i \in [n]$, compute commitments to $\mathsf{sk}_i^{\mathsf{TE}}$ as: $\mathsf{cm}_{\mathsf{sk}_i} = \mathsf{CS.Com}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i})$ where $\rho_{\mathsf{sk}_i} \leftarrow \mathcal{H}_4(\mathsf{sk}_i^{\mathsf{TE}})$. The trusted third party distributes the public key $\mathsf{pk} = (\mathsf{pk}_{\mathsf{TE}}, \mathsf{ck}, \{\mathsf{cm}_{\mathsf{sk}_i}\}_{i\in[n]})$ to all parties, and a pair of secret key share and random $\mathsf{sk}_i = (\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ to each party P_i. In practice, this is substituted by a suitable distributed key generation protocol but we treat this as a trusted setup for the sake of simplicity.

For any message $m \in \{0,1\}^\lambda$, we encrypt the message by $\mathbf{\Pi}_{\mathsf{TSIPE}}.\mathsf{Enc}(\mathsf{pk}, m)$ by sampling $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^\beta$ as an input to an MPC-hard function $\mathsf{scratch}(s, z, \cdot)$, and searching for β number of nonces $\{w_1, \ldots, w_\beta\} \in \{0,1\}^{\alpha-\beta-2}$ (as described in step 2 in Figure 10) such that the first ζ bits of each $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ are zero. Given Lemma 1, this succeeds except with negligible probability. Finally, we compute the ciphertext $c = (c_1, c_2, c_3, z)$ as: $c_1 \leftarrow \mathsf{CS.Com}(\mathsf{ck}, (s\|z\|w\|q\|m); \rho_1)$ where $\rho_1 = \mathcal{H}_1(s\|w\|w\|q)$, $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$, $c_2 \leftarrow \mathsf{TE.Enc}\left(\mathsf{pk}_{\mathsf{TE}}, s; \rho_2\right)$ where $\rho_2 = \mathcal{H}_2(s\|z\|w\|q)$ and $c_3 = \mathcal{H}_2(s\|z\|w\|q) \oplus m$. Note that z is revealed in the ciphertext in order to allow for SIP extractability (described under TSIPE SIP extractability proof).

For the decryption of a ciphertext $c = (c_1, c_2, c_3, z)$ by $\mathbf{\Pi}_{\mathsf{TSIPE}}.\mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_i, c, \{\nu_i\}_{i\in[T]})$, a set of $t+1$ or more parties do the following: (i) first, decrypt c_2 as $s \leftarrow \mathsf{TE.Combine}\left(\mathsf{pk}_{\mathsf{TE}}, \{\nu_i\}_{i\in T}\right)$ where $\nu_i \leftarrow \mathsf{TE.ParDec}(\mathsf{sk}_i^{\mathsf{TE}}, c)$ and the correctness of it follows from the correctness of the threshold encryption scheme (as per Definition 25); (ii) then, search for β number of nonces $\{w_1, \ldots, w_\beta\} \in \{0,1\}^{\alpha-\beta-2}$ (as described in step 2 in Figure 10) by computing $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ such that the first ζ bits of each q_i are 0, and retrieve the original message as $m = c_3 \oplus \mathsf{PRF}(s\|z\|w\|q)$ where $\rho_1 = \mathcal{H}_1(s\|z\|w\|q)$, $\rho_2 = \mathcal{H}_1(s\|z\|w\|q)$, $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$, and the correctness of it follows from the correctness of the MPC-hard function $\mathsf{scratch}$ (in Figure 1) as per Lemma 1. Now, notice that for a party P_i who has access of the $\mathsf{sk}_i = (\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ and $(s, z, w, q, \rho_1, \rho_2, m)$ which is sufficient to generate a

self-incriminating proof π_i computed as a $\mathsf{NIZKPoK}$ with witness $(s, z, w, q, \rho_1, \rho_2, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ showing:

$$\Big\{\Big(\vee_{j\in[n]}\mathsf{CS.Com}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i}) = \mathsf{cm}_{\mathsf{sk}_j}\Big) \wedge$$
$$c_1 = \mathsf{CS.Com}(\mathsf{ck}, (s\|z\|w\|q\|m); \rho_1) \wedge$$
$$c_2 = \mathsf{TE.Enc}\left(\mathsf{pk}_{\mathsf{TE}}, s; \rho_2\right) \wedge$$
$$c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m\}$$

the correctness of self-incriminating proof π follows from the underlying $\mathsf{NIZKPoK}$ scheme (as per Definition 28) commitment scheme CS (as per Definition 19), threshold encryption scheme 25 and pseudorandom function. Therefore, $\mathbf{\Pi}_{\mathsf{TSIPE}}$ satisfies the correctness properties.

