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 arbitrary 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 Multi-party 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 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 investigate the (im)possibility and applications of our notions while providing matching constructions 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 [25] 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 secret key holder may do so at any time without being detected. Compared with the other main public key primitive of digital signatures, observe that this problem is unique for decryption. 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.

Instead of giving the full power to decrypt a ciphertext to a single party who controls a secret key, the notion of *threshold* public key encryption [23,24] distributes decryption power among a set of parties (*i.e.,* a decryption committee). In this setting, no single party can decide to decrypt a ciphertext and a set of more than t out of n total parties must cooperate to successfully decrypt. Compared to the standard public key encryption notion, threshold encryption allows the decryption committee to enforce rules about what ciphertexts should be decrypted. However, this notion requires that no more than t parties act maliciously. Otherwise, the same issue from the standard public key setting occurs as a sufficiently

⋆ 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.

large subset 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 act 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" decryptions of arbitrary ciphertexts without ever being detected. Even if we construct schemes that readily allow for detecting rogue decryptions through publicly available information, powerful cryptographic tools such as anonymous channels [15] and Multiparty Computation (MPC) [16, 32] can be employed by malicious parties to avoid detection. In this context, we ask the following questions:

Is it possible to detect rogue decryptions in (threshold) public key encryption schemes? If yes, is that still possible even when the secret key owner(s) employ cryptographic tools to avoid detection?

We answer both questions in the affirmative by proposing definitions and matching constructions of both standard and 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. Our notions guarantee security even against adversaries who may employ cryptographic tools such as MPC to avoid producing a SIP, thus thwarting advanced "cryptovirological" attacks.

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 applications 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 notions 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 committees that provide threshold decryption as a service and are implicitly trusted by their clients, without any means for detecting malfeasance.

One concrete example of where accountability in decryption can bring utility is in the case of "encrypted mempools". In the blockchain space, a mempool is the set of transactions that are known, but have not yet been persisted to the ledger. More specifically they are known to the miners/validators. Thus allowing the miners/validators to "front-run" these during persistence, to gain financial profit via so-called "miner extractable value" [19]. A common way of preventing this is to encrypt the mempools until they are persisted [45] 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.

In other cases, the decrypting party (or parties) might not be incentivized to misbehave, but external parties might be incentivized to bribe the decrypter to incorrectly 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. This happens in the threshold setting when using outsourced secure multi-party computation [39] with a scheme based on homomorphic encryption [6, 21]. For example in the case of loan benchmarking; where clients are banks who input customers' confidential financial information [20], which if revealed to competing banks, could give them a competitive advantage. In the blockchain space, we see this in systems realizing privacy-preserving smart contracts, such as Oasis [44] (based on TEEs) or Zama's fhEVM [18] (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. Gambling contracts that require randomness computed from a seed are particularly vulnerable to malicious decryptions, as knowledge of the seed could yield knowledge of all future values of "random" variables.

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 "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 the following

primitives: Encryption with Self-Incriminating Proof (SIPE), Encryption with 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 produced (SIPE) or published (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 provide concrete constructions of our proposed primitives and investigate the limits of these notions. We summarize our main contributions as follows:

- Definitions for the notions of Encryption with Self-Incriminating Proof (SIPE), which guarantees that a SIP is produced, and of Encryption with Public Self-Incriminating Proof (PSIPE), which guarantee that a SIP is published, in the context of standard encryption.
- A construction of Encryption with Public Self-Incriminating Proof (PSIPE) building on a public ledger to ensure publication of SIPs and to instantiate the witness encryption building block.
- Definitions and a matching construction for the notion of Threshold Encryption with Self-Incriminating Proof (SIPE), building on Individual Cryptography by Dziembowski *et al.* [26].

In our SIPE notion, we model the guarantee that a party who decrypts a ciphertext must produce a SIP. However, as we discuss later, this guarantee's usefulness is questionable, since an adversary can always erase the SIP that it has produced internally. Moreover, the SIPE notion seems intrinsically intertwined with the notion of extractable witness encryption, leaving little hope for realizing it under standard assumptions without setup.

The issues with the SIPE notion motivate us to investigate a stronger notion of standard public key encryption with SIPs where we force the decrypting party to publish the SIP as part of the decryption process. We call this notion PSIPE and define it for an underlying public ledger, which is assumed as a setup. Notice that assuming this setup is necessary for capturing the guarantee that a SIP is published, *i.e.,* all honest parties learn the SIP. Although this notion requires setup, we show a construction based on standard assumptions. This construction could trivially be generalized to the threshold setting, by each party doing a partial decryption, and validating a quorum of proofs of partial decryption. Still, such a construction would be expensive. Furthermore, if we can assume at least one honest party then there is no need to force *each* party to post a proof to a public ledger (since an honest party would always publish the necessary information according to the protocol). Thus we introduce a new protocol for the threshold notion TSIPE, which does not require a public ledger, nor witness encryption. This protocol ensures that at least one of the parties who cooperate in decrypting a given ciphertext will learn a SIP showing that this ciphertext has been decrypted. This guarantee holds even if the parties performing rogue decryption employ MPC to try and learn the plaintext, while hiding the SIP. The motivation for this notion is that this SIP can be used as a deterrent to keep parties from joining rogue decryption operations, lest the SIP be leaked and their collective malfeasance revealed. We show a construction build from a regular threshold encryption scheme, a commitment scheme, and a non-interactive proof of knowledge, which we prove secure in a similar model as the recent work on Individual Cryptography [26].

1.2 Technical Overview

We present a construction of a PSIPE scheme based on a public ledger realized by a Proof-of-Stake blockchain, an extractable witness encryption (eWE) scheme for a specific language, and a signature scheme. We later show that we can realize this eWE efficiently from standard assumptions using the underlying blockchain via techniques from [37] suitably combined with a publicly verifiable secret sharing scheme to enforce public proofs of malfeasance as proposed in [12]. 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 cannot be feasibly computed by an MPC protocol in the model from [26].

In our construction $\mathbf{\Pi}_{\mathsf{PSIPE}}$ realizing the notion of PSIPE, the core idea is to force the decryptor to generate and publish on the blockchain a signature σ on a reference message d that is valid under a given public key pk to decrypt a ciphertext c. To do so, we encrypt a message m using an eWE scheme under a statement $\mathsf{pk}, d, \mathtt{B}$ for a random d, the prescribed pk and a current state of the blockchain $\mathtt{B}$, obtaining a ciphertext c' and defining our PSIPE as $c = (c', d)$. Now c' can only be decrypted by a party who has a witness $(\sigma, \tilde{\mathtt{B}})$ where $\tilde{\mathtt{B}}$ is a future valid state of the blockchain that evolved from $\mathtt{B}$ containing σ such that σ verifies as a valid signature on d under pk. Hence, the decryptor is forced to publish σ to obtain a $\tilde{\mathtt{B}}$ that allows it to decrypt c. 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 [36]. We later also use the blockchain to realize the eWE scheme based on standard assumptions using the "eWE on Blockchains" construction from [37], using techniques from [12] to ensure that it cannot be abused to decrypt a ciphertext without generating a SIP.

In the threshold setting, we start from the distributed adversarial model and the concept of MPC-hard function, both introduced in the recent work on Individual Cryptography [26]. In the distributed adversary model, each corrupted party is a sub-adversary that acts individually, without being controlled by a monolithic adversary. However, sub-adversaries can execute interactive protocols, including MPC-based ones, to mount their attacks. Simply, a function is said to be MPC-hard if it is infeasible to compute in MPC, but easy to compute in plain. This helps in reasoning that at least one of the sub-adversaries must learn the entire input to the MPC-hard function to feasibly evaluate it. As shown in [26], MPC-hard functions can be constructed from hash functions similarly to Proof-of-Work puzzles used in blockchain-based consensus protocols, with careful analysis to avoid attacks that exploit the structure of the underlying hash function and of the puzzle.

