Tetris! Traceable Extendable Threshold Ring Signatures and More

Gennaro Avitabile[1], Vincenzo Botta[2], and Dario Fiore[1]

[1] IMDEA Software Institute, Madrid, Spain. {gennaro.avitabile,dario.fiore}@imdea.org
[2] Sapienza University of Rome, Rome, Italy. vincenzo.botta@uniroma1.it

Abstract. Traceable ring signatures enhance ring signatures by adding an accountability layer. Specifically, if a party signs two different messages within the protocol, their identity is revealed. Another desirable feature is *extendability*. In particular, *extendable threshold* ring signatures (ETRS) allow to *non-interactively* update already finalized signatures by enlarging the ring or the set of signers. Combining traceability and extendability in a single scheme is unexplored and would offer a new tool for privacy-preserving voting schemes in scenarios where the voters are not known in advance. In this paper, we show how to reconcile both properties by introducing and constructing a new cryptographic primitive called Tetris. Notably, our Tetris construction simultaneously achieves strong anonymity and linear-size signatures, which is the main technical challenge in existing techniques. To solve this challenge, we develop a new approach to traceability that leads to several conceptual and technical contributions. Among those, we introduce and construct, based on Groth-Sahai proofs, *extendable* shuffle arguments that can be *non-interactively* updated by several provers.

Keywords: Ring Signatures · Traceability · Malleable Proof Systems.

1 Introduction

A ring signature scheme allows a signer to generate a signature on behalf of a publicly known group of potential signers called ring $\mathcal{R}$ [45]. In ring signatures, the identity of the signer is hidden among all the possible signers in the ring. Ring signatures are crucial in many applications including anonymous authentication [42], privacy-protecting cryptocurrencies [48], whistleblowing [45], and e-voting [49,46]. An interesting feature of ring signatures is that, unlike group signatures [25,12], they operate without the need of an online central authority. This feature is extremely useful in decentralized scenarios where such authority simply does not exist. However, ring signatures offer an unlimited level of anonymity which is not always desirable. For example, consider a simple voting protocol where voters have to pick between two choices. To cast a vote, a user simply produces a signature of the desired choice using their own secret key. A dishonest voter might exploit the high level of anonymity of ring signatures to cast a double vote without being detected. To tackle these issues, variants of ring signatures known as *linkable* ring signatures [39] and *traceable* ring signatures have been proposed [30,31]. The former allows one to establish whether two signatures have been produced by the same user or not. The latter has additional features. In traceable ring signatures, a signature on a message m is also issued w.r.t. a topic τ (e.g., a unique identifier of an election). Given two valid signatures w.r.t. the same topic τ and different messages, one can trace the public key of the signer who generated both signatures. If τ and m are equal in both signatures, the tracing procedure just reveals that they were both generated by the same signer, without revealing their identity. On the other hand, signatures issued w.r.t. different topics are unlinkable. A traceable ring signature must also be *exculpable*, meaning that it is unfeasible to produce two signatures to *falsely* accuse a user of having signed two different messages on the same topic. The level of accountability offered by traceability is strictly stronger than linkability. Indeed, aside from preventing malicious behavior, they enable identification of any dishonest party.

Threshold ring signatures generalize ring signatures allowing $t \geq 1$ signers to hide their identity within a ring of size $n \geq t$. The signature guarantees that at least t *different* signers in the ring signed the message without revealing which ones. Generally, threshold ring signatures require fixing the ring and the threshold at signature generation time with no possibility of further modifying them without a new intervention of the signers themselves. This is a significant limitation as all the potential signers must be known in advance.

More recently, the concept of Extendable Threshold Ring Signatures (ETRS) was introduced in [5]. ETRS allow to non-interactively update a threshold ring signature on a certain message so that the updated signature has a greater threshold and/or an augmented ring. This is done via two operations named *join* and *extend* respectively. A key property of ETRS is the *strong anonymity* notion defined in [8]. The adversary can see all the signature's updates resulting from threshold increments and ring extensions that a signature undergoes from when it is generated up until the protocol ends. This property is crucial for voting since all the updates are publicly available. Despite the main application of ETRS being voting (when participants are not known in advance), all the known ETRS providing even the weaker notion of linkability (i.e., [4, Sec. 5.3]) have "quadratic" $O(tn)$ size.

1.1 Our Contributions

We formalize and construct the first Traceable Extendable Threshold Ring Signature ($\mathfrak{Tetris}$). Our first contribution is a general definition of traceable ring signatures that takes into account the threshold setting as well as extendability. Indeed, to the best of our knowledge, previous definitions of traceability are only for (plain non-threshold i.e., with $t = 1$) ring signatures and allow tracing only between signatures having the same ring [30]. Simple adaptations of previous approaches fail at capturing the idea that *all* the misbehaving signers should be discovered (and not just a subset of them). Therefore, we adopt a fundamentally different approach in defining traceability (see Sec. 3 for more details).

The main technical challenge we solve is to construct a linear-size $\mathfrak{Tetris}$. Indeed, there are straightforward solutions to build a $\mathfrak{Tetris}$ with signatures of "quadratic", $O(tn)$, size. On the other hand, linear-size traceability techniques for non-extendable threshold ring signatures are not applicable as they inherently break strong anonymity. Towards our $\mathfrak{Tetris}$, we introduce and construct the following building blocks:

- We propose the notion of extendable non-interactive proof systems (EP). EP model proof systems for a generalized threshold relation (e.g., they cover shuffle arguments), where a proof for a certain statement x can be updated into a new proof for a different but related statement x'. Our notion can be seen as a generalization, also adding new properties, of the ENIWI of [8]. Among those, we introduce a notion of zero knowledge for extendable proof systems and a mild, but useful, notion of extractability that allows to perfectly extract *a part of* the witness.
- We introduce and construct extendable shuffle arguments, namely arguments proving that a t-size list of clear-text elements is a permuted subset of the values encrypted in a list of $n \geq t$ ciphertexts. The proof can be updated by several (independent) provers that can re-randomize the ciphertexts, add a ciphertext to the list, or reveal a new clear-text element.
- We introduce and construct doubly-authentication-preventing tags (DAPT) which are *deterministic* tags that are tied to a public key, a topic, and a message. These tags are anonymous when issued on different topics, but two tags generated with the same key, on the same topic, and different messages reveal the corresponding public key.

1.2 Technical Overview

We first discuss some approaches to traceability and their shortcomings. Then, we give a high-level description of our techniques and how we use them to overcome those issues.

A simple quadratic solution. A candidate technique to get a $\mathfrak{Tetris}$ is to adapt a widely known compiler to transform *linkable* ring signatures into *linkable threshold* ring signatures supporting the join operation. In a nutshell, whenever two linkable ring signatures over the same message and ring do not link to the same signer, they can be concatenated together to produce a 2-out-of-n signature. As a result, a signature with threshold t is composed of t ring signatures. Similarly, two traceable ring signatures can be concatenated if they do not trace to any public key in the ring. Therefore, applying this compiler starting from a *traceable ring* signature naturally gives a *traceable threshold* ring signature equipped with the join operation. If the base traceable ring signature also supports the extend operation, then we immediately get a $\mathfrak{Tetris}$. Thus, a viable approach

to get a $\mathfrak{Tetris}$ could involve modifying an existing traceable ring signature into one supporting extensions, and then applying the above compiler to get a $\mathfrak{Tetris}$. Unfortunately, currently known approaches [5,8] to get extendability are inherently linear in n, leading to a base extendable traceable ring signature with size $O(n)$ and thus to a $\mathfrak{Tetris}$ of size $O(tn)$.

Towards a linear-size $\mathfrak{Tetris}$*.* A natural starting point is to combine a strongly anonymous $O(n)$-size ETRS [8] with traceability techniques of previous works [30,29,18]. The key idea of these works is to publish a pseudo-random tag for each of the public keys in the ring. Whenever pk_i is one of the signers, its tag is uniquely determined by pk_i and the topic τ, while the tags of non-signers are just placeholders indistinguishable from legitimate tags. To trace a pair of signatures, it suffices to look for identical tags in both signatures. However, this approach is at odds with strong anonymity. Indeed, an adversary who has access to the full evolution of a signature can easily guess which signers have performed a join operation along the way. This is because once a *new* signer joins a signature, it is necessary to replace its placeholder tag with a new one as there exists only one valid tag for a key that is in the set of signers. Moreover, for the same reason, it is not possible to change the tags related to the previous signers. Given a pair of signatures (σ_{j-1}, σ_j) before and after the j-th join operation, the signers of σ_{j-1} are the public keys corresponding to the tags that have remained unchanged in σ_j. If the tags were re-randomizable, one could avoid such an attack by re-randomizing the list of tags after every join operation so that every signature in the sequence contains seemingly unrelated tags. However, since the tracing algorithm must work exclusively using publicly available information, it seems unlikely that one could come up with re-randomizable traceability tags that when compared reveal the dishonest signers.

An alternative approach is to break the link between tags and public keys and publish only t tags (i.e., one tag per signer). The signature also contains a NIZK proof[3] that the t tags come from t public keys in the ring, while hiding the actual correspondence between individual tags and public keys. Notice that the above attack is not applicable anymore, as no information is published about non-signers. We remark that the NIZK proof should be *extendable*, allowing future signers to extend the ring or to increase the threshold by providing a new tag. One could use extendable proofs for disjunctions and prove the above statement via t disjunctive proofs stating that each of the t tags comes from one of the n public keys in the ring. Known extendable proofs ([8]) support disjunctive threshold relations, that are proofs stating that t out of n base statements $(x_1, \ldots, x_n)$ are in a base language $\mathcal{L}$. These proofs have size $O(n\mathsf{S})$[4], where S is the size of a proof for $x_i \in \mathcal{L}$. For example, such relations can be used to express the above predicate via t disjunctive proofs stating that each of the t tags is generated w.r.t the topic τ and the message m using one of the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$. This approach gives again a signature of size $O(tn)$ (i.e., t proofs of size $n\mathsf{S}$, with S independent from t and n).

In this work, we start from the idea of publishing only t tags and we build new tools allowing us to instantiate the above template to get $O(n)$-sized signatures providing both traceability and strong anonymity. In the next paragraphs, we give an high-level overview of such building blocks and how we combine them.

Extendable non-interactive proof systems (EP*).* We propose the notion of extendable non-interactive proof systems (EP). EP model proof systems for a generalized threshold relation and are inspired from the ENIWI of [8]. ENIWI are defined w.r.t. a threshold relation, which is in turn defined w.r.t. a poly-time relation $R_{\mathcal{L}}$ as $R_{\mathcal{L}_{tr}} = \{(x = (k, x_1, \ldots, x_n), w = ((w_1, \alpha_1), \ldots, (w_k, \alpha_k))) | 1 \leq \alpha_1 < \ldots < \alpha_k \leq n \wedge \ \forall\ j \in [k] : (x_{\alpha_j}, w_j) \in R_{\mathcal{L}}\}$. Let $\mathcal{L}_{tr}$ be the corresponding NP-language and let us call *active* statements the statements that are in the set $\{x_{\alpha_j}\}_{j \in [k]}$; the other statements are called *inactive*. In words, the prover wants to prove that k *different* statements out of n are in the language $\mathcal{L}$. An ENIWI supports two kinds of transformations called extend and add operations:

[3] In this work, we interchangeably use the word proof and argument. Generally, unless specified, soundness is assumed to hold against a PPT adversary.

[4] Although using SNARKs in a recursive fashion one might get succinct extendable proofs [24,22], those proof systems are costly in terms of proving time and memory and rely on strong assumptions. We are instead interested in efficient provers and more standard assumptions.

– **Extend**: transform a proof for $(k, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$ into a proof for $(k, x_1, \ldots, x_n, x_{n+1}) \in \mathcal{L}_{tr}$.
– **Add**: transform a proof for $(k, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$ into a proof for $(k+1, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$.

The extend operation can be executed without involving any private input from the previous provers. However, the same does not apply to the add operation. When a prover computes a proof Π for a statement $x = (k, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$, it generates auxiliary values $\mathsf{AUX} = (\mathsf{aux}_1, \ldots, \mathsf{aux}_n)$ alongside the proof. The auxiliary value aux_i is used later to perform the add operation through an additional algorithm named PAdd. Given an accepting proof Π for $(k, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$, a witness w_i for an unused index i where $(x_i, w_i) \in R_{\mathcal{L}}$, and the corresponding auxiliary value aux_i, PAdd outputs a proof Π' for $(k+1, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$. Similarly, the algorithm PExt is employed for the extend operation, which does not need any auxiliary value. When provided with an accepting proof for $(k, x_1, \ldots, x_n) \in \mathcal{L}_{tr}$ and a statement x_{n+1}, PExt produces a proof Π' for $(k, x_1, \ldots, x_{n+1}) \in \mathcal{L}_{tr}$ and the auxiliary value aux_{n+1} associated with the statement x_{n+1}. This auxiliary value aux_{n+1} can subsequently be used to perform an add operation using witness w_{n+1} s.t. $(x_{n+1}, w_{n+1}) \in R_{\mathcal{L}}$. In [8] the notion of extended witness indistinguishability is introduced. It is a witness indistinguishability (WI) notion which allows the adversary to additionally get *some of* the auxiliary values.