IND-CPA Security. IND-CPA security of TSIPE can be proven via the following sequence of hybrid arguments where start with the original $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ and finish at a hybrid where the ciphertext contains no information about the message:

H_0**:** The first hybrid is $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$'s view in the real-world game $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ for the TSIPE.

H_1**:** Recall the threshold encryption from our $\mathbf{\Pi}_{\mathsf{TSIPE}}$ construction proceeds by sampling two random as $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^{\beta}$ and then compute a threshold encryption as $c_2 = \mathsf{TE.Enc}\left(\mathsf{pk}_{\mathsf{TE}}, s; \rho_2\right)$ where $\rho_2 = \mathcal{H}_2(s\|z\|w\|q)$. The hybrid H_1 is the same as hybrid H_0 except that instead of generating a threshold encryption according to the above process, we just sample a random string in $\{0,1\}^{n\cdot(\alpha-\beta-2)+\beta}$ and use that to generate the encryption. We show that the two hybrids H_0 and H_1 are indistinguishable unless $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ breaks the IND-CPA security of the underlying threshold encryption scheme TE.

H_2**:** Recall the commitment process from our $\mathbf{\Pi}_{\mathsf{TSIPE}}$ construction proceeds by sampling two random as $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^{\beta}$. Next, it searches for β number of nonces $\{w_1, \ldots, w_\beta\} \in \{0,1\}^{\alpha-\beta-2}$ such that the first ζ bits of each $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ are 0. Then it computes a commitment as $c_1 = \mathsf{CS.Com}(\mathsf{ck}, (s\|z\|w\|q\|m); \rho_1)$ where $\rho_1 = \mathcal{H}_1(s\|z\|w\|q)$, $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$. The hybrid H_2 is the same as hybrid H_1 except that instead of generating a commitment according to the above process, we just sample a random string in $\{0,1\}^{(n+\beta)\cdot(\alpha-\beta-2)+\beta^2+\beta+\lambda}$ and use that to generate the commitment. We show that the two hybrids H_1 and H_2 are indistinguishable unless $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ breaks the hiding property of the underlying commitment scheme CS.

H_3**:** Recall the message encryption $c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m$ from our $\mathbf{\Pi}_{\mathsf{TSIPE}}$, construction proceeds by sampling two random as $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^{\beta}$, and next, it searches for β number of nonces $\{w_1, \ldots, w_\beta\} \in \{0,1\}^{\alpha-\beta-2}$ such that the first ζ bits of each $q_i \leftarrow \mathsf{scratch}(s, z, w_i) \in \{0,1\}^{\beta}$ are 0. Then it encrypt the message m as $c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m$ where $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$. The hybrid H_3 is the same as hybrid H_2 except that instead of generating an encryption according to the above process, we just sample a random string $r_3 \in \{0,1\}^{\lambda}$ and use that to compute the encryption as $c_3 = r_3 \oplus m$. We show that two hybrids H_2 and H_3 are indistinguishable until $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ breaks the computational indistinguishability in the pseudorandom function.

Assume by contradiction that there exists a (δ, Υ)-distributed adversary $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ when executing TSIPE from Figure 10. Such an adversary is able to distinguish between the hybrids in the sequence above. We will show that this $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ can be used to construct adversaries breaking the hiding property of the commitment scheme and the IND-CPA property of the threshold encryption scheme.