When constructing our protocol $\mathbf{\Pi}_{\mathsf{TSIPE}}$, which realizes the notion for TSIPE, we start with a regular threshold encryption scheme and embed 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 to obtain the message while discarding the SIP, which they cannot do as at least one sub-adversary must learn the input to evaluate the MPC-hard function. To encrypt a message m we first sample an input in to an MPC-hard function and compute it to obtain an output out, which is used to derive a one-time pad key $\mathcal{H}(\mathsf{out})$ using a random oracle $\mathcal{H}$. Our ciphertext is then composed by a commitment c_1 to $(\mathsf{in}, \mathsf{out}, m)$, an encryption c_2 of in with the underlying threshold encryption scheme and $c_3 = \mathcal{H}(\mathsf{out}) \oplus m$. To decrypt c_1, c_2, c_3, the sub-adversaries must first compute out, which requires at least one of them to learn in. Now we can generate a SIP as a zero-knowledge proof showing knowledge of an opening to commitment c_1 and of a secret key share for the underlying threshold encryption scheme, without revealing which secret key share is used.

1.3 Related Work

The concept of identifying malicious decryptions carried out by a key-holder was initially considered by Ryan [46] and dubbed *accountable encryption*. Later, it was more formally defined by Li *et al.* [42], who showed security definitions and protocols for the identification of malicious decryptions when the decryption key is stored and used through a single TEE.

Using cryptography for malicious purposes was first considered by Young and Yung [48] in 1996. They studied the idea of "Cryptovirology", which consists of using cryptographic tools maliciously. Specifically, the work of [48] focuses on the malicious use of public-key encryption. For example, they show how cryptographic tools can be used to mount extortion-based attacks that cause loss of access to information, loss of confidentiality, and information leakage; tasks which cryptography typically prevents. In our case, we initiate the study of *Distributed Cryptovirology*, specifically, 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* [17,35,41]. Chor *et al.* [17] 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, Boneh *et al.* [10] initiated the study of traitor-tracing in the context of threshold decryption and showed several constructions for it. However, as mentioned in [26], none of above approaches [10,17,41,48] considers a distributed adversarial model [26]. 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 [1, 3] and collusion-preserving [2] 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 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 in plain.

Recently, Dziembowski *et al.* [26] considered a distributed adversarial model and studied the malicious use of MPC within it. More specifically, they introduced and studied the notion of *individual cryptography*. They say an algorithm is individual if, in every implementation of that algorithm, there always exists a single party with full knowledge of the input to that algorithm. 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.* [40] introduced the concept of *proof of complete knowledge* (PoCK) which is very similar to the notion of PoCK in [26]. A PoCK guarantees that a single party has complete knowledge of its secret. In particular, Kelkar *et al.* [40] showed a construction of PoCK that directly achieves a zero-knowledge property.

1.4 Organization

The rest of the paper is organized as follows: Sec. 2 discusses our technical preliminaries. Sec. 3 introduces the syntax and security definition of our SIPE notion. Sec. 4 presents the syntax, security definition, and construction of our PSIPE notion. Sec. 5 presents the syntax, security definition, and construction of our TSIPE notion. Sec. 6 concludes our work with some future research directions.

2 Preliminaries

In this section, we introduce the notation, building blocks, and models used in this paper. We refer the reader to Appendix A for more details.

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 highest threshold of parties that cannot compromise security. Hence $0 < t < n$. 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. We will also use the notion of an extractor algorithm, EXT, in connection with game-based security. EXT is a PPT algorithm that is queried on a transcript and must produce an output of a specific format with a certain success probability. We typically use this on the transcript of one or more adversarial algorithm executions, including their input, output, and randomness.

2.1 Building Blocks

Digital Signatures. A digital signature scheme is a tuple of PPT algorithms $\mathsf{SIG} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$. The key generation algorithm $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{KG}(1^\lambda)$ outputs a key-pair $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}})$. The signing algorithm $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{SIG}}, m)$, outputs a signature σ on message $m \in \{0,1\}^*$ under $\mathsf{sk}_{\mathsf{SIG}}$. The verification algorithm $1/0 \leftarrow \mathsf{Vf}(\mathsf{pk}_{\mathsf{SIG}}, m, \sigma)$, outputs 1 if and only if σ is a valid signature on m generated under $\mathsf{sk}_{\mathsf{SIG}}$. A SIG scheme must satisfy the notions of *correctness* and *existential unforgeability against adaptive chosen message attacks* (EUF-CMA) [34]), formalized in Appendix A.1.

Commitment Schemes. A commitment scheme CS consists of the tuple of PPT algorithms $(\mathsf{Setup}, \mathsf{Commit})$. The setup algorithm $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ outputs a commitment key ck, defining a message space $\mathcal{M}$ and a randomness space $\mathcal{R}$. The commitment algorithm $\mathsf{cm} \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$ takes as inputs ck, a message $\mathsf{s} \in \mathcal{M}$ and randomness $\rho \in \mathcal{R}$, outputting a commitment cm. A commitment scheme CS must satisfy the standard properties of computational *binding* and *hiding*, formalized in Appendix A.2.

Extractable Witness Encryption. We recall the concept of witness encryption from [30]. 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 PPT algorithms $(\mathsf{Enc}, \mathsf{Dec})$. The encryption $c \leftarrow \mathsf{Enc}(1^\lambda, \mathcal{L}_{\mathsf{eWE}}, \mathsf{inst}, m)$ takes as input the language $\mathcal{L}_{\mathsf{eWE}}$, a statement inst and a message $m \in \mathcal{M}$, outputting a ciphertext c. The decryption algorithm $m/\perp \leftarrow \mathsf{Dec}(c, \mathsf{wit})$ takes as input a ciphertext c and a (witness) wit, outputting m if and only if $(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{eWE}}$. We require an extractable witness encryption scheme eWE to satisfy the properties of *correctness* and *extractable security* (*i.e.,* it is possible to extract wit from an adversary who distinguishes ciphertexts), formalized in Appendix A.3.

Threshold Encryption. A threshold encryption scheme TE consists of a tuple of PPT algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine})$. The setup algorithm $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$, it takes as input the security parameter 1^λ, the number of shares n and the threshold t, outputting a public key pk and a threshold decryption key share sk_i for each party P_i. The encryption algorithm $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, takes as input pk and a message m, outputting a ciphertext c. The partial decryption algorithm $\mu_i \leftarrow \mathsf{ParDec}(\mathsf{pk}, \mathsf{sk}_i, c)$, it takes as input pk, sk_i and c, producing a partial decryption μ_i. The combine algorithm $m/\perp \leftarrow \mathsf{Combine}(\mathsf{pk}, c, \{\mu_i\}_{i\in T})$, takes as input pk, c and $\{\mu_i\}_{i\in T}$ where $|T| \geq t+1$, and outputs the plaintext message m. We require a threshold encryption scheme TE to satisfy the standard properties of *correctness* and *IND-CPA security*, formalized in Appendix A.4.

NIZKs. A non-interactive zero-knowledge (NIZK) [9] proof system for an **NP**-language $\mathcal{L}_{\mathsf{NIZK}}$ with witness relation $\mathcal{R}_{\mathsf{NIZK}}$ is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$. Algorithm $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$ outputs a common random string crs, implicitly used by the other algorithms. The prover algorithm $\pi \leftarrow \mathsf{P}(\mathsf{crs}, \mathsf{inst}, \mathsf{wit})$, the prover algorithm takes as input a statement $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ and a witness wit, outputting a proof π. The verifier algorithm $1/0 \leftarrow \mathsf{V}(\mathsf{crs}, \mathsf{inst}, \pi)$ takes as input a statement $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ and a proof π, outputting 1 if and only if it accepts the proof π. We require a NIZK proof system $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ to satisfy the properties of *completeness*, *soundness*, and *zero-knowledge*. Relations are written as $\mathcal{R}_{\mathsf{NIZK}} = \{(\mathsf{inst}, \mathsf{wit}) \mid R(\mathsf{inst}, \mathsf{wit}) = 1\}$ where $\mathsf{inst} \in \mathcal{L}_{\mathsf{NIZK}}$ is the statement, wit is the witness and R is some predicate. Our construction uses non-interactive zero-knowledge proof for knowledge (NIZKPoK), thus requiring the proof constructor to know the witness wit. 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 $\mathcal{R}_{\mathsf{NIZK}}$. The syntax and security definition are formalized in Appendix A.5.