The notion of EP supports more expressive types of threshold relations (see Sec. 4), for example it also models shuffle arguments. Unlike ENIWI, EP also allow to transform (e.g., re-randomize) the statements. Namely, after an extend/add operation, the base statements $x_1, \ldots, x_n$ can be transformed into different (but related according to a specific transformation) statements $x'_1, \ldots, x'_n$. Additionally, we introduce a definition of extended zero knowledge with a similar spirit to extended witness indistinguishability and a mild, but useful, notion of extractability (reminiscent of F-extractability [28, Def. 3]) that allows to perfectly extract what are the active statements along with *a function of* (part of) the rest of the witness (see Sec. 4).

ENIWI *internals.* To give a better understanding of our tools, we briefly describe the ENIWI of [8], which is based on the remarkable malleability of the Groth-Sahai (GS) proof system [37,23]. GS proofs are a commit-and-prove framework to prove the satisfiability of several types of equations over bilinear groups. In GS, secret variables are committed, and the prover generates proof elements from the committed values and commitment randomnesses. The proof is verified based on the statement, commitments, and proof elements. More concretely, partial knowledge of satisfying assignments for k out of n equations is proved by introducing additional *binary* "switch" variables bit_i s.t. when $\mathsf{bit}_i = 1$ the i-th equation is left unaltered, while when $\mathsf{bit}_i = 0$ the i-th equation admits the trivial solution, thus allowing for simulation. Then, an additional equation proving that $\sum_{i=1}^{n} \mathsf{bit}_i = k$ guarantees that only $n-k$ equations can be simulated, while the prover must hold a satisfying assignment for k of them.

The core idea of [8] is the observation that the proof elements of a GS proof are computed by linearly combining, along with some randomizers, the committed variables and the commitment randomnesses. This means that given (a function of) the values and the randomnesses of some committed variables, it is always possible to erase their contribution from a proof element in order to replace the old variables with freshly committed ones, assuming that the new assignment of the variables satisfies the new equation being proven[5].

Roughly speaking, auxiliary values correspond to commitment openings for these switch variables bit_i allowing to update a proof for $\sum_{i=1}^{n} \mathsf{bit}_i = k$ into a proof for $\sum_{i=1}^{n} \mathsf{bit}_i = k+1$ without knowledge of the other bit_j with $j \neq i$.

A key observation that we use is that several extendable proof systems can be connected to each other by sharing the same switch variables; this guarantees that the active indices (i.e, $i \in [n]$ s.t. $\mathsf{bit}_i = 1$) are the same in all proofs.

Extendable shuffle arguments. We introduce and construct an EP for the shuffle relation that we call extendable shuffle argument. The goal of an extendable shuffle argument is to prove that each element e_i of a public list $\{e_1, \ldots, e_k\}$ of $k \leq n$ elements is the value committed in a *different* commitment in a list $\{c_1, \ldots, c_n\}$ of n commitments. For $k = n$ the relation proved by an extendable shuffle argument coincides with a shuffle argument to known values (e.g., [3,43]), where one proves that a set of public values is obtained

[5] Even though the techniques shown in [8] are applied only to prove k out of n Pairing-Product Equations (PPEs), the same techniques apply with minimal adjustments to all the other equation types supported by GS.

by first permuting and then opening a set of public commitments. Formally, the relation for the extendable shuffle argument is the following:

$$R_{\mathsf{SH}} = \{(x = (c_1, \ldots, c_n, e_1, \ldots, e_k), w = (\phi, r_1, \ldots, r_n) | \forall\ i \in [k] : \\ \phi \text{ is an injective map } [k] \to [n] \wedge\ c_{\phi(i)} \leftarrow \mathsf{Com}(e_i, r_i)\}. \tag{1}$$

An extendable shuffle argument allows one to update previously generated proofs in the following two ways:

- **Extend**: On input a proof for $x = (c_1, \ldots, c_n, e_1, \ldots, e_k) \in \mathcal{L}_{sh}$, and a new commitment c_{n+1}, output a proof for $x' = (c'_1, \ldots, c'_n, c'_{n+1}, e_1, \ldots, e_k) \in \mathcal{L}_{sh}$.
- **Add**: On input a proof for $x = (c_1, \ldots, c_n, e_1, \ldots, e_k) \in \mathcal{L}_{sh}$, a new element e_{k+1}, a new commitment c'_i to e_{k+1} (along with its opening randomness) where position i has not been previously used, and an auxiliary value aux_i, output a proof for $x' = (c'_1, \ldots, c'_n, e_1, \ldots, e_k, e_{k+1}) \in \mathcal{L}_{sh}$.

Notice that we apply a further transformation to the statement when performing an add/extend operation by re-randomizing all the commitments (i.e., when we write c'_i we mean that c'_i is the re-randomization of c_i). This is an instantiation of our new notion of EP (see Sec. 4) that differs from the ENIWI of [8] where the base statements do not change after add/extend operations. This additional feature is crucial to guarantee the strong anonymity of our Tetris.

We build our extendable shuffle argument by modifying the shuffle argument by Groth and Lu [36] which is in turn based on GS proofs. We first adapt their work to asymmetric bilinear groups and then use the techniques of [8] to turn their regular shuffle into an extendable one by modifying the involved equations using switch variables (see Sec. 8 for more details). Our extendable shuffle satisfies our new notion of extended zero knowledge.

Doubly-authentication-preventing tags. We also define and construct new traceability tags that are compatible with our approach. We name such tags doubly-authentication-preventing tags (DAPT), inspired by doubly-authentication-preventing signatures (DAPS) [26,21]. A DAPS is a digital signature scheme with accountability. Messages consist of both an address and a payload component. If a signer signs two messages with identical addresses but different payloads, their secret key will be revealed. However, we cannot use DAPS as traceability tags since, being digital signatures, they are publicly verifiable and thus they reveal the public key w.r.t. they were generated.

Therefore, we define the related but different notion of DAPT. In a DAPT, a user is equipped with a public key and a secret key. There is a tag generation algorithm Tag that on input the secret key, a topic τ, and a message m deterministically generates a tag e. Unlike DAPS, an adversary who is only in possession of the public key cannot distinguish a valid DAPT from a random element of the tag space. Therefore, tags are not linkable to a specific signer. However, a DAPT is also provided with a tracing algorithm TagTrace that on input a public key and two tags generated on the same topic and different messages, outputs 1 if those tags were both generated w.r.t. the public key given in input. This property will allow us to trace malicious signers in our Tetris. Additionally, DAPT enjoy a non-frameability property stating that it is unfeasible to produce two tags that trace to a public key without having the corresponding secret key. This property will make our Tetris exculpable.

Putting everything together. To build a Tetris, we combine an EP for a threshold relation with our extendable shuffle argument (SH). The public key of the Tetris is composed by the public key of a DAPT and the public key of a public key encryption scheme, while the secret key simply includes the corresponding individual secret keys. To compute a signature on message m with tag τ, the first signer computes $e \leftarrow \mathsf{Tag}(\mathsf{sk}_i, \tau, m)$ and $c_i \leftarrow \mathsf{Com}(e; r)$ with randomness r. Then, it generates other commitments c_j with $j \in [n], j \neq i$ by committing to 0. The signature includes e as well as all the commitments. The signer then proves using the EP that 1 out of the n commitments actually commits to a tag that was honestly generated starting from τ, m, and one of the DAPT keys in the ring. The resulting auxiliary values are individually encrypted under the public keys of the corresponding potential future signers as done in [8].

Nevertheless, this does not suffice to get traceability as the EP proof above is about the value in c_i and not the clear-text e, i.e., e might not be in any of the commitments and just be a random value. Therefore, the signer uses SH to prove that the clear-text tag e actually comes from the commitments list. However, this is still not enough. Indeed, the EP proof certifies that one commitment contains the correct tag, while the SH proof certifies that the clear-text tag e corresponds to the content of one of the commitments, but in principle there is no guarantee that the two proofs are talking about the same commitment. To solve this issue we make EP and SH share the same switch variables bit_i so that we are guaranteed that the two proofs talk about the same commitments. This assures that all the t clear-text tags of a valid $\mathfrak{Tetris}$ actually correspond to honestly generated tags w.r.t. τ, m and t different keys in the ring.

The Trace algorithm of our $\mathfrak{Tetris}$ simply runs the $\mathsf{TagTrace}$ algorithm of the DAPT on all possible triplets of tags and public keys. Finally, the extend and join operations exploit the extendability of the underlying proof systems.

1.3 Related Work

Ring signatures were introduced by Rivest et al. [45]. Several works have focused on improving the signature size [27,2,44,38,52,6,41,51,34,9], the minimal assumptions required [1,15,40,10], the security against quantum adversaries [2,13,38,19,14]. For some applications (e.g., e-voting) plain ring signatures are not enough and additional properties are required. Accountability properties based on trusted authorities were explored for both group and ring signatures [50,17,32,16]. Liu et al.[39] introduced likability allowing *anyone* to determine if multiple signature were made by the same user without revealing their identity. Fujisaki et al. [30] introduced traceable ring signatures allowing deanonymization of a signer that produces multiple signatures on different messages. Subsequent works focused on improving the size or the assumptions of traceable ring signatures [29,7,19,47]. Threshold ring signature were introduced by Bresson et al. [20], in which t members of a group of n parties can sign a message keeping their identity hidden. Aranha et al. [5] have enhanced the functionality of threshold ring signatures by proposing ETRS. Avitabile et al. [8] proposed a stricter anonymity definition for ETRS along with a new construction based on Groth-Sahai (GS) proofs [37,33,28].

2 Notation and Preliminaries

Let $\mathcal{L}$ be an NP language, we call $R_{\mathcal{L}}$ the corresponding poly-time relation. We denote the security parameter as λ.We work over bilinear groups $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h}) \leftarrow \mathcal{G}(1^\lambda)$. $\mathcal{G}(1^\lambda)$ is a generator algorithm that on input the security parameter, outputs the description of a bilinear group. We call such description group key gk. $\hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}$ are prime p order groups, $\hat{g}, \check{h}$ are generators of $\hat{\mathbb{G}}, \check{\mathbb{H}}$ respectively, and $e : \hat{\mathbb{G}} \times \check{\mathbb{H}} \to \mathbb{T}$ is a non-degenerate bilinear map. In this paper, we will use additive notation for the group operations and multiplicative notation for the bilinear map e. Moreover, in some cases, we specify the randomness rand used by an algorithm A on an input x during the execution writing $A(x; \mathsf{rand})$. For details on standard tools (e.g., commitment schemes, public key encryption, non-interactive proof systems, and Groth-Sahai proofs) see App. A.

Assumption 1 (DDH) *The Decisional Diffie-Hellman (DDH) holds in $\hat{\mathbb{G}}$ if for all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ distinguishes the two distributions $(\hat{g}, \xi\hat{g}, \rho\hat{g}, \xi\rho\hat{g})$ and $(\hat{g}, \xi\hat{g}, \rho\hat{g}, \kappa\hat{g})$, where $\xi, \rho, \kappa \leftarrow\$ \mathbb{Z}_p$ is negligible. Tuples of the form $(\hat{g}, \xi\hat{g}, \rho\hat{g}, \xi\rho\hat{g})$ are called Diffie-Hellman (DH) tuples. The DDH problem in $\check{\mathbb{H}}$ is defined in a similar way.*

For the sake of convenience, we state the following assumption that we call Extended DDH. Notice that the Extended DDH assumption can be tightly reduced to DDH.

Assumption 2 (Extended DDH) *The Extended Decisional Diffie-Hellman (EDDH) holds in $\hat{\mathbb{G}}$ if for all PPT adversaries $\mathcal{A}$, the success probability in the experiment of figure Fig. 1 is*

$$\Pr\left[\mathsf{Exp}^{\mathrm{ExtDDH}}_{\mathcal{A},\hat{\mathbb{G}}}(\lambda) = win\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

$\mathsf{Exp}^{\text{ExtDDH}}_{\mathcal{A},\hat{\mathbb{G}}}(\lambda)$	$\mathcal{C}(b,a)$
$b \leftarrow\!\!\$\ \{0,1\}, a \leftarrow\!\!\$\ \mathbb{Z}_p, \hat{a} = a\hat{g}$	$t \leftarrow\!\!\$\ \mathbb{Z}_p, \hat{t} = t\hat{g}$
$b' \leftarrow \mathcal{A}^{\mathcal{C}(b,a)}(\hat{a})$	**if** $b = 0$:
if $b' = b$:	$\hat{z} = a\hat{t}$
return *win*	**else** :
else : **return** *lose*	$z \leftarrow\!\!\$\ \mathbb{Z}_p, \hat{z} = z\hat{g}$
	return $(\hat{t}, \hat{z})$

Fig. 1. Extended DDH experiment.