Hybrid H_1. In hybrid H_1, we use the threshold encryption to encrypt a randomly chosen string. Hence, the advantage of $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ in hybrid H_1 can be directly reduced to IND-CPA security of the underlying threshold encryption scheme. We construct an adversary $\mathcal{A}_{\mathsf{TE}}$ who is talking with the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$ and an internal copy of $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ for which it simulates $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$. The adversary $\mathcal{A}_{\mathsf{TE}}$ proceeds as follows:

1. $\mathcal{A}_{\mathsf{TE}}$ picks the threshold parameter (n, t) and choose a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| = |a| \leq t$.
2. $\mathcal{A}_{\mathsf{TE}}$ generates $(1^\lambda, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in\tilde{T}})$ by $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$ and forwards a pair $(\mathsf{pk}, \mathsf{sk}_j)$ to $\mathcal{A}_{j,\mathsf{TSIPE}}$ for all $j \in \tilde{T}$, acting as the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$.

3. $\mathcal{A}_{\mathsf{TE}}$ receives a tuple (m_0, m_1) from $\mathcal{A}_{1,\mathsf{TSIPE}}$. The $\mathcal{A}_{\mathsf{TE}}$ forwards the tuple $((s_0; \rho_{2,0}), (s_1; \rho_{2,1}))$ to the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$. The tuple $((s_0; \rho_{2,0}), (s_1; \rho_{2,1}))$ is computed as follows:
   - First it samples $s_0 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$, $s_1 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$, $z_0 \xleftarrow{\$} \{0,1\}^{\beta}$ and $z_1 \xleftarrow{\$} \{0,1\}^{\beta}$.
   - Then it searches for β number of nonces $\{w_{0,1}, \ldots, w_{0,\beta}\} \in \{0,1\}^{\alpha-\beta-2}$ and $\{w_{1,1}, \ldots, w_{1,\beta}\} \in \{0,1\}^{\alpha-\beta-2}$ such that the first ζ bits of:

$$q_{0,i} \leftarrow \mathsf{scratch}(s, z_0, w_{0,i}) \text{ are zero, for all } i \in [\beta]$$

$$q_{1,i} \leftarrow \mathsf{scratch}(s, z_1, w_{1,i}) \text{ are zero, for all } i \in [\beta]$$

     and set w_0, w_1 and q_0, q_1 as:

$$w_0 = (w_{0,1} \| \ldots \| w_{0,\beta}), \quad w_1 = (w_{1,1} \| \ldots \| w_{1,\beta})$$

$$q_0 = (q_{0,1} \| \ldots \| q_{0,\beta}), \quad q_1 = (q_{1,1} \| \ldots \| q_{1,\beta})$$

   - Finally it computes the randomness as $\rho_{2,0} = \mathcal{H}_2(s_0 \| z_0 \| w_0 \| q_0)$ and $\rho_{2,1} = \mathcal{H}_2(s_1 \| z_1 \| w_1 \| q_1)$.
4. $\mathcal{A}_{\mathsf{TE}}$ receives the challenge ciphertext c'_b from the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$. Then, $\mathcal{A}_{\mathsf{TE}}$ forwards $c_b = (c_1, c_2, c_3)$ to $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ where: $c_2 = c'_b$ and guess b^* to construct,

$$\begin{aligned} c_1 &\leftarrow \mathsf{CS.Com}(\mathsf{ck}, (s_{b^*} \| z_{b^*} \| w_{b^*} \| q_{b^*} \| m_{b^*}); \rho_{1,b^*}) \\ c_3 &\leftarrow \mathsf{PRF}(s_{b^*} \| z_{b^*} \| w_{b^*} \| q_{b^*}) \oplus m_{b^*} \\ z &\leftarrow z_{b^*} \end{aligned}$$

   where $\rho_{1,b^*} = \mathcal{H}_1(s_{b^*} \| z_{b^*} \| w_{b^*} \| q_{b^*})$.
5. Finally, $\mathcal{A}_{\mathsf{TE}}$ receives a guess b' from $\mathcal{A}_{1,\mathsf{TSIPE}}$. $\mathcal{A}_{\mathsf{TE}}$ forwards the guess b' to the $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$.

Notice that $\mathcal{A}_{\mathsf{TE}}$ simulates $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ exactly as in a real execution but guesses b^*. Since b^* is guessed at random, $\mathcal{A}_{\mathsf{TE}}$'s advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$ in negligibly close to half of the advantage of $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$. Hence, if $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$, then $\mathcal{A}_{\mathsf{TE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$.