2.2 Proof-of-Stake (PoS) Blockchains

In this section, we recall in almost verbatim form the overview given in [13] of the model for PoS blockchain protocol execution of [36]. In PoS-based blockchains, each participant is associated with some stake in the system, which could be measured as a positive rational value. These protocols rely on a lottery mechanism that ensures (of all eligible parties) 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 start with the framework of Goyal *et al.* [36] which, in turn, builds on the analysis done in [28, 43]. We refer the reader to revisit the abstraction used in [36]. Below we present a summary of the framework and the main properties we will use in this paper. Moreover, we note that in [36] it is proven that there exist PoS blockchain protocols (e.g., Snow White [8], Ouroboros Praos [22]) with the properties described below. We refer the reader to Appendix A.6 for a detailed description of blockchain protocol execution and its properties.

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 signature scheme public 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 $B_i = (\mathcal{H}(B_{i-1}), \mathtt{d}, \mathtt{aux}))$ where $\mathcal{H}(B_{i-1})$ is a collision-resistant hash of the previous block, $\mathtt{d}$ is data and $\mathtt{aux}$ 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: $m = (P_i, P_j, q, \mathtt{aux})$ where P_i transfers q coins to P_j along with some optional, auxiliary information $\mathtt{aux}$.

Blockchain Protocol Execution. 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. The update state algorithm $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$, takes as input the security parameter 1^λ and outputs st which is the local state of the blockchain along with metadata. The get records algorithm $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$, 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). The broadcast algorithm $\mathsf{Broadcast}(1^\lambda, m)$, 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.

More UC formally, we model the blockchain as a uniquely defined execution based on the inputs, random coins, and messages received from all involved parties in line with the work of Goyal *et al.* [36]. More specifically, we define the execution as a random variable $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ when executing the blockchain Γ^V with security parameter 1^λ with the UC environment $\mathcal{Z}$ and adversary $\mathcal{A}$. The view of each participating party P_i is then denoted as $\mathsf{VIEW}_{P_i}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$, and the view of the adversary $\mathcal{A}$ as $\mathsf{VIEW}_{\mathcal{A}}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$. We simplify this expression with VIEW_i, respectively $\mathsf{VIEW}_{\mathcal{A}}$, when it does not lead to ambiguity.

Defining u-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,

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

Evolving Blockchains. To define Encryption with Public Self-Incriminating Proofs scheme (in Definition 8), 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 1. *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 $\tilde{\mathtt{B}}$, $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) \in \{0,1\}$ 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 2 below.

Definition 2 (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 1). Let $\mathsf{SIG} =$*

$(\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ *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 2.

2.3 The Distributed Adversarial Model and MPC-hard Functions

In this section, we describe the distributed adversarial model and an MPC-hard function that we consider for our proposed primitive TSIPE definition and construction. This description is taken almost verbatim from [26].

The *distributed adversary* [26] 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.* [26] define the notion of MPC-hard cryptographic tasks. 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) some secret information $\mathtt{s}$ completely.

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-adversary $\mathcal{A}_j$ 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 to MPC protocols [6, 21].

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.

Security Games for Distributed Adversary. To formalize "knowledge" in the distributed adversary attack scenario, Dziembowski *et al.* [26] used the concept of a "knowledge extractor" proposed in [5]. Bellare and Rogaway [5] 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.* [26] 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 $\mathsf{s} \in \mathsf{kEXT}_p(\tau_j^{\mathsf{fast}})$.

Instantiating an MPC-hard Function. Dziembowski *et al.* [26] 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 of 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 [26] 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 [26] 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 11), we use $\mathsf{scratch}$ function and the goal is that searches for 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 Sec. 4 in [26] for a more detailed description, correctness, and security (MPC-hardness) of the $\mathsf{scratch}$ procedure.

MPC-hard Function: $\mathsf{scratch}$

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. Compute $q = \mathcal{H}(10 \| s_n \| q_n^d)$.
4. Output q

Fig. 1: Construction of a MPC-hard Function from [26].

We use the MPC-hard function scratch in our constructions, our goal is to later extract the inputs to 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 [26].

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 scratch procedure requires $(nd+1)$ hashes $\mathcal{H}$. For a party, observe that each 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, $\lfloor (nd+1)\cdot 2^{\zeta}\cdot\beta/(nd+1)\rfloor = 2^{\zeta}\cdot\beta$ 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 [26] for the formal statement) less than β times exponentially small. Note that this budget allows the η-bounded party P to evaluate scratch $\lfloor \eta/(nd+1)\rfloor = \lfloor (nd+1)\cdot 2^{\zeta+1}\cdot\beta/(nd+1)\rfloor = 2^{\zeta+1}\cdot\beta$ times.

Lemma 1 (Derived from [26]). *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 [26], 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 $\mathcal{O}_{\mathcal{H}}$.

3 Encryption with Self-Incriminating Proofs

In this section, we introduce the notion of public key Encryption with Self-Incriminating Proof (SIPE). 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. 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 SIPE 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 [37].

3.1 Formal Syntax and Security Definitions

Here, we describe the syntax of SIPE, followed by formal security definitions.

Definition 3 (Encryption with Self-Incriminating Proofs). *An encryption with self-incriminating proof scheme* SIPE *consists of the following* PPT *algorithms* (KG, Enc, Dec, Vf)*, which have the following syntax:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$, *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.*

An encryption with self-incriminating proof scheme SIPE *must satisfy the following properties:* ***Correctness*** *(Definition 4),* ***Unforgeability*** *(Definition 5),* ***IND-CPA Security*** *(Definition 6) and* ***Self-Incriminating Proof Extractability*** *(Definition 7).*

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 4 (Correctness). *A scheme* $\mathsf{SIPE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf})$ *is correct if for any security parameter* λ *and any message* $m \in \{0,1\}^\lambda$, *the following holds:*

- ***Message decryption correctness:***

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

- ***Self-incriminating proof correctness:***

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

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 [34]. 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 5 via a game $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIPE},\mathcal{A}}$, which is presented in Figure 2.

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

Game steps:

1. Initialize an empty list $\mathcal{Q} \leftarrow \emptyset$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$
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:

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

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

Definition 5 (Unforgeability). *A scheme* $\mathsf{SIPE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf})$ *is unforgeable if for any security parameter* λ *and for all* PPT *adversaries* $\mathcal{A}$ *given unlimited oracle access to* $\mathcal{O}_{\mathsf{sk}}^{\mathsf{Dec}}(\cdot)$*, advantage* $\mathsf{Adv}^{\mathsf{Unforge}}_{\mathsf{SIPE},\mathcal{A}}$ *of the* $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{SIPE},\mathcal{A}}$ *(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 6 via a game $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SIPE},\mathcal{A}}$, which is presented in Figure 3.

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

Game steps:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$
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'

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:

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

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

Definition 6 (IND-CPA Security). *A scheme* $\mathsf{SIPE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf})$ *is IND-CPA secure if for any security parameter* λ *and for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SIPE},\mathcal{A}}$ *of* $\mathcal{A}$ *in* $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SIPE},\mathcal{A}}$ *(in Figure 3) is negligible.*

Self-Incriminating Proof Extractability. This property guarantees that decryption always produces a self-incriminating proof, *i.e.,* there exists an extractor that can obtain this proof by interacting with an adversary who succeeds in distinguishing between challenge ciphertexts. We formalize this notion in Definition 7 via a game $\mathsf{Game}^{\mathsf{SIP\text{-}Ext}}_{\mathsf{SIPE},\mathcal{A}}$ for self-incriminating proof extractability with a standard monolithic adversary, shown in Figure 4.

$\mathsf{Game}^{\mathsf{SIP\text{-}Ext}}_{\mathsf{SIPE},\mathcal{A}}$

Game steps:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$; `// Guarantee that the keys are generated correctly.`
2. $(\mathsf{st}, m_0, m_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk})$
3. Samples $b \xleftarrow{\$} \{0,1\}$
4. $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
5. $b' \leftarrow \mathcal{A}_2(\mathsf{st}, c_b)$

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:

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

The extractor EXT's advantage in this game is:

$$\mathsf{Succ}^{\mathsf{SIPE}}_{\mathsf{EXT}} = \left| \mathsf{Pr}\left[\pi \leftarrow \mathsf{EXT}(\mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk}), \mathcal{A}_2(\mathsf{st}, c_b)) \mid \mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1\right] \right|$$

Fig. 4: Self-Incriminating Proof Extractability Game for SIPE scheme.