Assumption 3 (Type 2 Extended DDH) *The Type 2 Extended Decisional Diffie-Hellman holds in* $\hat{\mathbb{G}}$ *if for all* PPT *adversaries* $\mathcal{A}$*, the success probability in the experiment of figure Fig. 2 is*

$$\Pr\left[\mathsf{Exp}^{\text{ExtDDH-2}}_{\mathcal{A},\hat{\mathbb{G}}}(\lambda) = win\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

$\mathsf{Exp}^{\text{ExtDDH-2}}_{\mathcal{A},\hat{\mathbb{G}}}(\lambda)$	$\mathsf{ODDH}(b, a, \mathsf{honest})$
$b \leftarrow\!\!\$\ \{0,1\}, a \leftarrow\!\!\$\ \mathbb{Z}_p, \hat{a} = a\hat{g}$	$t \leftarrow\!\!\$\ \mathbb{Z}_p, \hat{t} = t\hat{g}$
$b' \leftarrow \mathcal{A}^{\mathcal{C}(b,a,\cdot)}(\hat{a})$	**if** $\mathsf{honest} = 1 \vee b = 0$:
if $b' = b$:	$\hat{z} = a\hat{t}$
return *win*	**else** :
else : **return** *lose*	$z \leftarrow\!\!\$\ \mathbb{Z}_p, \hat{z} = z\hat{g}$
	return $(\hat{t}, \hat{z})$

Fig. 2. Type 2 Extended DDH experiment.

Theorem 1. *If Ass. 2 holds w.r.t.* $\hat{\mathbb{G}}$*, then Ass. 3 also holds w.r.t.* $\hat{\mathbb{G}}$*.*

Proof. We construct an adversary $\mathcal{B}$ for Ass. 3 which internally runs $\mathcal{A}$ that breaks Ass. 2 with probability noticeably bigger than $\frac{1}{2}$. $\mathcal{B}$ forwards to $\mathcal{A}$ the $\hat{a}$ that it gets in input. Then, $\mathcal{B}$ replies to oracle queries made by $\mathcal{A}$ by making a query to its own $\mathcal{C}$ oracle. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. $\mathcal{B}$ perfectly simulates the view of an $\mathcal{A}$ playing in $\mathsf{Exp}^{\text{ExtDDH}}_{\mathcal{A},\hat{\mathbb{G}}}(\lambda)$ and thus wins in its game with the same advantage that $\mathcal{A}$ has in winning the $\mathsf{Exp}^{\text{ExtDDH}}_{\mathcal{A},\hat{\mathbb{G}}}(\lambda)$ game.

Assumption 4 (SXDH) *The Symmetric eXternal Diffie-Hellman (SXDH) assumption holds relative to* $\mathcal{G}$ *if there is no* PPT *adversary* $\mathcal{A}$ *that breaks the DDH problem in both* $\hat{\mathbb{G}}$ *and* $\check{\mathbb{H}}$ *for* $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h}) \leftarrow \mathcal{G}(1^\lambda)$*.*

Assumption 5 (Co-CDH) *The Co-Computational Diffie-Hellman assumption holds relative to* $\mathcal{G}$ *if for all* PPT *adversary* $\mathcal{A}$ *that are given* $(\hat{g}, \xi\hat{g}, \rho\hat{g}, \rho\check{h})$*, where* $\xi, \rho \leftarrow\!\!\$\ \mathbb{Z}_p$*, the probability that* $\mathcal{A}$ *outputs* $\hat{s}$ *s.t.* $\xi\hat{g} \cdot \rho\check{h} = \hat{s} \cdot \check{h}$ *is negligible.*

We also state the following two assumptions which we rely on in Sec. 8, namely the *subset* permutation pairing assumption and the *subset* simultaneous pairing assumption. They are inspired (and clearly implied)

by their non-subset (i.e., when $k = n$) analogues introduced in [36] for symmetric bilinear groups. The permutation pairing assumption was adapted to the asymmetric setting by González [35]. We adapt the simultaneous pairing assumption to the asymmetric setting as well.

Assumption 6 (Subset Permutation Pairing Assumption) *We say that the subset permutation pairing assumption holds if for* $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h}) \leftarrow \mathcal{G}(1^\lambda)$*, for all* $n \in \mathbb{N}, k \leq n$*, and for all* PPT *adversaries* $\mathcal{A}$ *we have:*

$$\Pr\left[\begin{array}{c|c} (\{\check{a}_i\},\{\check{b}_i\},\{\hat{c}_i\}) \leftarrow \mathcal{A}(\mathsf{gk}, k, \{\hat{g}_i\},\{\hat{\gamma}_i\},\{\check{h}_i\},\{\check{\delta}_i\}) & \\ \sum_{i=1}^k \check{a}_i = \sum_{i=1}^k \check{h}_i,\ \sum_{i=1}^k \check{b}_i = \sum_{i=1}^k \check{\delta}_i & x_1,\ldots,x_n \leftarrow Z_p, \\ \forall i \in [k]\ \hat{g}\cdot\check{a}_i = \hat{c}_i \cdot \check{h}, \hat{g}\cdot\check{b}_i = \hat{c}_i\cdot\check{a}_i & \{\hat{g}_i\} = \{x_i\hat{g}\}, \{\hat{\gamma}_i\} = \{2x_i\hat{g}\} \\ \{\check{a}_i\},\{\check{b}_i\} \textit{ are not a permutation of } \{\check{h}_i\},\{\check{\delta}_i\} & \{\check{h}_i\} = \{x_i\check{h}\}, \{\check{\delta}_i\} = \{2x_i\check{\delta}\} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Assumption 7 (Subset Simultaneous Pairing Assumption) *We say that the subset simultaneous pairing assumption holds if for* $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h}) \leftarrow \mathcal{G}(1^\lambda)$*, for all* $n \in \mathbb{N}, k \leq n$*, and for all* PPT *adversaries* $\mathcal{A}$ *we have:*

$$\Pr\left[\begin{array}{c|c} \{\hat{\mu}_i\} \leftarrow \mathcal{A}(\mathsf{gk}, k, \{\hat{g}_i\},\{\hat{\gamma}_i\},\{\check{h}_i\},\{\check{\delta}_i\}) & \\ \sum_{i=1}^k \hat{\mu}_i \cdot \check{h} = 0_{\mathbb{T}}, \sum_{i=1}^k \hat{\mu}_i \cdot \check{\delta}_i = 0_{\mathbb{T}} & x_1,\ldots,x_n \leftarrow Z_p \\ \exists i : \hat{\mu}_i \neq \hat{0} & \{\hat{g}_i\} = \{x_i\hat{g}\}, \{\hat{\gamma}_i\} = \{2x_i\hat{g}\} \\ & \{\check{h}_i\} = \{x_i\check{h}\}, \{\check{\delta}_i\} = \{2x_i\check{\delta}\} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

3 Traceable Extendable Threshold Ring Signatures

A $\mathfrak{Tetris}$ scheme consists of the following PPT algorithms and a poly-time relation $R_{\mathsf{key}}^{\mathfrak{Tetris}}$. We model signatures as a pair $\sigma = (\mu, \mathsf{tinfo})$, where we call tinfo tracing information as it is used by the Trace algorithm.

- $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$: outputs public parameters pp and trapdoor td.[6]
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$: generates a new public and secret key pair.
- $\sigma \leftarrow \mathsf{Sign}(\tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \mathsf{sk})$: on input a topic τ, a message m, a secret key sk corresponding to a public key pk_i with $i \in \mathcal{R}$, returns a signature σ.
- $0/1 \leftarrow \mathsf{Verify}(t, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma)$: on input a signature σ, a message m, a topic τ, a threshold t, and a set of public keys $\{\mathsf{pk}_i\}_{i\in\mathcal{R}}$; it outputs 1 to accept, and 0 to reject.
- $\sigma' \leftarrow \mathsf{Join}(\tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \mathsf{sk}, \sigma)$: on input a signature σ for message m and topic τ produced w.r.t. ring $\mathcal{R}$ with threshold t, and the new signer secret key sk (whose pk is in $\mathcal{R}$), outputs a new signature σ' with threshold $t+1$.
- $\sigma' \leftarrow \mathsf{Extend}(\tau, m, \sigma, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \{\mathsf{pk}_i\}_{i\in\mathcal{R}'})$: extends σ with threshold t for ring $\mathcal{R}$ into a new signature σ' with threshold t for ring $\mathcal{R} \cup \mathcal{R}'$.
- $T \leftarrow \mathsf{Trace}(\tau, m_1, m_2, \mathsf{tinfo}_1, \mathsf{tinfo}_2, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_1}, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_2})$: It is a *deterministic* algorithm. On input a topic τ, two messages and tracing information pairs $(m_1, \mathsf{tinfo}_1), (m_2, \mathsf{tinfo}_2)$, it outputs T that takes one of the following values: (i) indep; (ii) (linked, n); (iii) a set $\mathsf{PK} = \{\mathsf{pk}_i\}_{i\in\mathcal{R}\subseteq\mathcal{R}_1\cap\mathcal{R}_2}$. Where indep means that there are no common signers, (linked, n) means that $m_1 = m_2$ and there are n common signers, and when $m_1 \neq m_2$, PK is the set of traced keys.

We use the notion of ladder introduced in [5]. A ladder lad is a sequence of tuples $(\mathsf{action}, \mathsf{input})$, where action takes a value in the set $\{\mathsf{Sign}, \mathsf{Join}, \mathsf{Extend}\}$ and the value of input depends on the value of action. If $\mathsf{action} = \mathsf{Sign}$, then input is a pair $(\mathcal{R}, i)$, where $\mathcal{R}$ is the ring for the signature and i is the signer's identity. If $\mathsf{action} = \mathsf{Join}$, then input is an identifier i that identifies the signer joining the signature. If $\mathsf{action} = \mathsf{Extend}$, then input is a ring $\mathcal{R}$ that is the ring to use to extend the previous one. We use $\mathsf{lad}.\mathcal{S}$ to indicate the set of signers of a ladder lad. We notice that a ladder unequivocally determines a sequence of signatures, each one with a specific ring and threshold value. We also define the Proc algorithm, whose description is in (Fig. 3). Proc takes as input a topic, a message, a ladder, and a list of keys, and outputs the sequence of all the signatures corresponding to each step of the ladder. It outputs $\perp$ whenever the ladder is malformed. A $\mathfrak{Tetris}$ must satisfy all the following properties, in Fig. 4 we list the oracles used by our definitions.

[6] The public parameters pp produced by Setup are implicitly available to all the other algorithms.

$\mathsf{Proc}(\tau, m, \mathsf{L_{keys}}, \mathsf{lad})$

$\Sigma \leftarrow \emptyset, t = 0, \mathsf{Th} \leftarrow \emptyset$

parse $\mathsf{lad} = ((\mathsf{action}^1, \mathsf{input}^1), \ldots, (\mathsf{action}^l, \mathsf{input}^l))$

if $\mathsf{action}^1 \neq \mathsf{Sign}$: **return** $\perp$

else :

  parse $\mathsf{input}^1 = (\mathcal{R}^1, i^1)$

  for $j \in \mathcal{R}^1$;

    if $(j, \mathsf{pk}_j, \cdot) \notin \mathsf{L_{keys}}$: **return** $\perp$

  if $\mathsf{sk}_{i^1} = \perp \vee i^1 \notin \mathcal{R}^1$: **return** $\perp$

  $\mathcal{R} \leftarrow \mathcal{R}^1, \mathcal{S} \leftarrow \{i^1\}$

  $\sigma \leftarrow \mathsf{Sign}(\tau, m, \{\mathsf{pk}_j\}_{j \in \mathcal{R}}, \mathsf{sk}_{i^1})$

  $\Sigma \leftarrow \Sigma \cup \{\sigma\}, t \leftarrow 1, \mathsf{Th} \leftarrow \mathsf{Th} \cup \{t\}$

for $l' \in [2, \ldots, l]$:

  if $\mathsf{action}^{l'} = \mathsf{Sign}$: **return** $\perp$

  else :

    if $\mathsf{action}^{l'} = \mathsf{Join}$:

      parse $\mathsf{input}^{l'} = (i^{l'})$

      if $i^{l'} \notin \mathcal{R} \vee i^{l'} \in \mathcal{S}$: **return** $\perp$

      $\sigma \leftarrow \mathsf{Join}(\tau, m, \{\mathsf{pk}_j\}_{j \in \mathcal{R}}, \mathsf{sk}_i^{l'}, \sigma)$

      $\Sigma \leftarrow \Sigma \cup \{\sigma\}, \mathcal{S} \leftarrow \mathcal{S} \cup \{i^{l'}\}, t \leftarrow t + 1, \mathsf{Th} \leftarrow \mathsf{Th} \cup \{t\}$

    if $\mathsf{action}^{l'} = \mathsf{Extend}$:

      parse $\mathsf{input}^{l'} = (\mathcal{R}^{l'})$

      for $j \in \mathcal{R}^{l'}$:

        if $(j, \mathsf{pk}_j, \cdot) \notin \mathsf{L_{keys}}$: **return** $\perp$

      $\sigma \leftarrow \mathsf{Extend}(\tau, m, \sigma, \{\mathsf{pk}_j\}_{j \in \mathcal{R}}, \{\mathsf{pk}_j\}_{j \in \mathcal{R}^{l'}})$

      $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}^{l'}, \Sigma \leftarrow \Sigma \cup \{\sigma\}, \mathsf{Th} \leftarrow \mathsf{Th} \cup \{t\}$

    else : **return** $\perp$

return $(\Sigma, \mathsf{Th}, \mathcal{R})$

Fig. 3. The algorithm Proc for $\mathfrak{Tetris}$.