Hybrid H_2. In hybrid H_2, we use the commitment scheme to commit to a randomly chosen string. Hence, the advantage of $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ in hybrid H_2 can be directly reduced to the hiding property of the underlying commitment scheme. We construct an adversary $\mathcal{A}_{\mathsf{CS}}$ who is talking with the challenger of $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$ and an internal copy of $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ for which it simulates $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$. The adversary $\mathcal{A}_{\mathsf{CS}}$ proceeds as follows:

1. $\mathcal{A}_{\mathsf{CS}}$ picks the threshold parameter (n, t) and choose a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| = |a| \leq t$.
2. $\mathcal{A}_{\mathsf{CS}}$ generates $(1^\lambda, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in\tilde{T}})$ by $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$ and forwards a pair $(\mathsf{pk}, \mathsf{sk}_j)$ to $\mathcal{A}_{j,\mathsf{TSIPE}}$ for all $j \in \tilde{T}$, acting as the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$.
3. $\mathcal{A}_{\mathsf{CS}}$ receives a tuple (m_0, m_1) from $\mathcal{A}_{1,\mathsf{TSIPE}}$. Then, $\mathcal{A}_{\mathsf{CS}}$ forwards the tuple $((s_0, z_0, w_0, q_0, m_0)_0, (s_1, z_1, w_1, q_1, m_1)_1)$ to the challenger of $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$. The tuple $((s_0, z_0, w_0, q_0, m_0)_0, (s_1, z_1, w_1, q_1, m_1)_1)$ is computed as follows:
   - First it samples $s_0 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$, $s_1 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)})$, $z_0 \xleftarrow{\$} \{0,1\}^{\beta}$ and $z_1 \xleftarrow{\$} \{0,1\}^{\beta}$.
   - Then it searches for β number of nonces $\{w_{0,1}, \ldots, w_{0,\beta}\} \in \{0,1\}^{\alpha-\beta-2}$ and $\{w_{1,1}, \ldots, w_{1,\beta}\} \in \{0,1\}^{\alpha-\beta-2}$ such that the first ζ bits of:

$$q_{0,i} \leftarrow \mathsf{scratch}(s, z_0, w_{0,i}) \text{ are zero, for all } i \in [\beta]$$

$$q_{1,i} \leftarrow \mathsf{scratch}(s, z_1, w_{1,i}) \text{ are zero, for all } i \in [\beta]$$

     and set w_0, w_1 and q_0, q_1 as:

$$w_0 = (w_{0,1} \| \ldots \| w_{0,\beta}), \quad w_1 = (w_{1,1} \| \ldots \| w_{1,\beta})$$

$$q_0 = (q_{0,1} \| \ldots \| q_{0,\beta}), \quad q_1 = (q_{1,1} \| \ldots \| q_{1,\beta})$$

4. $\mathcal{A}_{\mathsf{CS}}$ receives the challenge commitment cm_b from the challenger of $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$. Then, $\mathcal{A}_{\mathsf{CS}}$ and forwards $c_b = (c_1, c_2, c_3, z)$ to $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ where: $c_1 = \mathsf{cm}_b$; choose a random from $r_2 \in \{0,1\}^{n\cdot(\alpha-\beta-2)+\beta+\lambda}$ and construct $c_2 \leftarrow \mathsf{TE.Enc}(\mathsf{pk}_{\mathsf{TE}}, (r_2))$; and choose a random $r_3 \in \{0,1\}^\lambda$, construct $c_3 = r_3 \oplus m_b$, and guess b^* to construct $z = z_{b^*}$

5. Finally, $\mathcal{A}_{\mathsf{CS}}$ receives a guess b' from $\mathcal{A}_{1,\mathsf{TSIPE}}$. $\mathcal{A}_{\mathsf{CS}}$ forwards the guess b' to the $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$.

Now, notice that $\mathcal{A}_{\mathsf{CS}}$ simulates $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ exactly as in H_1 but guesses b^*. Since, c_2 and c_3 are computed using random value, $\mathcal{A}_{\mathsf{CS}}$'s advantage in $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$ is negligibly close to the advantage of $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$, and b^* is guessed at random, $\mathcal{A}_{\mathsf{CS}}$'s advantage in $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$ in negligibly close to half of the advantage of $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$. Hence, if $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$, then $\mathcal{A}_{\mathsf{CS}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$.