Definition 7 (Self-Incriminating Proof Extractability). *A scheme* $\mathsf{SIPE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf})$ *has self-incriminating proof extractability if for any security parameter* λ *and for all polynomials* $p(\cdot)$ *and* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{sk}), \mathcal{A}_2(\mathsf{st}, c_b))$ *with* $\mathsf{Adv}^{\mathsf{SIP\text{-}Ext}}_{\mathsf{SIPE},\mathcal{A}}$ *in* $\mathsf{Game}^{\mathsf{SIP\text{-}Ext}}_{\mathsf{SIPE},\mathcal{A}}$ *(Figure 4), there exist a polynomial* $q(\cdot)$ *and* PPT *extractor* EXT *that outputs* $\pi \leftarrow \mathsf{EXT}(\mathcal{A}_1, \mathcal{A}_2)$ *such that* $\mathsf{Vf}(\mathsf{pk}, c_b, \pi) = 1$ *with success probability* $\mathsf{Succ}^{\mathsf{SIPE}}_{\mathsf{EXT}}$ *such that the following holds:*

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

3.2 Theoretical and Practical Issues with SIPE Notion

We introduce SIPE in the monolithic adversary setting (in Definition 3). This notion captures 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. However, we observe that a simple SIPE notion in the monolithic adversary setting seems very hard to realize based on standard assumptions. Namely, formalizing that a SIP must be produced by an adversary who knows the secret key, requires us to define a notion of extractability akin to extractable witness encryption. Besides the seeming relationship of this notion with very strong primitives, even if we could do it, its usefulness would be unclear. More specifically, the adversary may just choose to erase the SIP that it is forced to produce during decryption. Hence, the original SIPE notion is not very useful for any real-world applications. This is our main motivation for investigating the stronger notion of Encryption with Public Self-Incriminating Proof, which we introduce in Sec. 4. While this stronger notion requires a public ledger as a setup, it solves the significant practical and theoretical shortcomings of our original SIPE notion.

4 Encryption with Public Self-Incriminating Proof

Since the notion of Self-Incriminating Proof Extractability has limited applicability and is strongly related to extractable witness encryption, 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 8, followed by formal security properties. Then we provide a construction of our PSIPE in Figure 6.

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 [37] 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 $\tilde{\mathtt{B}}$ of the ledger has evolved from an initial state $\mathtt{B}$, which is crucial for our definitions and constructions. This verification is captured by the evolving blockchain predicate (defined in Sec. 2.2), *i.e.,* $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1$ iff $\tilde{\mathtt{B}}$ is obtained as a future state of executing the blockchain protocol starting from $\mathtt{B}$.

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 8 (Encryption with Public Self-Incriminating Proof). *An encryption with public self-incriminating proof scheme* PSIPE *consists of the following* PPT *algorithms* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$ *in the context of a blockchain* Γ^V *with evolved predicate* evolved *(as in Definition 1), which have the following syntax:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$*: same as* $\mathsf{SIPE.KG}(1^\lambda)$.
2. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: same as* $\mathsf{SIPE.Enc}(\mathsf{pk}, m)$.
3. $(m, \pi)/\perp \leftarrow \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$*: same as* $\mathsf{SIPE.Dec}(\mathsf{pk}, \mathsf{sk}, c)$.
4. $1/0 \leftarrow \mathsf{Vf}(\mathsf{pk}, c, \pi)$*: same as* $\mathsf{SIPE.Vf}(\mathsf{pk}, c, \pi)$.

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***, ***Unforgeability***, ***IND-CPA Security*** *and* ***Public Self-Incriminating Proof*** *(Definition 9).*

The **Correctness**, **Unforgeability**, and **IND-CPA Security** properties of our PSIPE primitive are identical to those of SIPE (defined in Definition 3) as specified in Definition 4, Definition 5, and Definition 6, respectively.

Public Self-Incriminating Proof. We formalize the public self-incriminating proof property in the context of a blockchain Γ in Definition 9 via a game $\mathsf{Game}^{\text{PUB-SIP}}_{\text{PSIPE},\mathcal{A},\Gamma}$, presented in Figure 5. 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 captured by requiring key registration, *i.e.,* each party must publish their public key on the public ledger together with a zero-knowledge proof of knowledge of a valid corresponding secret key.

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

Game steps:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$; // 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. $b' \leftarrow \mathcal{A}_5(\mathsf{st}_2, \mathsf{VIEW}'_{\mathcal{A}_4})$

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:

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

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

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

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. 5: Public Self-Incriminating Proof game for PSIPE scheme.

Definition 9 (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* $\mathsf{Adv}^{\text{PUB-SIP}}_{\text{PSIPE},\mathcal{A},\Gamma}$ *in* $\mathsf{Game}^{\text{PUB-SIP}}_{\text{PSIPE},\mathcal{A},\Gamma}$*, 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* $\mathsf{Succ}^{\text{PSIPE}}_{\text{EXT},i}$ *where*

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

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 [47].

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.

4.1 Construction of PSIPE

In this section, we present a concrete construction of encryption with public self-incriminating proof $\mathsf{PSIPE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Vf}, \mathsf{ProofExt})$, which we call $\mathbf{\Pi}_{\mathsf{PSIPE}}$. Our construction is described formally in Figure 6. 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 $\mathsf{SIG} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ (in Definition 15) and an extractable witness encryption scheme $\mathsf{eWE} = (\mathsf{Enc}, \mathsf{Dec})$ (in Definition 21) for the language $\mathcal{L}_{\Gamma^V}$ specified by the relation $\mathcal{R}_{\Gamma^V}$ (in Definition 2).

In Sec. 4.3, we observe that since we assume a PoS blockchain as a setup, the witness encryption scheme can be realized under standard assumptions via the *extractable Witness Encryption on Blockchain (eWEB)* notion of [37] using techniques from [37] and [12].

Overview of our $\mathbf{\Pi}_{\mathsf{PSIPE}}$. 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 a message m 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 finally defining our PSIPE ciphertext as $c = (\hat{c}, d)$.

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 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 $\mathtt{B}$ 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 $\hat{c}$.

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 [36] and ensure that the protocol cannot be abused to decrypt a ciphertext without publishing a SIP.

4.2 Security Analysis

We formally state the security of $\mathbf{\Pi}_{\mathsf{PSIPE}}$ in Theorem 1, which is proven in Appendix B.

Theorem 1. *Assuming that: (i) Γ is a blockchain protocol (as in Definition 31) with validity predicate V, the $(\alpha, \beta, \ell_1, \ell_2)$-distinguishable forking property (as in Definition 37) and associated predicate*

Construction of PSIPE Scheme: Π_{PSIPE}

Parameters: A security parameter λ.

Building-blocks: Our construction uses the following building blocks:

- A blockchain protocol $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ (as in Definition 31) with validity predicate V and 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 1).
- An extractable witness encryption scheme $\mathsf{eWE} = (\mathsf{Enc}, \mathsf{Dec})$ (defined in Sec. 2.1) for the language $\mathcal{L}_{\Gamma^V}$ specified by the relation $\mathcal{R}_{\Gamma^V}$ (defined in Definition 2).
- An EUF-CMA secure signature scheme $\mathsf{SIG} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ (described in Sec. 2.1).

- $\mathsf{KG}(1^\lambda)$:
  1. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.KG}(1^\lambda)$.
  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. A randomly chosen $d \in \{0,1\}^\lambda$ for the message m.
  3. Encrypt the message m as $\hat{c} \leftarrow \mathsf{eWE.Enc}(1^\lambda, \mathcal{L}_{\Gamma^V}, (\mathsf{pk}, d, \mathtt{B}), m)$, where $(\mathsf{pk}, d, \mathtt{B}) \in \mathcal{L}_{\Gamma^V}$.
  4. Output $c = (\hat{c}, d)$.
- $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$:
  1. Parse c as $(\hat{c}, d)$.
  2. Compute the self-incriminating proof as $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{pk}, \mathsf{sk}, d)$.
  3. Publish the self-incriminating proof π on the blockchain Γ by executing $\mathsf{Broadcast}(1^\lambda, (\mathsf{pk}, d, \pi))$.
  4. 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.
  5. Decrypt $\hat{c}$ with self-incriminating proof π and $\tilde{\mathtt{B}}$; $m \leftarrow \mathsf{eWE.Dec}(\hat{c}, (\pi, \tilde{\mathtt{B}}))$.
  6. Output (m, π).
- $\mathsf{Vf}(\mathsf{pk}, c, \pi)$:
  1. Parse c as $(\hat{c}, d)$.
  2. Output $\mathsf{SIG.Vf}(\mathsf{pk}, d, \pi)$.
- $\mathsf{ProofExt}(\mathsf{pk}, c)$:
  1. Parse c as $(\hat{c}, d)$.
  2. Run $\tilde{\mathsf{st}} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\tilde{\mathtt{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$.
  3. Find a block $B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1 + \ell_2}$ containing the record $(\mathsf{pk}, d, \pi) \in B^*$, and output π.
  4. Otherwise, if $(\mathsf{pk}, d, \pi) \notin \tilde{\mathtt{B}}$, then output $\perp$.