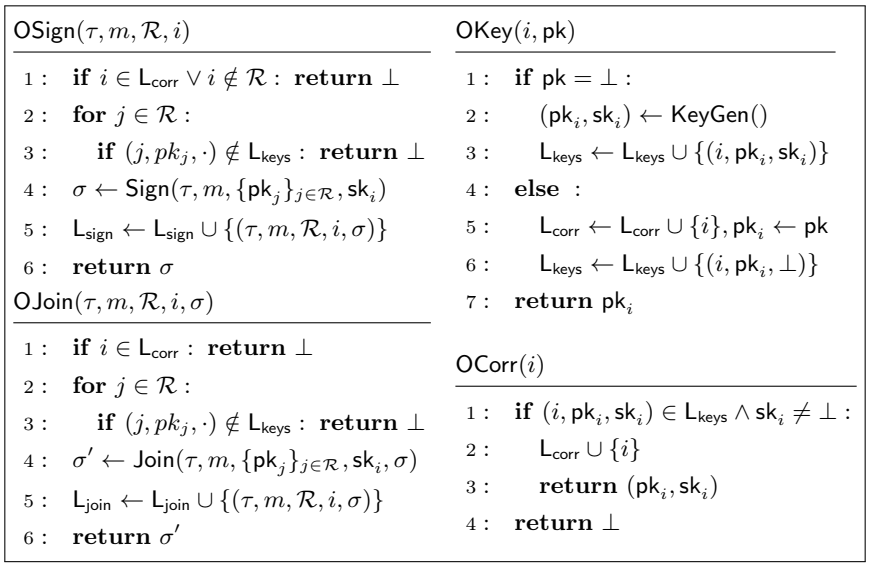

Fig. 4. Oracles for the security definitions of $\mathfrak{Tetris}$.

Definition 1 (Correctness). *For all $\lambda \in \mathbb{N}$, any message m, any topic τ, any ladder* lad *of polynomial size ℓ identifying a ring $\mathcal{R}$:*

$$\Pr\left[\begin{array}{c}\left(\bigwedge_{j=1}^{\ell} \mathsf{Verify}(t_j, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma_j) = 1\right) \\ \vee(\Sigma, \mathsf{Th}, \mathcal{R}) = \perp\end{array} \middle| \begin{array}{c}(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda); \\ \mathsf{L}_{\mathsf{keys}} \leftarrow \{\mathsf{KeyGen}()\}_{i\in\mathcal{R}}; \\ (\Sigma, \mathsf{Th}, \mathcal{R}) \leftarrow \mathsf{Proc}(\tau, m, \mathsf{L}_{\mathsf{keys}}, \mathsf{lad}); \\ \{\sigma_1, \ldots, \sigma_\ell\} = \Sigma; \\ \{t_1, \ldots, t_\ell\} = \mathsf{Th};\end{array}\right] = 1.$$

Definition 2 (Verifiability of Keys). *For all $\lambda \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, it holds that $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}()$ iff $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathfrak{Tetris}}$. We also require that for all pk there exists a* unique sk *s.t. $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathfrak{Tetris}}$.*

We define tracing correctness to model that, on honestly generated signatures, Trace should return the expected output (e.g., the set of common public keys if $m_1 \neq m_2$). Precisely, below we give a strong version of this notion that considers adversarially generated keys. This strong notion will be useful in our constructions and clearly implies the weaker one using honestly generated keys.

Definition 3 (Tracing Correctness). *For all* PPT $\mathcal{A}$ *and all $\lambda \in \mathbb{N}$ the success probability in the experiment $\mathsf{Exp}^{\mathrm{Tcorr}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 5) is $\Pr\left[\mathsf{Exp}^{\mathrm{Tcorr}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda) = win\right] \leq \mathsf{negl}(\lambda)$.*

On defining traceability. Previous definitions for $t = 1$ [30] require the adversary to output a topic τ, a ring $\mathcal{R}$ of size n, and $n + 1$ accepting message/signature pairs w.r.t. τ and $\mathcal{R}$. $\mathcal{A}$ generates all the keys by itself and wins the game if the signatures it produces look signed by different signers according to Trace. If the scheme is traceable, the adversary should not be able to win as there must exist at least a pair of signatures sharing the same signer. However, modeling traceability for general $t \geq 1$ is far more complicated. One may adapt the definition of [30] requiring $\mathcal{A}$ to provide enough signatures so that at least two of them must have at least c overlapping signers. However, unlike the $t = 1$ case where the size of the overlap is fixed, $\mathcal{A}$ may produce signatures with $c' > c$ overlapping signers. Therefore, such a definition would allow schemes tracing c keys while we want to identify *all* the $c' > c$ malicious signers. We solve this issue proposing a definition featuring an extractor Ext which is able to perfectly identify the guilty signers and whose output should match the one of Trace. In the following definition, we require the existence of such extractor.

$\mathsf{Exp}^{\mathrm{Tcorr}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$

1 : $(\mathsf{pp},\mathsf{td}) \leftarrow \mathsf{Setup}(\lambda)$
2 : $(\tau, m_0, m_1, \mathsf{lad}_0, \mathsf{lad}_1, \mathsf{L}_{\mathsf{keys}}) \leftarrow \mathcal{A}(\mathsf{pp})$
3 : $\forall (i, \mathsf{pk}_i, \mathsf{sk}_i) \in \mathsf{L}_{\mathsf{keys}} \wedge \mathsf{sk}_i \neq \bot :$
4 : **if** $(\mathsf{sk}_i, \mathsf{pk}_i) \notin R^{\mathfrak{Tetris}}_{\mathsf{key}}$: **return** *lose*
5 : $\forall (i, \mathsf{pk}_i, \mathsf{sk}_i), (j, \mathsf{pk}_j, \mathsf{sk}_j) \in \mathsf{L}_{\mathsf{keys}} \wedge \mathsf{sk}_i, \mathsf{sk}_j \neq \bot \wedge i \neq j :$
6 : **if** $\mathsf{pk}_i = \mathsf{pk}_j$: **return** *lose*
7 : $\forall b \in \{0,1\} : \mathsf{val}_b \leftarrow \mathsf{Proc}(\tau, m_b, \mathsf{L}_{\mathsf{keys}}, \mathsf{lad}_b)$
8 : **if** $\exists b \in \{0,1\} : \mathsf{val}_b = \bot$ **return** *lose*
9 : $\forall b \in \{0,1\}$: **parse** $\mathsf{val}_b = (\Sigma_b, \mathsf{Th}_b, \mathcal{R}_b)$
10 : $\forall b \in \{0,1\}$: **parse** $\Sigma_b = \{\sigma^b_1, \ldots, \sigma^b_{\ell_b} = (\mu_b, \mathsf{tinfo}_b)\}$
11 : $T \leftarrow \mathsf{Trace}(\tau, m_0, m_1, \mathsf{tinfo}_0, \mathsf{tinfo}_1, \{\mathsf{pk}_j\}_{j\in\mathcal{R}_0}, \{\mathsf{pk}_k\}_{k\in\mathcal{R}_1})$
12 : **if** $\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S} = \emptyset \ \wedge \ T \neq \mathsf{indep}$: **return** *win*
13 : **if** $m_0 \neq m_1 \wedge T \neq \mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S}$: **return** *win*
14 : **if** $m_0 = m_1 \wedge T \neq (\mathsf{linked}, (|\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S}|)$: **return** *win*
15 : **return** *lose*

Fig. 5. Tracing correctness game of $\mathfrak{Tetris}$. Tracing correctness game of $\mathfrak{Tetris}$. In line 6, if public keys are tuples, we say that two public keys are equal if they are equal in at least one position. This is justified as sampling the same value in the public key tuple basically requires sampling the same value in the secret key tuple (e.g., DL of a group element).

In the following definition, we require the existence of such extractor. On input the trapdoor and a valid signature, Ext must extract the signers of the signature. The existence of Ext is crucial to design a traceability experiment which is able to determine the right winning condition for its adversary.

Definition 4 (Signers Extraction). *There exists a* PPT *algorithm* Ext *that for all* $\lambda \in \mathbb{N}$, $(\mathsf{pp},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\Lambda = (t, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma = (\mu, \mathsf{tinfo}))$ *s.t.* $\mathsf{Verify}(\mathsf{pp}, \Lambda) = 1$ *works as follows:*

$$\mathsf{Ext}(\mathsf{td}, \Lambda) = \begin{cases} \bot \\ (\mathsf{PK}, \mathsf{Ind}). \end{cases}$$

Additionally, we require the following two properties to hold:

1. *Whenever* Ext *does not return* $\bot$*, it holds that:*

$$\Pr[(\Lambda, \mathsf{PK}) \in \mathcal{L}_{\mathsf{Ext}} \mid (\mathsf{PK}, \mathsf{Ind}) \leftarrow \mathsf{Ext}(\mathsf{td}, \Lambda)] = 1.$$

Where $\mathsf{PK}[i] = \mathsf{pk}_{\mathsf{Ind}[i]}$ *for* $i \in [t]$*, and* $\mathsf{pk}_{\mathsf{Ind}[i]} \in \{\mathsf{pk}_i\}_{i\in\mathcal{R}}$*. Moreover, the language* $\mathcal{L}_{\mathsf{Ext}} = \{x = (\Lambda, \mathsf{PK}) | \exists w = (\mathsf{sk}_{\mathsf{Ind}[1]}, \ldots, \mathsf{sk}_{\mathsf{Ind}[t]}, \mathsf{lad}, \mathsf{rand})$ s.t. $(\forall i \in [t]\ (\mathsf{sk}_{\mathsf{Ind}[i]}, \mathsf{pk}_{\mathsf{Ind}[i]}) \in R^{\mathfrak{Tetris}}_{\mathsf{key}}) \wedge \mathsf{P}(x, w) = 1\}$*, where* $\mathsf{P}(x, w)$ *computes:*
(a) $\mathsf{L}_{\mathsf{keys}} = [(j, \mathsf{pk}_j, \mathsf{sk}_j)_{j\in\mathcal{R}}]$ *as*

$$(j, \mathsf{pk}_j, \mathsf{sk}_j) = \begin{cases} (j, \mathsf{pk}_j, \mathsf{sk}_{\mathsf{Ind}[j]}) \text{ for } j \in \mathsf{Ind} \\ (j, \mathsf{pk}_j, \bot) \text{ for } j \in \mathcal{R} \setminus \mathsf{Ind}, \end{cases}$$

(b) $(\Sigma, \mathsf{Th}, \mathcal{R}) = \mathsf{Proc}(\tau, m, \mathsf{L}_{\mathsf{keys}}, \mathsf{lad}; \mathsf{rand})$.
(c) Return 1 iff $\Sigma = [\cdot, \ldots, (\mu', \mathsf{tinfo}')]$ *and* $\mathsf{tinfo}' = \mathsf{tinfo}$*, where* $\sigma = (\mu, \mathsf{tinfo})$.

2. For all $\lambda \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all PPT $\mathcal{A}$ it holds that:

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(\mathsf{pp}, \Lambda) = 1 \wedge \\ \mathsf{Ext}(\mathsf{td}, \Lambda) = \bot\end{array} \;\middle|\; (\Lambda) \leftarrow \mathcal{A}(\mathsf{pp})\right] \leq \mathsf{negl}(\lambda).$$

Intuitively, the property (1) above ensures that the extractor finds valid public keys, and that from their unique secret keys it is possible to generate signatures with the same tracing information tinfo' as the one in input, tinfo. Property (2) instead models that a PPT adversary cannot find a tuple that correctly verifies and makes the extractor fail.

We define traceability with a simple game that requires the adversary to output a pair of signatures and keys of its choice. The adversary wins the game if the output of Trace does not agree with the information derived from running Ext on both signatures.[7] Notice that $\mathcal{A}$ does not need any oracle in this game as it can adversarially sample keys with the knowledge of their secret key.