Hybrid H_3. In hybrid H_3, instead of generating an encryption $c_3 = \mathsf{PRF}(s\|z\|w\|q) \oplus m$, we use a randomly chosen string $r_3 \in \{0,1\}^\lambda$ to encrypt the message as $c_3 = r_3 \oplus m$. Now, we have computational indistinguishability of the pseudorandom function, since $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ cannot guess the random r_3 except with probability $\mathsf{poly}(\lambda)/2^\lambda$ since it can only make λ queries to $\mathcal{H}_2$ and there are 2^λ possible outputs. $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ can only guess (s,z,w,q). Hence, we simulate $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ exactly as in H_2 except with the negligible probability that $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ guesses (s,z,w,q).

We conclude the proof by observing that in the above hybrid argument, we reach a contradiction and thus our assumption of the existence of $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ against the IND-CPA of $\mathbf{\Pi}_{\mathsf{TSIPE}}$ cannot be true.

Unforgeability. If there exists an adversary $\mathcal{A}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$ or a (δ,Υ)-distributed adversary $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ for our scheme TSIPE, we show that these adversaries can be used to construct an adversary $\mathcal{A}_{\mathsf{NIZK}}$ breaking the soundness property of the $\mathsf{NIZKPoK}$ proof system used to generate the SIP π. We analyze each case separately.

Assume by contradiction that there exists a (δ,Υ)-distributed adversary $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ when executing TSIPE from Figure 10. Such a $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ is able to generate a valid SIP π, which is a $\mathsf{NIZKPoK}$ taking as witness $(s,z,w,q,\rho_1,\rho_2,m,\mathsf{sk}_i^{\mathsf{TE}},\rho_{\mathsf{sk}_i})$. Hence, this adversary is generating π for a ciphertext c_b it cannot decrypt in order to obtain the encryption randomness and message $(s,z,w,q,\rho_1,\rho_2,m)$. We construct an efficient adversary $\mathcal{A}_{\mathsf{NIZK}}$ with black-box access to $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ that has a non-negligible advantage in breaking soundness (in Definition 30) property of the NIZK scheme as per Definition 28 or the IND-CPA security of TSIPE.

We construct an adversary $\mathcal{A}_{\mathsf{NIZK}}$ who breaks the soundness property of the $\mathsf{NIZKPoK}$ proof system with non-negligible probability given an internal copy of $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$. $\mathcal{A}_{\mathsf{NIZK}}$ proceeds as follows:

1. $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ picks the threshold parameter (n,t) and chooses a subset $\tilde{T} \subset [n]$ of parties to corrupt, such that $|\tilde{T}| = |a| \leq t$.
2. $\mathcal{A}_{\mathsf{NIZK}}$ computes $(\mathsf{pk},\{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda,n,t)$ and forwards a $(\mathsf{sk}_j$ to $\mathcal{A}_{j,\mathsf{TSIPE}}$ for all $j \in \tilde{T}$, acting as the challenger of $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$.
3. When $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ queries $\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_i,i\in[n]\setminus\tilde{T}}$ with (i,c), $\mathcal{A}_{\mathsf{NIZK}}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup c$ (where $\mathcal{Q}$ initially empty) and answers the query with $\nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i,c)$ (which it can do since it has computed $\{\mathsf{sk}_i\}_{i\in[n]}$). When $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ queries $\mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}$ with m, $\mathcal{A}_{\mathsf{NIZK}}$ answers with $c \leftarrow \mathsf{Enc}(\mathsf{pk},m)$.
4. $\mathcal{A}_{\mathsf{NIZK}}$ receives a tuple (c',π') from $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$.
5. $\mathcal{A}_{\mathsf{NIZK}}$ returns π'.