Fig. 6: Construction of Encryption with Public Self-Incriminating Proof.

$\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) \in \{0,1\}$ *(as in Definition 1), (ii)* eWE *is an extractable witness encryption scheme (as in Definition 21) for the language* $\mathcal{L}_{\Gamma^V}$ *specified by the relation* $\mathcal{R}_{\Gamma^V}$ *(Definition 2), (iii)* SIG *is a EUF-CMA secure signature scheme as per Definition 15. Then our protocol* $\mathbf{\Pi}_{\mathsf{PSIPE}}$ *in Figure 6 is a secure encryption with public self-incriminating proof scheme* PSIPE *as per Definition 8.*

4.3 Instantiating $\mathbf{\Pi}_{\mathsf{PSIPE}}$

In the construction of $\mathbf{\Pi}_{\mathsf{PSIPE}}$ in Figure 6 we employ a extractable witness encryption (eWE) scheme as a building block. Although we only require a eWE scheme that supports one specific language, it is an arguably complex language and no such schemes are known under standard assumptions. Hence, in order to obtain a concrete instantiation of $\mathbf{\Pi}_{\mathsf{PSIPE}}$, we build on the underlying blockchain-based public ledger to instantiate a eWE scheme for the language we require while using only standard assumptions.

It has been shown in [37] 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 $\mathbf{\Pi}_{\mathsf{PSIPE}}$. 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 [37] 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 non-interactive zero knowledge proof

of knowledge 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 eWEB in instantiating our $\mathbf{\Pi}_{\mathsf{PSIPE}}$ 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 [12] 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 [31]). We observe that this construction can be instantiated with the efficient publicly verifiable secret sharing scheme for random anonymous committees presented in [14], which also allows for the secret to be periodically re-shared towards a newly selected dynamic anonymous committee.

The key observation of [12], 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 [37] 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 [37], requiring encryptors to use this alternative secret sharing scheme, while the decryption remains the same. As observed in [12], 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 dynamic anonymous committees, who re-share this secret key towards new committees whenever a decryption happens or when parties leave the protocol execution (akin to a YOSO threshold encryption scheme [7]).

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.

5 Threshold Encryption with Self-Incriminating Proof

In this section, we introduce a novel primitive called "threshold encryption with self-incriminating proof" (TSIPE). A threshold encryption system assigns a key share to each of the n parties so that at least $t+1$ of these parties can decrypt a ciphertext. As in the standard public key encryption case, any set of t 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[5].

In order to argue about security in the threshold setting, we use the Distributed Adversary model and the notion of MPC-hard computation introduced by Dziembowski *et al.* [26], which we recall in Sec. 2.3. This model allows us to argue about security in a setting where all parties are potentially corrupted and may collude in order to decrypt arbitrary ciphertexts but must do so by means of an interactive protocol that evaluates a MPC-hard function. A distributed adversary is composed of multiple sub-adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_a$ who may interact with the others when performing an attack. When evaluating the MPC-hard function within an MPC protocol, the communication and computational overhead are captured by means of an oracle $\mathcal{O}_{\mathcal{H}}$, which the parties must call via "slow" queries. While each sub-adversary has a strict bound on the number of "slow" calls to this oracle, they may perform a much larger number of "fast" calls only bounded by their runtime, which models computing the MPC-hard function in the clear.

We provide a formal syntax and security definition of our primitive TSIPE in Sec. 5.1. Finally, we provide a construction of TSIPE in Sec. 5.2.

[5] A rational party can be incentivized to publish this proof of decryption.

Obtaining a Threshold Version of PSIPE. The goal of our TSIPE notion is to ensure that one party executing a threshold decryption protocol learns a SIP *without* requiring a public ledger as setup as in our previous PSIPE notion. Notice, however, that it is possible to trivially obtain a threshold version of our PSIPE construction, where the decryption power is shared among a number of parties and the SIP generated by a threshold decryption operation is guaranteed to be published. This can be done by realizing our PSIPE construction using a threshold signature protocol along with a suitable distributed key generation protocol to distribute secret shares of the signing key to the set of parties performing decryption. In this setting, a minimum number of parties must collaborate to generate the signature π on d that is used as SIP and that must be published on the public ledger in order to decrypt the eWE ciphertext containing the message. The rest of the construction does not need to be modified, since composable threshold signature protocols and distributed key generation protocols can be obtained in general for any signature scheme (in case efficient specific purpose protocols for a given signature scheme do not exist, they can be obtained via general purpose secure multi-party computation). In the remainder of this section, our goal is to obtain a notion of TSIPE where a public ledger is not required, and instead obtain a weaker guarantee that at least one party obtains a SIP by exploring the fact that a set of computationally constrained parties must execute an interactive threshold decryption protocol. While this notion does not guarantee that this SIP will be learned by an honest party, it significantly relaxes the setup assumptions we must rely on in the PSIPE setting.

5.1 Formal Syntax and Security Definitions

We first describe the syntax of TSIPE, followed by formal security definitions.

Definition 10 (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 computational 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 trust 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 threshold encryption scheme self-incriminating proof TSIPE *scheme must satisfy the following properties:* ***Correctness*** *(Definition 11),* ***Unforgeability*** *(Definition 12),* ***IND-CPA Security*** *(Definition 13), and* ***Self-Incriminating Proof Extractability*** *(Definition 14).*

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 11 (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,*

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

- ***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,*

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

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 7, 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 8. 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 12 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 7 and 8, 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. 7: 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 12 (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* $n, t \in \mathbb{N}$

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

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

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 7) 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 8) are negligible.*

IND-CPA Security: In Definition 13, 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 9.

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

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. $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
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. 9: IND-CPA Security Game for TSIPE scheme.

Definition 13 (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 9) 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 10.

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

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. $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
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. 10: Self-Incriminating Proof Security Game for TSIPE scheme.

Observe that unlike the work of Dziembowski *et al.* [26] described in Sec. 2.3, 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 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 14 (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 10 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* $\mathsf{mode} = \mathsf{fast}$ *(defined in Sec. 2.3).*

5.2 Construction

In this section, we present a concrete construction of our threshold encryption with self-incriminating $\mathsf{TSIPE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine}, \mathsf{Vf})$, which we call $\mathbf{\Pi}_{\mathsf{TSIPE}}$. The formal construction of our scheme $\mathbf{\Pi}_{\mathsf{TSIPE}}$ is described in Figure 11.

Threshold Encryption with Self-Incriminating Proof Scheme: $\mathbf{\Pi}_{\mathsf{TSIPE}}$

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: Our construction uses the following building blocks:

- A threshold encryption scheme $\mathsf{TE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Combine})$ described in Sec. 2.1.
- A commitment scheme $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit})$ described in Sec. 2.1.
- An MPC-hard function scratch described in Figure 1.
- A NIZKPoK proof system described in Sec 2.1.
- Random Oracles:
  - $\mathcal{H}_1 : \{0,1\}^{(n+\beta)\cdot(\alpha-\beta-2)} \to \{0,1\}^{\lambda}$.
  - $\mathcal{H}_2 : \{0,1\}^{(n+\beta)\cdot(\alpha-\beta-2)+\beta^2} \to \{0,1\}^{\lambda}$.
  - $\mathcal{H}_3 : \{0,1\}^{*} \to \{0,1\}^{\alpha-\beta-2}$.
  - $\mathcal{H}_4 : \{0,1\}^{*} \to \{0,1\}^{\lambda}$.