Definition 5 (Traceability). *There exists a PPT algorithm Ext s.t. for all PPT $\mathcal{A}$ and all $\lambda \in \mathbb{N}$, the success probability in $\mathsf{Exp}^{\mathrm{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 6) is $\Pr\left[\mathsf{Exp}^{\mathrm{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda) = \mathit{win}\right] \leq \mathsf{negl}(\lambda)$.*

Finally, we define anonymity and exculpability. The former models that an adversary should not learn which keys were used to generate a signature. Notably, we do this by letting the adversary choose two ladders and see the full evolution of the signature. The latter models that an adversary cannot falsely accuse a honest user of producing two signatures on the same topic, if the user did not do it. Slightly below we give more details on the checks performed in the experiments.

Definition 6 (Anonymity). *For all PPT $\mathcal{A}$ and all $\lambda \in \mathbb{N}$ the success probability in $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 6) is $\Pr\left[\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda) = \mathit{win}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$. The ladders output by $\mathcal{A}$ are well-formed if all the actions in the ladders are pairwise of the same type and they have the same ring as input.*

Definition 7 (Exculpability). *For all PPT $\mathcal{A}$ and all $\lambda \in \mathbb{N}$ the success probability in $\mathsf{Exp}^{\mathrm{Exculp}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 7) is $\Pr\left[\mathsf{Exp}^{\mathrm{Exculp}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda) = \mathit{win}\right] \leq \mathsf{negl}(\lambda)$.*

On the admissible adversaries of anonymity and exculpability. In our definitions there are some obvious checks that are performed by the experiment to exclude adversaries winning the game by just making use of the oracles. However, there is a less obvious check (line 10 of $\mathsf{Exp}^{\mathrm{Exculp}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 7) and line 11 of $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 6). Namely, any key for which $\mathcal{A}$ asked two queries over the same topic τ and two different messages $m_0 \neq m_1$ is considered as if it was corrupted. This is because, since the aim of traceable signatures is to prevent such behaviour, we allow for constructions with the additional feature that every signer who signs different messages loses any security guarantee. Finally, the check of line 10 of $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 6) excludes adversary that can easily distinguish both ladders using Trace.

Additional definitions and lemmas. We now give the definition of unforgeability and show that a $\mathfrak{Tetris}$ which is both traceable and exculpable is also unforgeable. Additionally, we show that tracing correctness and signers extraction imply traceability. As [5,8], we do not model malicious signers trying to prevent others from joining a signature.

Definition 8 (Unforgeability). *For all PPT adversaries $\mathcal{A}$ and for all $\lambda \in \mathbb{N}$ the success probability in the experiment $\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ of Fig. 8 is*

$$\Pr\left[\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda) = \mathit{win}\right] \leq \mathsf{negl}(\lambda).$$

[7] Notice that the Ext required in this definition operates as the extractor of signers extraction (Def. 4).

$\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$

$b \leftarrow\!\$\ \{0,1\}, \mathsf{L_{keys}}, \mathsf{L_{corr}}, \mathsf{L_{sign}}, \mathsf{L_{join}} \leftarrow \emptyset$
$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{O} \leftarrow \{\mathsf{OSign}, \mathsf{OKey}, \mathsf{OCorr}, \mathsf{OJoin}\}$
$(\tau^*, m^*, \mathsf{lad}_0^*, \mathsf{lad}_1^*) \leftarrow \mathcal{A}^{\mathsf{O}}(\mathsf{pp})$
$\Sigma \leftarrow \mathsf{Chal}_b(\tau^*, m^*, \mathsf{lad}_0^*, \mathsf{lad}_1^*)$
$b^* \leftarrow \mathcal{A}^{\mathsf{O}}(\Sigma)$
if $\exists\, i \in \mathsf{lad}_0^*.\mathcal{S} \cup \mathsf{lad}_1^*.\mathcal{S}$ s.t. $i \in \mathsf{L_{corr}}$:
  return *lose*
for $i \in \mathsf{lad}_0^*.\mathcal{S} \cup \mathsf{lad}_1^*.\mathcal{S}$:
  if $\exists\, (\tau^*, \cdot, \cdot, i, \cdot) \in \mathsf{L_{sign}} \cup \mathsf{L_{join}}$: **return** *lose*
  if $\exists\, (\tau, m_1, \cdot, i, \cdot), (\tau, m_2, \cdot, i, \cdot) \in \mathsf{L_{sign}} \cup \mathsf{L_{join}} \wedge m_1 \neq m_2$: **return** *lose*
if $b^* \neq b$: **return** *lose* **else** : **return** *win*

$\mathsf{Chal}_b(\tau^*, m^*, \mathsf{lad}_0^*, \mathsf{lad}_1^*)$

if $(\mathsf{lad}_0^*, \mathsf{lad}_1^*)$ is not well-formed :
  return $\perp$
if $\exists\, i \in \mathsf{lad}_0^*.\mathcal{S}$ s.t. $i \in \mathsf{L_{corr}}$:
  return $\perp$
if $\exists\, i \in \mathsf{lad}_1^*.\mathcal{S}$ s.t. $i \in \mathsf{L_{corr}}$:
  return $\perp$
$\mathsf{val}_0 \leftarrow \mathsf{Proc}(\tau^*, m^*, \mathsf{L_{keys}}, \mathsf{lad}_0^*)$
$\mathsf{val}_1 \leftarrow \mathsf{Proc}(\tau^*, m^*, \mathsf{L_{keys}}, \mathsf{lad}_1^*)$
if $\mathsf{val}_0 = \perp \vee \mathsf{val}_1 = \perp$:
  return $\perp$
parse $\mathsf{val}_0 = (\Sigma_0, \mathsf{Th}_0, \mathcal{R}_0)$
parse $\mathsf{val}_1 = (\Sigma_1, \mathsf{Th}_1, \mathcal{R}_1)$
return Σ_b

$\mathsf{Exp}^{\mathrm{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\tau, t_0, t_1, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_0}, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_1}, m_0, m_1, \sigma_0, \sigma_1) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{td})$
if $\exists b \in \{0,1\}$ s.t. $\mathsf{Verify}(t_b, \tau, m_b, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_b}, \sigma_b) = 0$: **return** *lose*
$\forall b \in \{0,1\} : \mathsf{out}_b \leftarrow \mathsf{Ext}(\mathsf{td}, \tau, m_b, t_b, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_b}, \sigma_b)$
if $\exists b \in \{0,1\}$ s.t. $\mathsf{out}_b = \perp$: **return** *win*
$\forall b \in \{0,1\}$: **parse** $\mathsf{out}_b = (\mathsf{PK}_b, \mathsf{Ind}_b)$
$\forall b \in \{0,1\}$: **parse** $\sigma_b = (\mu_b, \mathsf{tinfo}_b)$
$T \leftarrow \mathsf{Trace}(\tau, m_0, m_1, \mathsf{tinfo}_0, \mathsf{tinfo}_1, \{\mathsf{pk}_j\}_{j\in\mathcal{R}_0}, \{\mathsf{pk}_k\}_{k\in\mathcal{R}_1})$
if $\mathsf{PK} = \mathsf{PK}_0 \cap \mathsf{PK}_1 = \emptyset \ \wedge T \neq \mathsf{indep}$: **return** *win* **else** : **return** *lose*
if $m_0 \neq m_1$:
  if $\mathsf{PK} \neq T$: **return** *win* **else** : **return** *lose*
else :
  parse $T = (\mathsf{linked}, n)$
  if $n \neq |\mathsf{PK}|$: **return** *win* **else return** *lose*

Fig. 6. Anonymity and traceability games of $\mathfrak{Tetris}$.

$\mathsf{Exp}^{\mathrm{Exculp}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$

1 : $\mathsf{L}_{\mathsf{keys}}, \mathsf{L}_{\mathsf{corr}}, \mathsf{L}_{\mathsf{sign}}, \mathsf{L}_{\mathsf{join}} \leftarrow \emptyset, \mathsf{O} \leftarrow \{\mathsf{OSign}, \mathsf{OKey}, \mathsf{OCorr}, \mathsf{OJoin}\}$
2 : $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(\tau^*, t_0^*, t_1^*, \mathcal{R}_0^*, \mathcal{R}_1^*, m_0^*, m_1^*, \sigma_0^*, \sigma_1^*) \leftarrow \mathcal{A}^{\mathsf{O}}(\mathsf{pp})$
4 : **if** $\exists b \in \{0,1\}$ $\mathsf{Verify}(t_b^*, \tau^*, m_b^*, \{\mathsf{pk}_i\}_{i \in \mathcal{R}_b^*}, \sigma_b^*) = 0$: **return** *lose*
5 : $\mathsf{PK} \leftarrow \mathsf{Trace}(\tau^*, m_0^*, m_1^*, \sigma_0^*, \sigma_1^*, \{\mathsf{pk}_i\}_{i \in \mathcal{R}_0^*}, \{\mathsf{pk}_i\}_{i \in \mathcal{R}_1^*})$
6 : **if** $\mathsf{PK} = \mathsf{indep} \vee \mathsf{PK} = (\mathsf{linked}, \cdot)$:
7 : **return** *lose*
8 : $\forall\, \mathsf{pk}_i \in \mathsf{PK}$:
9 : **if** $\mathsf{pk}_i \in \mathsf{L}_{\mathsf{corr}}$: $\mathsf{PK} = \mathsf{PK} \setminus \mathsf{pk}_i$
10 : **elseif** $\exists (\tau, m_0, \mathcal{R}', \sigma', j), (\tau, m_1, \mathcal{R}'', \sigma'', j) \in \mathsf{L}_{\mathsf{sign}} \cup \mathsf{L}_{\mathsf{join}}$ s.t.
$m_0 \neq m_1 \wedge j \in \mathcal{R}_0 \wedge j \in \mathcal{R}_1 \wedge \mathsf{pk}_i = \mathsf{pk}_j$:
11 : $\mathsf{PK} = \mathsf{PK} \setminus \mathsf{pk}_i$
12 : **if** $\mathsf{PK} = \emptyset$: **return** *lose* **else** : **return** *win*

Fig. 7. Exculpability game of $\mathfrak{Tetris}$.

Lemma 1. *If a $\mathfrak{Tetris}$ is both exculpable (Def. 7) and traceable (Def. 5) then it is also unforgeable.*

Proof. Let us assume that there exists an adversary $\mathcal{A}$ that breaks the unforgeability property with non-negligible probability. We build an adversary $\mathcal{B}$ that wins the exculpability game with the same probability. Let $\mathcal{C}$ be the challenger of the exculpability game, $\mathcal{B}$ replies to all the oracle calls made by $\mathcal{A}$ by forwarding such calls to $\mathcal{C}$. $\mathcal{B}$ stores all the queries made by $\mathcal{A}$ along with the corresponding replies from $\mathcal{C}$. Let $(t^*, \tau^*, m^*, \mathcal{R}^*, \sigma^*)$ be the forgery output by $\mathcal{A}$. Let us focus on the queries that $\mathcal{A}$ performs on the public keys contained in $\mathcal{R}^*$.[8] The number of these queries is given by $q = c + q_1 + q_2$ where c is the number of corrupted keys in $\mathcal{R}^*$, q_1 is the number of $\mathsf{Join}/\mathsf{Sign}$ queries over (τ^*, m^*), and q_2 is the number of pairs of $\mathsf{Join}/\mathsf{Sign}$ queries over the same topic and different messages made w.r.t. the same public key. Let us call any of the above queries a compromising query, and let us refer to the public key of such a query as a compromised key. Notice that $q < t^*$, therefore, the maximum number of public keys in $\mathcal{R}^*$ on which $\mathcal{A}$ performed a compromising query is $t^* - 1$, thus there must exist a set $P \subseteq \mathcal{R}^*$ containing at least one *not* compromised key. Since the signature scheme achieves signer extraction (Def. 4), the probability that there are no public keys in P which is also in the set of signers[9] of σ^* is negligible (the reduction to signers extraction is very straightforward). For each $\mathsf{pk}_j \in P$, $\mathcal{B}$ looks at the queries made by $\mathcal{A}$ for a $\mathsf{Join}/\mathsf{Sign}$ query involving (τ^*, m, j), with $m \neq m^*$. If such a query exists, let $\sigma_j, \mathcal{R}_j, t_j$ be the signature, the ring, and the threshold resulting from the reply of $\mathcal{C}$ to such query. Otherwise, $\mathcal{B}$ queries $\mathcal{C}$ to generate a signature on $(\tau^*, m, \mathcal{R}^*, j)$, for arbitrary $m \neq m^*$ receiving back a signature σ_j over ring $\mathcal{R}_j = \mathcal{R}^*$ with threshold $t_j = 1$. For each $\mathsf{pk}_j \in P$, $\mathcal{B}$ runs $\mathsf{PK}_j \leftarrow \mathsf{Trace}(\tau^*, m^*, m, \mathsf{tinfo}^*, \mathsf{tinfo}_j, \mathcal{R}^*, \mathcal{R}_j)$, where tinfo^* and tinfo_j are taken from σ^* and σ_j respectively. Thanks to traceability (the reduction is very straightforward), there must exist at least one $\mathsf{pk}_j \in P$ such that $\mathsf{pk}_j \in \mathsf{PK}_j$, therefore $\mathcal{B}$ sends $(\tau^*, t^*, t_j, \mathcal{R}^*, \mathcal{R}^*, m^*, m, \sigma^*, \sigma_j)$ to $\mathcal{C}$. Assuming that the queries made by $\mathcal{B}$ do not cause the removal of any $\mathsf{pk}_j \in P$ from PK in lines 9 and 10 of the exculpability experiment (Fig. 7), then $\mathcal{B}$ wins the exculpability game with the same probability of $\mathcal{A}$ winning the unforgeability game. It remains to show that the queries carried out by $\mathcal{B}$ do not cause the removal of any $\mathsf{pk}_j \in P$ from PK in lines 9 and 10 of the exculpability experiment (Fig. 7). To show this, we observe that since $\mathsf{pk}_j \in P$ the following facts about the queries performed by $\mathcal{A}$ over pk_j hold:

[8] For simplicity, we say that a public key belongs to the ring instead of saying that it belongs to the set $\{\mathsf{pk}_j\}_{j \in \mathcal{R}^*}$.
[9] More precisely, there exists a ladder that includes this public key in the set of signers and leads to the tinfo^* contained in σ^*.