It is clear that the (δ,Υ)-distributed adversary $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$'s view in the game $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ is indistinguishable from the view simulated by $\mathcal{A}_{\mathsf{NIZK}}$, since $\mathcal{A}_{\mathsf{NIZK}}$ executes $\mathsf{Setup}(1^\lambda,n,t)$ and simulates $\mathcal{O}^{\mathsf{ParDec}}_{\mathsf{sk}_i,i\in[n]\setminus\tilde{T}}$ and $\mathcal{O}^{\mathsf{Enc}}_{\mathsf{pk}}$ exactly as in the game. Since we assume that $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$, it is able to generate (c',π') such that $1 \leftarrow \mathsf{Vf}(\mathsf{pk},c',\pi')$ where $c' \notin \mathcal{Q}$. Hence, π is a valid $\mathsf{NIZKPoK}$ for the SIP statement of TSIPE that is generated without knowledge of the witness $(s,z,w,q,\rho_1,\rho_2,m,\mathsf{sk}_i^{\mathsf{TE}},\rho_{\mathsf{sk}_i})$, which violates the soundness and proof of knowledge properties of the $\mathsf{NIZKPoK}$ proof system. Thus, the existence of $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ that has non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ contradicts the security properties of $\mathsf{NIZKPoK}$.

Assume by contradiction that there exists an adversary $\mathcal{A}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$. This adversary is able to generate a valid π for a ciphertext for which it knows $(s, z, w, q, \rho_1, \rho_2, m)$ (*i.e.,* the encryption randomness, plaintext message and partial decryption for a set T) without knowing a pair $(\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$. Once again, we can use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{\mathsf{NIZK}}$ that breaks the soundness and proof of knowledge properties of the NIZKPoK proof system that is used to generate the SIP π with non-negligible probability given $\mathcal{A}$. $\mathcal{A}_{\mathsf{NIZK}}$ proceeds as follows:

1. $\mathcal{A}$ picks the threshold parameter (n, t).
2. $\mathcal{A}_{\mathsf{NIZK}}$ computes $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$.
3. $\mathcal{A}_{\mathsf{NIZK}}$ executes $(c', \pi') \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}}}(1^\lambda, \mathsf{pk})$.
4. When $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ queries $\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}_i, i\in[n]}$ with (T, j, c), $\mathcal{A}_{\mathsf{NIZK}}$ computes $\nu_i \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, c)$ for $i \in T$ and returns $(\pi, m) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{sk}_j, c, \{\nu_i\}_{i\in T})$ (which it can do since it has computed $\{\mathsf{sk}_i\}_{i\in[n]}$).
5. $\mathcal{A}_{\mathsf{NIZK}}$ returns π'.

It is clear that $\mathcal{A}$'s view in the game $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ is indistinguishable from the view simulated by $\mathcal{A}_{\mathsf{NIZK}}$, since $\mathcal{A}_{\mathsf{NIZK}}$ executes $\mathsf{Setup}(1^\lambda, n, t)$ and simulates $\mathcal{O}^{\mathsf{Dec}}_{\mathsf{sk}_i, i\in[n]}$ exactly as in the game. Since we assume that $\mathcal{A}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$, it is able to generate (c', π') such that $1 \leftarrow \mathsf{Vf}(\mathsf{pk}, c', \pi')$ where $c' \notin \mathcal{Q}$. Hence, π is a valid NIZKPoK for the SIP statement of TSIPE that is generated without knowledge of the witness $(s, z, w, q, \rho_1, \rho_2, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$, which violates the soundness and proof of knowledge properties of the NIZKPoK proof system. Thus, the existence of $\mathcal{A}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge1}}_{\mathsf{TSIPE},\mathcal{A}}$ contradicts the security properties of NIZKPoK.