- **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 $\mathsf{cm}_{\mathsf{sk}_i} = \mathsf{CS.Commit}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i})$ 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. Choose two random s and z as: $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^{\beta}$.
  3. Search for β number of nonces by setting $\mathsf{cnt} = 1$, $w_1 = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$ and $i = 1$, and proceeding as follows:
     (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 do the following:
           A. Record w_i.
           B. Set $\mathsf{cnt} = \mathsf{cnt} + 1$, $w_{i+1} = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$ and $i = i + 1$, 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 $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$
  5. Compute ρ as $\rho = \mathcal{H}_1(s\|w)$
  6. Compute the following values:

$$c_1 \leftarrow \mathsf{CS.Commit}(\mathsf{ck}, (s\|m); \rho), \quad c_2 \leftarrow \mathsf{TE.Enc}\left(\mathsf{pk}_{\mathsf{TE}}, s\right), \quad c_3 = \mathcal{H}_2(s\|w\|q) \oplus m$$

  7. 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) and sk_i as $(\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$.
  2. Compute partial decryption $\nu_i \leftarrow \mathsf{TE.ParDec}(\mathsf{sk}_i^{\mathsf{TE}}, c_2)$
  3. 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 by computing as: $s \leftarrow \mathsf{TE.Combine}\left(\mathsf{pk}_{\mathsf{TE}}, \{\mu_i\}_{i\in T}\right)$.
  3. Search for β number of nonces by setting $\mathsf{cnt} = 1$, $w_1 = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$ and $i = 1$, and proceeding as follows:
     (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 do the following:
           A. Record w_i.
           B. Set $\mathsf{cnt} = \mathsf{cnt} + 1$, $w_{i+1} = \mathcal{H}_3(\mathsf{pk}\|s\|z\|\mathsf{cnt})$ and $i = i + 1$, 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 $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$
  5. Compute $\rho = \mathcal{H}_1(s\|w)$
  6. Retrieve the plaintext message as: $m = c_3 \oplus \mathcal{H}_2(s\|w\|q)$.
  7. Compute a self-incriminating proof π_i with $\mathsf{sk}_i = (\mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ as:[a]

$$\pi_i \leftarrow \mathsf{NIZKPoK}\left\{\left(s, \rho, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i}\right) \middle| \mathsf{CS.Commit}(\mathsf{ck}, (s\|m); \rho) = c_1 \wedge \left(\vee_{j\in[n]} \mathsf{CS.Commit}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i}) = \mathsf{cm}_{\mathsf{sk}_j}\right)\right\}$$

  8. 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 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. 11: Construction of Threshold Encryption with Self-Incriminating Proof

Overview of $\mathbf{\Pi}_{\mathsf{TSIPE}}$. We show a TSIPE construction, starting from a regular threshold encryption scheme and embed 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, we first sample two random $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^{\beta}$ as an input to an MPC-hard function $\mathsf{scratch}(s,z,\cdot)$ and 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 11), and we use its outputs to derive a one-time pad key as $\mathcal{H}_2(s\|w\|q)$ using a random oracle $\mathcal{H}_2$ 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,m), an encryption c_2 of s with the underlying threshold encryption scheme and $c_3 = \mathcal{H}_2(s\|w\|q) \oplus m$, akin to the technique of [27] but using the MPC-hard function to obtain the extra values w,q needed to derive the "one-time pad" key encrypting m.

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. Finally, we can retrieves the message as $m = c_3 \oplus \mathcal{H}_2(s\|w\|q)$.

Now notice that we can generate a SIP as a zero-knowledge proof showing knowledge of an opening (s,m) to commitment c_1 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 $\mathsf{cm}_{\mathsf{sk}_i} = \mathsf{CS.Commit}(\mathsf{ck},\mathsf{sk}_i^{\mathsf{TE}};\rho_{\mathsf{sk}_i})$ 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.

5.3 Security Analysis

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

Theorem 2. *Assuming that: (i)* TE *is an IND-CPA threshold encryption scheme as per Definition 24, (ii)* CS *is a secure commitment scheme as per Definition 18, (iii)* scratch *is a correct and secure MPC-hard function as per Figure 1, (iv)* NIZKPoK *is a secure non-interactive zero-knowledge proof of knowledge system as per Definition 27, 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 10 with η-bounded parties against a (δ,Υ)-distributed adversary $\mathcal{A}_{1,\mathsf{TSIPE}},\ldots,\mathcal{A}_{a,\mathsf{TSIPE}}$ (as defined in Sec. 2.3).*

6 Conclusion

In this work, we have started the foundational work on self-incriminating proofs of decryption; ensuring that *if* a ciphertext has been decrypted, then there is a public proof of this fact. We have shown how

to do this both in the standard public-key encryption case with our scheme encryption with public self-incrimination proof PSIPE construction, by leveraging a proof-of-stake based public ledger and in the threshold public-key encryption case with our scheme threshold encryption with public self-incrimination proof TSIPE construction, presenting a big step in the direction for trusting outsourcing of secret key material for encryption schemes. However, in the threshold setting our work only presents the first step in insuring that secure outsourcing of secret keys, as we have not considered *tracing* the leakage to any specific party. A promising direction for identifying cheating parties is integrating recent research in traitor tracing [17,35,41] or traceable secret sharing [11,38] into a TSIPE construction, which we leave as future work. Moreover, it is desirable to get a stronger notion of SIP unforgeability in the threshold case, ensuring that a non-qualified set of parties in the decryption committee cannot forge a SIP for a ciphertext even if they know its plaintext message and randomness. We leave achieving this stronger notion as future work. In both cases, future work also remains on how to incentivize parties to both be available and provide appropriate threshold decryption when requested, along with how to incentivize them *not* to misuse shares, something we believe to be achievable by means of financial incentives orchestrated by smart contracts on a blockchain.

References

1. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009). https://doi.org/10.1007/978-3-642-03356-8_31
2. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_9
3. Alwen, J., shelat, a., Visconti, I.: Collusion-free protocols in the mediated model. In: Wagner, D. (ed.) Advances in Cryptology – CRYPTO 2008. Lecture Notes in Computer Science, vol. 5157, pp. 497–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2008). https://doi.org/10.1007/978-3-540-85174-5_28
4. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1993). https://doi.org/10.1007/3-540-48071-4_28
5. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: Santis, A.D. (ed.) Advances in Cryptology – EUROCRYPT'94. Lecture Notes in Computer Science, vol. 950, pp. 92–111. Springer, Heidelberg, Germany, Perugia, Italy (May 9–12, 1995). https://doi.org/10.1007/BFb0053428
6. 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, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011). https://doi.org/10.1007/978-3-642-20465-4_11
7. 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
8. 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
9. 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
10. Boneh, D., Partap, A., Rotem, L.: Accountability for misbehavior in threshold decryption via threshold traitor tracing. Cryptology ePrint Archive (2023)
11. Boneh, D., Partap, A., Rotem, L.: Traceable secret sharing: Strong security and efficient constructions. IACR Cryptol. ePrint Arch. p. 405 (2024), https://eprint.iacr.org/2024/405
12. 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, Heidelberg, Germany, San Francisco, CA, USA (Apr 24–27, 2023). https://doi.org/10.1007/978-3-031-30872-7_7
13. 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, Heidelberg, Germany, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22969-5_6

14. 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, Heidelberg, Germany, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22963-3_22
15. 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
16. 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
17. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1994). https://doi.org/10.1007/3-540-48658-5_25
18. 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
19. 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. pp. 910–927. IEEE Computer Society Press, San Francisco, CA, USA (May 18–21, 2020). https://doi.org/10.1109/SP40000.2020.00040
20. 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, Heidelberg, Germany, Christ Church, Barbados (Feb 22–26, 2016)
21. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_38
22. 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, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78375-8_3
23. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1988). https://doi.org/10.1007/3-540-48184-2_8
24. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990). https://doi.org/10.1007/0-387-34805-0_28
25. 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
26. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38545-2_18
27. 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, Heidelberg, Germany, Espoo, Finland (May 31 – Jun 4, 1998). https://doi.org/10.1007/BFb0054115
28. 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, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46803-6_10
29. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014). https://doi.org/10.1007/978-3-662-44371-2_29
30. 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
31. 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, Part II. Lecture Notes in Computer Science, vol. 12826, pp. 64–93. Springer, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84245-1_3
32. 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

33. 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, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40084-1_30
34. 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
35. 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, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30620-4_21
36. 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, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017). https://doi.org/10.1007/978-3-319-70500-2_18
37. 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, Heidelberg, Germany, Virtual Event (Mar 8–11, 2022). https://doi.org/10.1007/978-3-030-97121-2_10
38. 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, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84252-9_24
39. 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
40. 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
41. 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
42. 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
43. Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 643–673. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017). https://doi.org/10.1007/978-3-319-56614-6_22
44. Project, O.P.: The oasis blockchain platform. Tech. rep., Oasis Protocol Foundation (2020)
45. Rondelet, A., Kilbourn, Q.: Threshold encrypted mempools: Limitations and considerations. arXiv preprint arXiv:2307.10878 (2023)
46. 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
47. 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
48. 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 15.