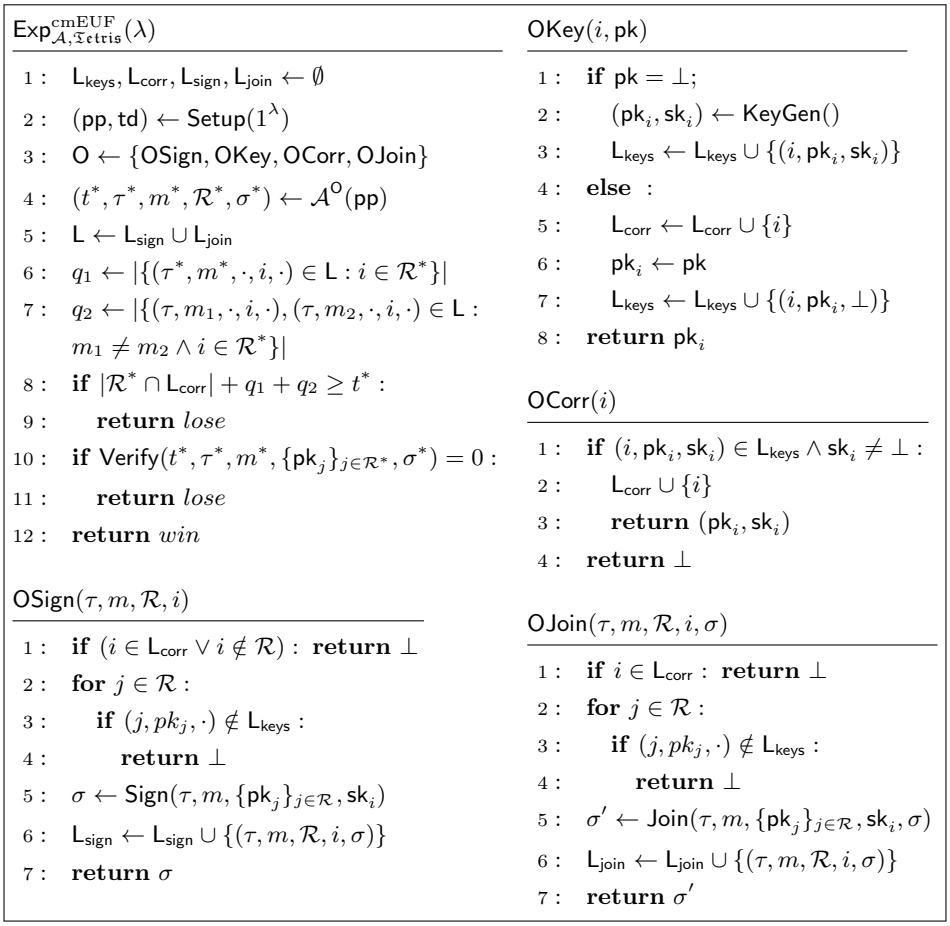

Fig. 8. Unforgeability game for $\mathfrak{Tetris}$ (security experiment and oracles).

– pk_j was never corrupted;
– no query over (τ^*, m^*) was ever asked for pk_j;
– no pair of $\mathsf{Join}/\mathsf{Sign}$ queries over the same topic and different messages were ever made for pk_j.

$\mathcal{B}$ does not perform any corruption query and it only asks, if not previously asked by $\mathcal{A}$, a query over (τ^*, m) with $m \neq m^*$. As a result, pk_j will not have pairs of $\mathsf{Join}/\mathsf{Sign}$ queries over the same topic and different messages and it will not be removed from PK in lines 9 and 10 of the exculpability experiment (Fig. 7).

Lemma 2. *Every $\mathfrak{Tetris}$ which has verifiability of keys (Def. 2), tracing correctness (Def. 3) and signers extraction (Def. 4) is traceable (Def. 5).*

Proof. The probability of $\mathcal{A}$ winning $\mathsf{Exp}^{\text{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$ is at most the sum of the probabilities of $\mathcal{A}$ being able to satisfy one of the following conditions:

1. $\mathcal{A}$ makes Ext output $\perp$ (i.e., line 4 of $\mathsf{Exp}^{\text{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 6));
2. $\mathcal{A}$ outputs two signatures σ_0, σ_1 s.t. PK extracted by Ext is empty but Trace does not output indep (i.e., line 9 of $\mathsf{Exp}^{\text{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$).
3. $\mathcal{A}$ outputs two signatures σ_0, σ_1 on different messages $m_0 \neq m_1$ which trace to a set of keys T different from the set of keys PK extracted by Ext (i.e., lines 10 and 11 of $\mathsf{Exp}^{\text{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$);
4. $\mathcal{A}$ outputs two signatures σ_0, σ_1 on the same message (i.e. $m_0 = m_1$) on which Trace returns (linked, n), but n is not equal to $|\mathsf{PK}|$, where PK is the set of keys extracted by Ext (i.e., lines 13-14 of $\mathsf{Exp}^{\text{Trace}}_{\mathsf{Ext},\mathcal{A},\mathfrak{Tetris}}(\lambda)$).

We argue that $\mathcal{A}$ is able to satisfy each of the above conditions only with negligible probability.

Let us consider case 1. The probability of $\mathcal{A}$ satisfying case 1 is negligible, otherwise we can break the signers extraction of $\mathfrak{Tetris}$ (Def. 4) by simply using one of the two signatures given in output by $\mathcal{A}$.

Let us now analyze the remaining cases. Since $\mathfrak{Tetris}$ satisfies signers extraction, we have that, there must exist two valid ladders, that, generate (via Proc), using the secret keys corresponding to the public keys extracted by Ext, two signatures containing *exactly* tinfo_0 and tinfo_1. Thus, due to tracing correctness of $\mathfrak{Tetris}$ (Def. 3) and the fact that Trace is deterministic, given σ_0 and σ_1 output by $\mathcal{A}$, the output of Trace is consistent with the one derived from Ext with overwhelming probability. Therefore, the probability of $\mathcal{A}$ satisfying one of case 2, 3, 4 is negligible.

4 Extendable Non-Interactive Proof Systems

We propose a more expressive poly-time relation than [8] called generalized threshold relation R_{GT} containing triples (ck, x, w), where $\mathsf{Com}_{\mathsf{ck}}$ is a commitment scheme with parameters ck and F is a (hard) predicate.

$$\begin{aligned} R_{\mathsf{GT}} = \{&(\mathsf{ck}, x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_k), w = (\phi, w_1, \ldots, w_k, \widetilde{r_1}, \ldots, \widetilde{r_n}))| \\ &\forall i \in [n]\ \exists\ \mathsf{bit}_i :\ \widetilde{c_i} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; \widetilde{r_i}) \wedge \mathsf{bit}_i \in \{0,1\} \wedge \textstyle\sum_{i=1}^{n} \mathsf{bit}_i = k \wedge \\ &\forall_{j=1}^{k}(\mathsf{bit}_{\phi(j)} = 1 \wedge F(\mathsf{ck}, x_{\phi(j)}, e_j, w_j) = 1) \wedge \phi : [k] \to [n] \wedge \phi \text{ is injective}\}. \end{aligned}$$

Two important specific cases are:

– By setting $\forall i \in [k]$ e_i to be empty strings and $F(\mathsf{ck}, x, \cdot, w) = R_{\mathcal{L}'}(x, w)$, we retrieve the threshold relation over $R_{\mathcal{L}'}$.
– By setting $F(\mathsf{ck}, x, e, w)$ to be the predicate that returns 1 iff $x = \mathsf{Com}_{\mathsf{ck}}(e; w)$, we retrieve the extendable shuffle relation (see relation (1)).

We call $\{\widetilde{c_i}\}_{i\in[n]}$ commitments to *switch variables* and $\{e_i\}_{i\in[k]}$ *clear-text* elements. Additionally, we call the set $\{x_{\phi(j)}\}_{j\in[k]}$ *active* statements, while $\{x_i\}_{i\in[n]} \setminus \{x_{\phi(j)}\}_{j\in[k]}$ *inactive*. Let $R_{\mathcal{L}}$ be an instantiation of R_{GT} where $\mathsf{Com}_{\mathsf{ck}}$ and F are concretely specified. An extendable non-interactive proof system EP also comes with *addition* ($\mathsf{AddT} = (\mathsf{AddT}_x, \mathsf{AddT}_w)$) and *extension* ($\mathsf{ExtT} = (\mathsf{ExtT}_x, \mathsf{ExtT}_w)$) transformations.

For example, $\mathsf{AddT}_x(\mathsf{ck}, x, e, \alpha, w_\alpha, \widetilde{r_\alpha}; r)$ is a transformation, with randomness r, that takes a statement $x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_{k-1})$, and transforms it to $x' = (k+1, x'_1, \ldots, x'_n, \widetilde{c'_1}, \ldots, \widetilde{c'_n}, e_1, \ldots, e_{k-1}, e)$. Namely, after an addition (or extension) operation the parts of the statement related to the threshold and the clear-text elements are deterministically modified independently from the randomness in input to the transformation, while the commitments and the base statements can be updated depending on the randomness (e.g., re-randomized). Analogously, AddT_w, on input the same transformation randomness given to AddT_x, transforms a witness w s.t. $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$ into a witness w' s.t. $(\mathsf{ck}, x', w') \in R_{\mathcal{L}}$. An EP for $R_{\mathcal{L}}$ consists of the following PPT algorithms along with transformations $\mathsf{ExtT}_x, \mathsf{ExtT}_w, \mathsf{AddT}_x, \mathsf{AddT}_w$. The group key $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$ is considered as an implicit input to all algorithms:

- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$: generates crs and trapdoor td.
- $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$: generates the commitment key ck.
- $c \leftarrow \mathsf{Com}_{\mathsf{ck}}(m; r)$: on input a message m and a randomness r, given the commitment key ck, returns a commitment c.
- $(\Pi, (\mathsf{aux}_1, \ldots, \mathsf{aux}_n)) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x, w)$: on input statement $x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_k)$, witness $w = (\phi, w_1, \ldots, w_k, \widetilde{r_1}, \ldots, \widetilde{r_n})$, s.t. $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$ outputs a proof Π and auxiliary values $(\mathsf{aux}_1, \ldots, \mathsf{aux}_n)$. Each aux_i is used later on to perform an add operation using a witness for a not previously used statement x_i.
- $0/1 \leftarrow \mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi)$: on input statement x, and a proof Π, outputs 1 to accept and 0 to reject.
- $(\Pi', x', \mathsf{aux}_{n+1}, (r_1, \ldots, r_{n+1})) \leftarrow \mathsf{PExt}(\mathsf{crs}, \mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}, \Pi, r)$: on input statements x, x_{n+1}, randomness $\widetilde{r_{n+1}}$ (used to commit to $\mathsf{bit}_{n+1} = 0$), a proof Π for $x \in \mathcal{L}$, and transformation randomness r, outputs an updated statement $x' = (k, x'_1, \ldots, x'_{n+1}, \widetilde{c'_1}, \ldots, \widetilde{c'_n}, e_1, \ldots, e_k)$, an updated proof Π' for $x' \in \mathcal{L}$, and randomnesses $r_1, \ldots, r_{n+1}$.
- $(\Pi', x', \mathsf{aux}'_\alpha, (r_1, \ldots, r_n)) \leftarrow \mathsf{PAdd}(\mathsf{crs}, \mathsf{ck}, x, w', \alpha, e, \widetilde{r_\alpha}, \mathsf{aux}_\alpha, \Pi, r)$: on input statement x, witness w', index α, element e, randomness $\widetilde{r_\alpha}$ (used to commit to $\mathsf{bit}_\alpha = 1$), auxiliary value aux_α, proof Π for $x \in \mathcal{L}$, and transformation randomness r, outputs an updated statement $x' = (k+1, x'_1, \ldots, x'_{n+1}, \widetilde{c'_1}, \ldots, \widetilde{c'_n}, e_1, \ldots, e_k, e)$, an updated proof Π' for $x' \in \mathcal{L}$, an updated auxiliary value aux'_α, and randomnesses $(r_1, \ldots, r_n)$.
- $\mathsf{aux}'_i \leftarrow \mathsf{AuxUpd}(\mathsf{crs}, \mathsf{ck}, \Pi, \mathsf{aux}_i, r_i)$: on input proof Π previous to addition or extension, the auxiliary value aux_i, and update randomness r_i, outputs updated auxiliary value aux'_i. AuxUpd is used to update the auxiliary values after an extend/add operation. The randomness given in input is the one given in output by either PExt or PAdd. To simplify the notation, we write $\mathsf{AUX}' \leftarrow \mathsf{AuxUpd}(\mathsf{crs}, \mathsf{ck}, x, \Pi, \mathsf{AUX}, r)$ to indicate that a list of auxiliary values is updated by appropriately parsing AUX and r and running the update operation on each element of the list.
- $0/1 \leftarrow \mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x, w, (\mathsf{aux}_1, \ldots, \mathsf{aux}_n), \Pi)$: on input statement x, witness w, auxiliary values $(\mathsf{aux}_1, \ldots, \mathsf{aux}_n)$, and proof Π, outputs 1 if the auxiliary values are consistent with the statement, the proof, and the witness. Returns 0 otherwise. If $\mathsf{AuxVerify}$ returns 1, we are guaranteed that the subsequent extend/add operations can be correctly performed[10].

An EP has to satisfy: (1) *completeness*, i.e., honestly generated proofs are always accepting and result in correct auxiliary values (w.r.t. $\mathsf{AuxVerify}$); (2) *soundness*, i.e., it is unfeasible to produce accepting proofs for false statements; (3) *admissible transformations*, i.e. transforming $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$ using AddT or ExtT with the intended inputs always gives $(\mathsf{ck}, x', w') \in R_{\mathcal{L}}$; (4) *transformation completeness*, i.e., addition and extension operations always give an accepting proof and correct auxiliary values when executed with the intended inputs. Furthermore, AddT_x (ExtT_x) and PAdd (PExt) must give the same x' in output when given in input the same r and x. Finally, $\mathsf{Com}_{\mathsf{ck}}$ must be perfectly binding and computationally hiding.

Definition 9 (Binding and Hiding of $\mathsf{Com}_{\mathsf{ck}}$). *For all $\lambda \in \mathbb{N}$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $\mathsf{Com}_{\mathsf{ck}}$ is a perfectly binding and computationally hiding commitment scheme.*

[10] We introduce $\mathsf{AuxVerify}$ merely as an internal utility to simplify the description of our definitions.

Definition 10 (Completeness). *An* EP *for* $R_{\mathcal{L}}$ *is complete if* $\forall\lambda \in \mathbb{N}$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$, $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$, *and* $(\Pi, \mathsf{AUX}) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x, w)$ *it holds that*

$$\Pr[\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi) = 1 \wedge \mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x, w, \mathsf{AUX}, \Pi) = 1] = 1$$

Definition 11 (Admissible Transformations). *The transformations* AddT_x, AddT_w, ExtT_x *and* ExtT_w *have the following properties:*

- AddT_x *is a transformation that on input* $(\mathsf{ck}, x, e, \alpha, w^*, \widetilde{r^*_\alpha}; r)$, *where* $x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_{k-1})$, e *is the added element,* α *is an index,* w^* *is the witness for position* α, $\widetilde{r^*_\alpha}$ *is a randomness, and* r *is the randomness of the transformation, output* $x' = (k+1, x'_1, \ldots, x'_n, \widetilde{c'_1}, \ldots, \widetilde{c'_n}, e_1, \ldots, e_{k-1}, e)$.
- AddT_w *is a transformation that on input* $(\mathsf{ck}, w, w^*, \alpha, \widetilde{r^*_\alpha}; r)$, *where* $w = (\phi, w_1, \ldots, w_{k-1}, \widetilde{r_1}, \ldots, \widetilde{r_n})$, *output* $w' = (\phi', w'_1, \ldots, w'_{k-1}, w^{*\prime}, \widetilde{r'_1}, \ldots, \widetilde{r'_n})$, *where* $\phi' = \phi \cup \{k+1 \to \alpha\}$[11].
- *Given* $\mathsf{ck}, x, e, w, w^*, \alpha, \widetilde{r^*_\alpha}, r$, *if* $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$, $x' \leftarrow \mathsf{AddT}_x(\mathsf{ck}, x, e, \alpha, w^*, \widetilde{r^*_\alpha}; r)$, *and* $w' \leftarrow \mathsf{AddT}_w(\mathsf{ck}, w, w^*, \alpha, \widetilde{r^*_\alpha}; r)$, $F(\mathsf{ck}, x'_\alpha, e, w'^*) = 1$, *then it holds that* $(\mathsf{ck}, x', w') \in R_{\mathcal{L}}$.
- ExtT_x *is a transformation that on input* $(\mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}; r)$ *output* $x' = (\mathsf{ck}, k, x'_1, \ldots, x'_n, x'_{n+1}, \widetilde{c'_1}, \ldots, \widetilde{c'_n}, \widetilde{c'_{n+1}}, e_1, \ldots, e_k)$ *where* $x = (\mathsf{ck}, k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_k)$.
- ExtT_w *is a transformation that on input* $(\mathsf{ck}, w, \widetilde{r_{n+1}}; r)$ *output* $w' = (\phi, w'_1, \ldots, w'_k, \widetilde{r'_1}, \ldots, \widetilde{r'_{n+1}})$ *where* $w = (\phi, w_1, \ldots, w_k, \widetilde{r_1}, \ldots, \widetilde{r_{n+1}})$.
- *Given* $\mathsf{ck}, x, x_{n+1}, \widetilde{c_{n+1}}, \widetilde{r_{n+1}}, r$, *if* $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$, $x' \leftarrow \mathsf{ExtT}_x(\mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}; r)$ *and* $w' \leftarrow \mathsf{ExtT}_w(\mathsf{ck}, w, \widetilde{r_{n+1}}; r)$, *then it holds that* $(\mathsf{ck}, x', w') \in R_{\mathcal{L}}$.

Definition 12 (Transformation Completeness). *An* EP *for* $R_{\mathcal{L}}$ *is transformation complete if* $\forall\lambda \in \mathbb{N}$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$, $(\mathsf{ck}, x, w) \in R_{\mathcal{L}}$, *where* $x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_k)$ *and* $w = (\phi, w_1, \ldots, w_k, \widetilde{r_1}, \ldots, \widetilde{r_n})$, *for all randomnesses* r, *and* (Π, AUX) *such that* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi) = 1$ *and* $\mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x, w, \mathsf{AUX}, \Pi) = 1$ *the following holds with probability 1:*

- *For all* $e, w^*, \alpha, \widetilde{r^*_\alpha}$ *s.t.* $\alpha \in [n]$, $\phi(i) \neq \alpha$ *for all* $i \in [k]$, *if* $(\Pi', x', \mathsf{aux}'_\alpha, r') \leftarrow \mathsf{PAdd}(\mathsf{crs}, \mathsf{ck}, x, w^*, \alpha, e, \widetilde{r^*_\alpha}, \mathsf{AUX}[\alpha], \Pi, r)$ *and* $F(\mathsf{ck}, x'_\alpha, e, w') = 1$ *then* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x', \Pi') = 1$ *and* $\mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x', w', \mathsf{AUX}', \Pi') = 1$. *The inputs to* $\mathsf{AuxVerify}$ *and* F *are obtained as follows:*
  - $x' \leftarrow \mathsf{AddT}_x(\mathsf{ck}, x, e, \alpha, w^*, \widetilde{r^*_\alpha}; r)$
  - $w' \leftarrow \mathsf{AddT}_w(\mathsf{ck}, w, w^*, \alpha, \widetilde{r^*_\alpha}; r)$
  - AUX' *is obtained as follows:* AUX'' *is obtained by replacing* aux_α *with* aux'_α *in* AUX, *and* $\mathsf{AUX}' \leftarrow \mathsf{AuxUpd}(\mathsf{crs}, \mathsf{ck}, x', \Pi', \mathsf{AUX}'', r')$
- *For all* x_{n+1}, $\widetilde{r_{n+1}}$, r, *if* $(\Pi', x', \mathsf{aux}_{n+1}, r') \leftarrow \mathsf{PExt}(\mathsf{crs}, \mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}, \Pi, r)$, *then* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x', \Pi') = 1$, *and* $\mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x', w', \mathsf{AUX}', \Pi') = 1$, *The inputs to* $\mathsf{AuxVerify}$ *are obtained as follows:*
  - $x' \leftarrow \mathsf{ExtT}_x(\mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}; r)$
  - $w' \leftarrow \mathsf{ExtT}_w(\mathsf{ck}, w, \widetilde{r_{n+1}}; r)$
  - AUX' *is obtained as follows:* AUX'' *is obtained by adding* aux_{n+1} *to* AUX, *and* $\mathsf{AUX}' \leftarrow \mathsf{AuxUpd}(\mathsf{crs}, \mathsf{ck}, x', \Pi', \mathsf{AUX}'', r')$.

Definition 13 (Soundness). *An* EP *for* $R_{\mathcal{L}}$ *is sound if for all* PPT $\mathcal{A}$ *and* $\forall\lambda \in \mathbb{N}$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$, *the probability that* $\mathcal{A}(\mathsf{crs}, \mathsf{ck})$ *outputs* (x, Π) *such that* $x \notin \mathcal{L}$ *but* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi) = 1$ *is negligible.*

We also require addition and extension privacy. That is, proofs updated with PAdd (or PExt) are indistinguishable from proofs done from scratch using Prv.

Definition 14 (Addition Privacy). *Consider the following experiment:*

- $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$
- $(x, w, e^*, w^*, \alpha, \widetilde{r_\alpha}, \Pi^*, \mathsf{AUX}^*, r) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{ck})$

[11] Notice that it must hold that for each $i \in [k]$, $\phi(i) \neq \alpha$.

– *If* $(\mathsf{ck}, x, w) \notin R_{\mathcal{L}}$ *or* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi^*) = 0$ *or* $\mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x, w, \mathsf{AUX}^*, \Pi^*) = 0$ *or* $\alpha \in Im(\phi)$ *(with* ϕ *taken from* w*) output* $\perp$ *and abort. Otherwise, sample* $b \leftarrow\!\$\ \{0,1\}$ *and do the following:*
  - *If* $b = 0$, $x' \leftarrow \mathsf{AddT}_x(\mathsf{ck}, x, e^*, \alpha, w^*, \widetilde{r^*_\alpha}, r)$, $w' \leftarrow \mathsf{AddT}_w(\mathsf{ck}, w, w^*, \alpha, \widetilde{r^*_\alpha}; r)$ $(\Pi, \mathsf{AUX}) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x', w')$
  - *If* $b = 1$, $(\Pi, x', \mathsf{aux}^*, r') \leftarrow \mathsf{PAdd}(\mathsf{crs}, \mathsf{ck}, x, w^*, \alpha, e^*, \widetilde{r^*_\alpha}, \mathsf{AUX}^*[\alpha], \Pi^*, r)$*. Replace in* AUX^* *the value* aux_α *with* aux^**.* $\mathsf{AUX} \leftarrow \mathsf{AuxUpd}(\mathsf{crs}, \mathsf{ck}, x, \Pi, \mathsf{AUX}^*, r')$
– $b' \leftarrow \mathcal{A}(x', \Pi, \mathsf{AUX})$.

For every PPT $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, $\Pr[b = b'] \leq 1/2 + \mathsf{negl}(\lambda)$.