Self-Incriminating Proof Extractability. Assume for the sake of contradiction that the protocol does *not* offer self-incriminating proof extractability. In that case, there exists a (δ, Υ)-distributed adversary $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ that has advantage $\mathsf{Adv}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ such that there does not exist an extractor kEXT_i for at least one $i \in [a]$ that can output a value π_i s.t. $\mathsf{Vf}(\mathsf{pk}, c, \pi_i) = 1$ with $\mathsf{Succ}^{\mathsf{TSIPE}}_{\mathsf{kEXT}} \geq \mathsf{Adv}^{\mathsf{SIP-SEC}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a} - \mathsf{negl}(\lambda)$. If this is the case, then it must be because no valid π_i can be produced with non-negligible probability from $(\mathsf{pk}, c, \mathsf{sk}_i, \tau_i^{\mathsf{fast}})$ even when $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ is able to distinguish the challenge ciphertext c_b. Observe that if this is the case, then it implies that no adversary $\mathcal{A}_{i,\mathsf{TSIPE}}$ for $i \in [a]$ has executed scratch correctly. However, this is not possible as it contradicts the IND-CPA security of TSIPE (as described above in TSIPE IND-CPA security proof). To see this, first, observe that the adversary needs to be able to learn a non-negligible amount of information about the message m_b encrypted by the challenger. However, the message m_b can only be derived by computing $\mathsf{PRF}(s\|z\|w\|q)$ where $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$, for values $s\|z\|w\|q$ that have high entropy and thus cannot be brute forced by a polynomial time adversary. More specifically, this input to the random oracle PRF has at least $\beta * (\alpha - \beta - 2) + \beta * (\beta - \zeta) = \beta * (\alpha - 2 - \zeta)$ bit of entropy due to the computation $q_j \leftarrow \mathsf{scratch}(s, z, w_j)$ for all $j \in [\beta]$ containing $\beta * (\alpha - 2 - \zeta)$ bits of entropy (as each w_j is $(\alpha - \beta - 2)$ bits and each q_j is β bits but it has first ζ bits are 0). Now, since $\alpha \geq 2\beta$ (as defined in Figure 1) and $\beta * (\beta - \zeta) \geq 2\lambda$, we get that,

$$\begin{aligned}
\beta * (\alpha - 2 - \zeta) &\geq \beta * (2\beta - 2 - \zeta) \\
&= 2\beta^2 - 2\beta - \beta\zeta \\
&= \beta^2 - \beta\zeta + \beta^2 - 2\beta \\
&= \beta * (\beta - \zeta) + \beta^2 - 2\beta \\
&= 2\lambda + \beta^2 - 2\beta \\
&\approx 2\lambda
\end{aligned}$$

for the security parameter λ. Hence, no polynomial-time adversary can brute-force these. Furthermore, observe that these are all derived using random oracles, and hence no, non-brute-force attack is possible. Thus, each q_j must be computed using scratch taking as input the values s, z and w_j, for all $j \in [\beta]$. This means that for some adversary $\mathcal{A}_{i,\mathsf{TSIPE}}$ to learn m_b it *must* have executed scratch correctly in which case Lemma 1 guarantees that there exists an extractor kEXT_i that obtains the s from τ_i^{fast} for at least one sub-adversary $\mathcal{A}_{i,\mathsf{TSIPE}}$.

Now, notice that the extractor kEXT_i has obtained s from τ_i^{fast}, and each extractor kEXT_i is also given $c = (c_1, c_2, c_3)$ and the public key pk; so the Lemma 1 guarantees that the extractor kEXT_i can

correctly compute β number of nonces $\{w_1, \ldots, w_\beta\}$ in polynomial-time by running $q_j \leftarrow \mathsf{scratch}(s, z, w_j)$ such that the first ζ bits of q_j are zero (as described in Figure 1), for all $j \in [\beta]$. Then the extractor kEXT_i can compute $\rho_1 = \mathcal{H}_1(s\|z\|w\|q)$ and $\rho_2 = \mathcal{H}_2(s\|z\|w\|q)$ using (s, z, w, q) where $w = (w_1\| \ldots \|w_\beta)$. Finally, the extractor kEXT_i can compute $m = c_3 \oplus \mathsf{PRF}(s\|z\|w\|q)$ where $q = (q_1\| \ldots \|q_\beta)$. Next, see that the last values needed construct a valid π_i using $\mathsf{NIZKPoK}$ is the public key pk and the secret share sk_i of the sub-adversary $\mathcal{A}_{i,\mathsf{TSIPE}}$. Since kEXT_i is also given $\mathsf{sk}_i = (\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$, hence it can compute a valid proof π_i. Thus we can conclude that $\mathbf{\Pi}_{\mathsf{TSIPE}}$ has Self-Incriminating Proof Extractability when $\mathsf{scratch}$ is MPC-hard as per defined in Figure 1.