Definition 15 (Digital Signature). *A digital signature scheme is a tuple of* PPT *algorithms* $\mathsf{SIG} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ *defined as follows:*

1. $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{KG}(1^\lambda)$ *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 16) and* ***unforgeability*** *(Definition 17) (i.e., existentially unforgeable against adaptive chosen message attacks (EUF-CMA) [34]) described below.*

Definition 16 (Correctness). *A signature scheme* $\mathsf{SIG} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ *is correct if for any security parameter* λ *and any message* $m \in \{0,1\}^\lambda$*, we have that*

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

Definition 17 (Unforgeability). *A signature scheme* $\mathsf{SIG} = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Vf})$ *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 12) is negligible.*

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

Game steps:

1. Initialize an empty list $\mathcal{Q} \leftarrow \emptyset$
2. $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{KG}(1^\lambda)$
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. 12: 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 Commitment Scheme

Here, we recall the syntax for a commitment scheme.

Definition 18 (Commitment Scheme). *A commitment scheme* CS *consists of the tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Commit})$ *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. $\mathsf{cm} \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$*, 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 19) and* ***hiding*** *(Definition 20) described below.*

Definition 19 (Binding). *A commitment scheme* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit})$ *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* $\mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho) = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}'; \rho')$ *for* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$*.*

Definition 20 (Hiding). *A commitment scheme* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit})$ *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 13) 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. $\mathsf{cm}_b \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{s}_b; \rho)$
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. 13: 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 [30, 33] as presented in [29].

Definition 21 (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 22) and* ***Extractable Security*** *(Definition 23) described below.*

Definition 22 (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 23.

Definition 23 (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 14) 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* $(\mathsf{inst}, \mathsf{wit}) \in \mathcal{R}_{\mathsf{eWE}}$*.*

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

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. 14: Extractable Security Game for eWE scheme.

A.4 Threshold Encryption

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

Definition 24 (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. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*, is a randomized algorithm that takes as input the public key* pk *and a message* m*, outputting a ciphertext* c*.*
3. $\mu_i \leftarrow \mathsf{ParDec}(\mathsf{pk}, \mathsf{sk}_i, c)$*, the partial decryption algorithm takes as input the public key* pk*, the secret key share* sk_i *and a ciphertext* c*, producing a partial decryption* μ_i*.*
4. $m/\perp \leftarrow \mathsf{Combine}(\mathsf{pk}, c, \{\mu_i\}_{i\in T})$*, 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* m *which* c *encrypts, otherwise output* $\perp$*.*

We require a threshold encryption scheme TE *satisfy the standard properties:* ***Correctness*** *(Definition 25) and* ***IND-CPA Security*** *(Definition 26) below.*

Definition 25 (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* $m \in \{0,1\}^\lambda$ *(from the permissible domain) where* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*, we have:*

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

Definition 26 (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 15) 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:

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

Fig. 15: IND-CPA Security Game for TE scheme.

A.5 NIZK Proof

We here outline the formal algorithms and definitions we assume for non-interactive zero-knowledge (NIZK) [9] proof system.

Definition 27 (Non-Interactive Zero-Knowledge Proof). *A non-interactive zero-knowledge proof system* NIZK *for an* **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*** *[9] proof system.*

Definition 28 (Completeness). *A NIZK proof system* $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for an* **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,*

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

Definition 29 (Soundness). *A NIZK proof system* $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for an* **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,*

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

Definition 30 (Zero-Knowledge). *A NIZK proof system* $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for an* **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,*

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

NIZK Proof-of-Knowledge (NIZKPoK) [4]. 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 $\mathcal{R}_{\mathsf{NIZK}}$.

A.6 Proof-of-Stake (PoS) Blockchains

In this section, we give an overview of the framework from [36] for arguing about PoS blockchain protocol security.

Blockchain Protocol Execution. We recall in almost verbatim form the overview given in [13] of the blockchain execution model from [36].

Definition 31 (Blockchain Protocol Execution [36]). *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.*
- $\mathsf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$*, the algorithm takes as input the security parameter* 1^λ *and state* st*, and outputs the longest sequence of valid blocks* 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 broadcast the message* m *over the network to all parties executing the blockchain protocol.*

At a very high level, execution of a blockchain protocol Γ^V is as follows: The participant in the protocol runs the $\mathsf{UpdateState}^V$ algorithm to get the latest blockchain state, the GetRecords algorithm is used to extract an ordered sequence of blocks encoded in the blockchain state variable and the 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 [13, 36]).

- 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 [43] supports adaptive corruption with erasures.

A note on admissible environments. Pass et al. [43] 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 [22], 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 [22] 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 $\mathcal{F}_{\mathtt{INIT}}$, 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 $\mathcal{F}_{\mathtt{INIT}}$ 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. We recall that running a blockchain protocol $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ with appropriate restrictions on $\mathcal{A}$ and $\mathcal{Z}$ will yield certain

compliant executions $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ where some security properties will hold with overwhelming probability. The existing works, including [28, 43], have converged towards a few security properties that characterizes blockchain protocols. These include: *Common Prefix* or *Chain Consistency*, *Chain Quality* and *Chain Growth*. From these basic properties, a number of stronger properties were derived in [36]. Among them, is the *Distinguishable Forking* property which will be the main requirement when introducing the "Encryption with Self-Incriminating Proof" scheme (in Sec. 3).

Definition 32 (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 33 (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 34 (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.*

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. 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. 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,*

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

B Proof of Theorem 1 (Π_{PSIPE} Security Proof)

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 $\mathbf{\Pi}_{\mathsf{PSIPE}}$ as follows:

Correctness. The correctness of $\mathbf{\Pi}_{\mathsf{PSIPE}}$ 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 $\mathbf{\Pi}_{\mathsf{PSIPE}}.\mathsf{KG}(\mathbf{1}^\lambda)$ as $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{SIG.KG}(1^\lambda)$ and for any message $m \in \{0,1\}^\lambda$, we encrypt the message by $\mathbf{\Pi}_{\mathsf{PSIPE}}.\mathsf{Enc}(\mathsf{pk},\mathbf{m})$ as $\hat{c} \leftarrow \mathsf{eWE.Enc}(1^\lambda, \mathcal{L}_{\Gamma^V}, (\mathsf{pk}, d, \mathtt{B}), m)$ where d is a reference signing message for the message m, and $\mathsf{st} \leftarrow \mathsf{UpdateState}(1^\lambda)$ and $\mathtt{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$. Finally, output the PSIPE ciphertext as $c = (\hat{c}, d)$

For decrypting a ciphertext $c = (\hat{c}, d)$ by $\mathbf{\Pi}_{\mathsf{PSIPE}}.\mathsf{Dec}(\mathsf{pk},\mathsf{sk},\mathbf{c})$, a decryptor first needs to generate a self-incriminating proof as $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{pk},\mathsf{sk},d)$, and then run the 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 $\mathsf{evolved}(\mathtt{B}, \tilde{\mathtt{B}}) = 1$ and that there exists a block $B^* \in \tilde{\mathtt{B}}^{\lceil \ell_1+\ell_2}$ such that $(\mathsf{pk}, d, \sigma) \in B^*$, so that with all but negligible probability, $\tilde{\mathtt{B}}$ and π can be used as the witness to decrypt ciphertexts $\hat{c}$ as $m \leftarrow \mathsf{eWE.Dec}(\hat{c}, (\pi, \tilde{\mathtt{B}}))$ where $((\mathsf{pk}, d, \mathtt{B}), (\pi, \tilde{\mathtt{B}})) \in \mathcal{R}_{\Gamma^V}$. Therefore, $m \leftarrow \mathsf{eWE.Dec}(c_2, (\pi, \tilde{\mathtt{B}}))$ follows from the correctness of the extractable witness encryption scheme as per Definition 21 and $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{pk},\mathsf{sk},d)$ follows from the correctness of the signature scheme as per Definition 15. Therefore, $\mathbf{\Pi}_{\mathsf{PSIPE}}$ satisfies the correctness condition.

Unforgeability. Assume, for the sake of contradiction, that we have an adversary $\mathcal{A}_{\mathsf{PSIPE}}$ that wins the game $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{PSIPE}}}$ (same as SIPE unforgeability game $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{PSIPE},\mathcal{A}_{\mathsf{SIPE}}}$ described in Figure 2) with non-negligible advantage when executing PSIPE from Figure 6. 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 12), 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 $(\mathsf{pk}, \cdot) \leftarrow \mathsf{SIG.KG}(1^\lambda)$, 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 $\pi \leftarrow \mathsf{SIG.Sign}(\mathsf{pk},\mathsf{sk},d)$ 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 (same as SIPE IND-CPA security game $\mathsf{Game}^{\mathsf{Unforge}}_{\mathsf{IND\text{-}CPA},\mathcal{A}_{\mathsf{SIPE}}}$ described in Figure 3). 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 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$. $\mathcal{A}_{\mathsf{eWE}}$ 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 Proof of Theorem 2 (Π_{TSIPE} Security Proof)

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 $\mathbf{\Pi}_{\mathsf{TSIPE}}.\mathsf{Setup}(\mathbf{1}^\lambda, \mathbf{n}, \mathbf{t})$ 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: $\mathsf{cm}_{\mathsf{sk}_i} = \mathsf{CS.Commit}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i})$ 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 $\mathbf{\Pi}_{\mathsf{TSIPE}}.\mathsf{Enc}(\mathsf{pk}, \mathbf{m})$ by sampling $s \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $z \xleftarrow{\$} \{0,1\}^\beta$, 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 11) 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: $c_1 \leftarrow \mathsf{CS.Commit}(\mathsf{ck}, (s\|m); \rho)$ where $\rho = \mathcal{H}_1(s\|w)$ and $w = (w_1\| \ldots \|w_\beta)$, $c_2 \leftarrow \mathsf{TE.Enc}(\mathsf{pk}_{\mathsf{TE}}, s)$ and $c_3 = \mathcal{H}_2(s\|w\|q) \oplus m$ where $q = (q_1\| \ldots \|q_\beta)$.

For the decryption of a ciphertext $c = (c_1, c_2, c_3, z)$ by $\mathbf{\Pi}_{\mathsf{TSIPE}}.\mathsf{Combine}(\mathsf{pk}, \mathsf{sk_i}, \mathbf{c}, \{\nu_\mathbf{i}\}_{\mathbf{i}\in[\mathbf{T}]})$, a set of $t+1$ or more parties do the following: (i) first, decrypt c_2 as $s \leftarrow \mathsf{TE.Combine}(\mathsf{pk}_{\mathsf{TE}}, \{\mu_i\}_{i\in T})$ and the correctness of it follows from the correctness of the threshold encryption scheme (as per Definition 24); (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 11) 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 $m = c_3 \oplus \mathcal{H}_2(s\|w\|q)$ where $\rho = \mathcal{H}_1(s\|w)$, $w = (w_1\| \ldots \|w_\beta)$ 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, ρ, m) which is sufficient to generate a self-incriminating proof π_i computed as a $\mathsf{NIZKPoK}$ with witness $(s, \rho, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$ showing $\mathsf{CS.Commit}(\mathsf{ck}, (s\|m); \rho) = c_1 \wedge \left(\vee_{j\in[n]} \mathsf{CS.Commit}(\mathsf{ck}, \mathsf{sk}_i^{\mathsf{TE}}; \rho_{\mathsf{sk}_i}) = \mathsf{cm}_{\mathsf{sk}_j}\right)$, and the correctness of self-incriminating proof π follows from the underlying $\mathsf{NIZKPoK}$ scheme (as per Definition 27) and commitment scheme CS (as per Definition 18). 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 $c_2 = \mathsf{TE.Enc}(\mathsf{pk}_{\mathsf{TE}}, s)$. The hybrid H_1 is the same as hybrid H_0 except that instead of generating a thresholds encryption according to the above process, we just sample a random string in $\{0,1\}^{n\cdot(\alpha-\beta-2)}$ 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 $c_1 = \mathsf{CS.Commit}(\mathsf{ck}, (s\|m); \rho)$ where $\rho = \mathcal{H}_1(s\|w)$ and $w = (w_1\| \ldots \|w_\beta)$. 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 $\{0,1\}^{n\cdot(\alpha-\beta-2)+\lambda}$ 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 $c_3 = \mathcal{H}_2(s\|w\|q) \oplus m$ 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 $q_i \leftarrow \mathsf{scratch}(s, z, w_i)$ are 0. Then it encrypt the message m as $c_3 = \mathcal{H}_2(s\|w\|q) \oplus m$ 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 random oracle model.

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 11. 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.

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, s_1) to the challenger of $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$ where $s_0 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $s_1 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$. Next, sample $z_0 \xleftarrow{\$} \{0,1\}^\beta$ and $z_1 \xleftarrow{\$} \{0,1\}^\beta$, and 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. $\mathcal{A}_{\mathsf{TE}}$ 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 $c_b = (c_1, c_2, c_3, z)$ to $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ where: $c_2 = c'_b$ and guess b^* to construct,

$$c_1 \leftarrow \mathsf{CS.Commit}(\mathsf{ck}, (s_{b^*} \| m_{b^*}); \rho), \quad c_3 = \mathcal{H}_2(s_{b^*} \| w_{b^*} \| q_{b^*}) \oplus m_{b^*}, \quad z = z_{b*}$$

where $\rho = \mathcal{H}_1(s_{b^*} \| w_{b^*})$.
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 $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\mathsf{TE},\mathcal{A}}$.

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 $((s_0, m_0)_0, (s_1, m_1)_1)$ to the challenger of $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$ where $s_0 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$ and $s_1 \xleftarrow{\$} \{0,1\}^{n\cdot(\alpha-\beta-2)}$). Next, sample $z_0 \xleftarrow{\$} \{0,1\}^\beta$ and $z_1 \xleftarrow{\$} \{0,1\}^\beta$, and 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 $r_2 \in \{0,1\}^{n\cdot(\alpha-\beta-2)+\lambda}$ and construct $c_2 \leftarrow \mathsf{TE.Enc}(\mathsf{pk}_{\mathsf{TE}}, (r_2))$; and choose a random $r_3 \in \{0,1\}^{\lambda}$ and 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 $\mathsf{Game}^{\mathsf{HIDE}}_{\mathsf{CS},\mathcal{A}}$.

In hybrid H_3, instead of generating an encryption $c_3 = \mathcal{H}_2(s\|w\|q) \oplus m$, 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 in the random oracle model, 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, w, q) since s are no longer in c_1 or c_2. 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, 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 11. 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 $(s, \rho, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$. Hence, this adversary is generating π for a ciphertext c_b it cannot decrypt in order to obtain the encryption randomness and message (s, ρ, 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 29) property of the NIZK scheme as per Definition 27 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 $(s, \rho, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$, which violates the soundness and

proof of knowledge properties of the NIZKPoK proof system. Thus, the existence of $\mathcal{A}_{1,\mathsf{TSIPE}}, \ldots, \mathcal{A}_{a,\mathsf{TSIPE}}$ that has non-negligible advantage in $\mathsf{Game}^{\mathsf{Unforge2}}_{\mathsf{TSIPE},\mathcal{A}_1,\ldots,\mathcal{A}_a}$ contradicts the security properties of 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, ρ, m) (*i.e.,* the encryption randomness and plaintext message) 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 $(s, \rho, m, \mathsf{sk}_i^{\mathsf{TE}}, \rho_{\mathsf{sk}_i})$, 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 $\mathcal{H}_2(s\|w\|q)$ where $w = (w_1\| \ldots \|w_\beta)$ and $q = (q_1\| \ldots \|q_\beta)$, for values $s\|w\|q$ that have high entropy and thus cannot be brute forced by a polynomial time adversary. More specifically, this input to the random oracle $\mathcal{H}_2$ 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 $c = (c_1, c_2, c_3, z)$ 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 $\rho = \mathcal{H}_1(s\|w)$ using (s, w) where $w = (w_1\| \ldots \|w_\beta)$. Finally, the extractor kEXT_i can compute $m = c_3 \oplus \mathcal{H}_2(s\|w\|q)$ 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.