Definition 15 (Extension Privacy). *Consider the following experiment:*

– $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$
– $(x, w, x_{n+1}, \widetilde{r_{n+1}}, \Pi^*, \mathsf{AUX}^*, r) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{ck})$
– *If* $(\mathsf{ck}, x, w) \notin R_{\mathcal{L}}$ *or* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi^*) = 0$ *or* $\mathsf{AuxVerify}(\mathsf{crs}, \mathsf{ck}, x, w, \mathsf{AUX}^*, \Pi^*) = 0$ *output* $\perp$ *and abort. Otherwise, sample* $b \leftarrow\!\$\ \{0,1\}$ *and do the following:*
  - *If* $b = 0$, $x' \leftarrow \mathsf{ExtT}_x(\mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}; r)$, $w' \leftarrow \mathsf{ExtT}_w(\mathsf{ck}, w, \widetilde{r_{n+1}}; r)$ $(\Pi, \mathsf{AUX}) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x', w')$
  - *If* $b = 1$ $(\Pi, x', \mathsf{aux}^*, r') \leftarrow \mathsf{PExt}(\mathsf{crs}, \mathsf{ck}, x, x_{n+1}, \widetilde{r_{n+1}}, \Pi^*, r)$*. Append the value* aux^* *to* AUX^**,* $\mathsf{AUX} \leftarrow \mathsf{AuxUpd}(\mathsf{crs}, \mathsf{ck}, x', \Pi, \mathsf{AUX}^*, r')$
– $b' \leftarrow \mathcal{A}(x', \Pi, \mathsf{AUX})$.

For every PPT $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, $\Pr[b = b'] \leq 1/2 + \mathsf{negl}(\lambda)$.

An EP may also satisfy (some of) the following additional properties.

Perfect Δ-Extraction is a perfect but mild extraction property. Indeed, instead of extracting the full witness, there exists an extractor which perfectly extracts the active indices and a function Δ of the rest of the witness.

Definition 16 (Perfect Δ-Extraction). *There exists a* PPT *extractor* Ext *that for all* $\lambda \in \mathbb{N}$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$*, and all* (x, Π) *s.t.* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi) = 1$ *it hold that:*

$$\Pr[(\mathsf{ck}, (x, \phi', w')) \in \mathcal{L}' \mid (\phi', w' = (w'_1, \ldots, w'_k))) \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}, \mathsf{ck}, x, \Pi)] = 1.$$

$$\begin{aligned} \textit{Where } \mathcal{L}' = \{(\mathsf{ck}, (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_k, \phi, w'))) | \exists w = (w_1, \ldots, w_k, \\ \mathsf{bit}_1, \ldots, \mathsf{bit}_n, \widetilde{r_1}, \ldots, \widetilde{r_n}) \textit{ s.t. } \forall i \in [n]\ \mathsf{bit}_i \in \{0,1\} \wedge\ \widetilde{c_i} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; \widetilde{r_i}) \wedge \\ \textstyle\sum_{i=1}^n \mathsf{bit}_i = k\ \wedge \forall_{j=1}^k (\mathsf{bit}_{\phi(j)} = 1 \wedge F(\mathsf{ck}, x_{\phi(j)}, e_j, w_j) = 1) \wedge w' = \Delta(w)\}. \end{aligned}$$

We now define three WI variants useful in security proofs. Such flavors have some similarities but do not seem to imply each other.

Witness addition indistinguishability (WAI) ensures that two addition operations over the same α, but with different witnesses for x_α are indistinguishable.

Definition 17 (WAI). *Consider the following experiment:*

– $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$
– $(x, \Pi, \alpha, \mathsf{aux}_\alpha, e, w_0, w_1, \widetilde{r'_\alpha}, r) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{ck})$
– *If* $F(\mathsf{ck}, x_\alpha, e, w_0) = 0$ *or* $F(\mathsf{ck}, x_\alpha, e, w_1) = 0$ *or* $\mathsf{PVfy}(\mathsf{crs}, \mathsf{ck}, x, \Pi) = 0$ *output* $\perp$ *and abort. Otherwise, sample* $b \leftarrow\!\$\ \{0,1\}$ *and do the following:*
– *Parse* $x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, e_1, \ldots, e_n)$ *and compute:*
  - $(\Pi', x', \mathsf{aux}'_\alpha, (r_1, \ldots, r_n)) \leftarrow \mathsf{PAdd}(\mathsf{crs}, \mathsf{ck}, x, w_b, \alpha, e, \widetilde{r'_\alpha}, \mathsf{aux}_\alpha, \Pi, r)$
– $b' \leftarrow \mathcal{A}(x', \Pi', \mathsf{aux}'_\alpha, (r_1, \ldots, r_n))$.

For every PPT $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, $\Pr[b = b'] \leq 1/2 + \mathsf{negl}(\lambda)$.

Fixed position witness indistinguishability (FPWI) guarantees that two proofs done for the same statement with two different witnesses sharing the same injective map ϕ are indistinguishable.

Definition 18 (FPWI). *Consider the following experiment:*

- $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$
- $(x, w_0, w_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{ck})$
- *Parse* $w_i = (\phi^i, w_1^i, \ldots, w_k^i, \widetilde{r_1^i}, \ldots, \widetilde{r_n^i})$ *for* $i \in \{0, 1\}$.
- *If* $(\mathsf{ck}, x, w_0) \notin R_\mathcal{L}$ *or* $(\mathsf{ck}, x, w_1) \notin R_\mathcal{L}$ *or* $\phi^0 \neq \phi^1$ *output* $\perp$ *and abort. Otherwise, sample* $b \leftarrow\!\$\ \{0, 1\}$ *and compute* $(\Pi, \mathsf{AUX}) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x, w_b)$.
- $b' \leftarrow \mathcal{A}(\Pi, \mathsf{AUX})$.

For every PPT $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, $\Pr[b = b'] \leq 1/2 + \mathsf{negl}(\lambda)$.

Extended witness indistinguishability (EWI) guarantees that two proofs for the same statement generated using different witnesses are indistinguishable. Importantly, this holds even if the adversary gets the auxiliary values corresponding to the statements that are *inactive* w.r.t. both witnesses. Additionally, the switch variables are committed by the experiment.

Definition 19 (EWI). *Consider the following experiment:*

- $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$
- $(x', w_0', w_1') \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{ck})$
- *Parse* $x' = (k, x_1, \ldots, x_n, e_1, \ldots, e_k)$
- *Parse* $w_b' = (\phi^b, w_1^b, \ldots, w_k^b)$ *with* $b \in \{0, 1\}$
- *If* $\exists b \in \{0, 1\}, j \in [k] : F(\mathsf{ck}, x_{\phi^b(j)}, e_j, w_j^b) = 0$ *output* $\perp$ *and abort.*
- *Sample* $b' \leftarrow\!\$\ \{0, 1\}$ *and do the following:*
  - *Construct* $x_{b'} = (k, x_1, \ldots, x_n, \widetilde{c_1^{b'}}, \ldots, \widetilde{c_k^{b'}}, e_1, \ldots, e_k)$, *and* $w_{b'} = (\phi^{b'}, w_1^{b'}, \ldots, w_k^{b'}, \widetilde{r_1^{b'}}, \ldots, \widetilde{r_n^{b'}})$, *where, for* $i \in [n]$, *for randomly sampled* $\widetilde{r_i^{b'}}$, *set* $\widetilde{c_i^{b'}} \leftarrow \mathsf{Com}_\mathsf{ck}(1; \widetilde{r_i^{b'}})$ *if* $i \in Im(\phi^{b'}(i))$, *and* $\widetilde{c_i^{b'}} \leftarrow \mathsf{Com}_\mathsf{ck}(0; \widetilde{r_i^{b'}})$
  - $(\Pi, (\mathsf{aux}_1, \ldots, \mathsf{aux}_n)) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x_{b'}, w_{b'})$
  - *Set* $S = ([n] \setminus (Im(\phi^0) \cup Im(\phi^1)))$, *and* $\mathsf{AUX} = \{\mathsf{aux}_i\}_{i \in S}$
- $b^* \leftarrow \mathcal{A}(x_{b'}, \Pi, \mathsf{AUX})$.

For every PPT $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, $\Pr[b^* = b'] \leq 1/2 + \mathsf{negl}(\lambda)$.

Extended zero knowledge (EZK) requires the existence of a simulator that, using a trapdoor, can simulate a proof and the auxiliary values of inactive statements.

Definition 20 (EZK). *Let* $\mathsf{O}EZK(b, \mathsf{crs}, \mathsf{ck}, x, w)$ *be defined as follows:*

- *Parse* w *as* $(\phi, w_1, \ldots, w_k, \widetilde{r_1}, \ldots, \widetilde{r_n})$
- *If* $(\mathsf{ck}, x, w) \notin R_\mathcal{L}$ *return* $\perp$
- *Else:*
  - *If* $b = 0$, $(\Pi, (\mathsf{aux}_1, \ldots, \mathsf{aux}_n)) \leftarrow \mathsf{Prv}(\mathsf{crs}, \mathsf{ck}, x, w)$
  - *If* $b = 1$, $(\Pi, (\mathsf{aux}_1, \ldots, \mathsf{aux}_n)) \leftarrow \mathsf{Sim}_1(\mathsf{crs}, \mathsf{ck}, \mathsf{td}, x)$
- *Set* $S = [n] \setminus Im(\phi)$ *and* $\mathsf{AUX} = \{\mathsf{aux}_i\}_{i \in S}$, *with* ϕ *taken from* w
- *Return* (Π, AUX).

There exists a polynomial time simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ *such that for every* PPT $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, *considering the following game:*

- $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(\mathsf{gk})$, $b \leftarrow\!\$\ \{0, 1\}$
- $b' \leftarrow \mathcal{A}^{\mathsf{O}EZK(b, \mathsf{crs}, \mathsf{ck}, \cdot, \cdot)}(\mathsf{crs}, \mathsf{ck})$

$\Pr[b = b'] \leq 1/2 + \mathsf{negl}(\lambda)$. *Moreover, the* crs *output by* Sim_0 *is computationally indistinguishable to the* crs *output by* CRSSetup.

5 Doubly-Authentication-Preventing Tags

A doubly-authentication-preventing tag (DAPT) consists of the following PPT algorithms and a poly-time relation R_{key}. Let $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$ be an implicit input to all algorithms. Given gk, a topic space $\mathcal{T}$, a message space $\mathcal{M}$, and a tag space $\mathcal{E}$ are implicitly defined.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{gk})$: outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$.
- $e \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m)$: it is a *deterministic* algorithm that on input the secret sk, topic $\tau \in \mathcal{T}$, and message $m \in \mathcal{M}$, outputs a tag $e \in \mathcal{E}$.
- $0/1 \leftarrow \mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk})$: it is a *deterministic* algorithm that on input two tags $e_0, e_1 \in \mathcal{E}$ on the same topic $\tau \in \mathcal{T}$ and different messages $m_0 \neq m_1 \in \mathcal{M}$, outputs 1 iff both tags were generated w.r.t. public key pk.

A DAPT has to satisfy all the following properties.

Definition 21 (Verifiability of Keys). *For all $\lambda \in \mathbb{N}$ and $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, it holds that $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{gk})$ iff $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}$. Additionally, we require that for all pk there exists a* unique *sk s.t. $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}$.*

Definition 22 (Tag Traceability). *For all $\lambda \in \mathbb{N}$, for any topic $\tau \in \mathcal{T}$, for any pair of messages $m_0 \neq m_1 \in \mathcal{M}$ it holds that, for any $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}$:*

$$\Pr\left[1 \leftarrow \mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}) \,\middle|\, \begin{matrix} e_0 \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m_0) \\ e_1 \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m_1) \end{matrix}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Additionally, for all PPT $\mathcal{A}$ and $\lambda \in \mathbb{N}$ let us consider the following game:

$$\begin{matrix} (\tau, m_0, m_1, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathcal{A}(\lambda) \\ e_0 \leftarrow \mathsf{Tag}(\mathsf{sk}_0, \tau, m_0) \\ e_1 \leftarrow \mathsf{Tag}(\mathsf{sk}_1, \tau, m_1) \end{matrix}$$

We say that $\mathcal{A}$ wins the above game if $\mathsf{sk}_0 \neq \mathsf{sk}_1 \wedge (\mathsf{pk}_0, \mathsf{sk}_0) \in R_{\mathsf{key}} \wedge (\mathsf{pk}_1, \mathsf{sk}_1) \in R_{\mathsf{key}} \wedge (1 = \mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}_0)) \vee (1 = \mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}_1))$. We require that the probability of $\mathcal{A}$ winning the above game be $\mathsf{negl}(\lambda)$.

Definition 23 (Pseudo-randomness). *For every PPT $\mathcal{A}$ and $\lambda \in \mathbb{N}$, the success probability in $\mathsf{Exp}^{\mathrm{TagPR}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$ (Fig. 9) is $\Pr\left[\mathsf{Exp}^{\mathrm{TagPR}}_{\mathcal{A},\mathsf{DAPT}}(\lambda) = win\right] \leq 1/2 + \mathsf{negl}(\lambda)$.*

Notice that, in Fig. 9, $\mathcal{A}$ can ask for honestly computed tags via the flag $\mathsf{honest} = 1$, independently of the value of b. The flag abrt is defined for convenience and can be ignored (i.e., $\mathcal{A}$ loses as soon as it makes an oracle call with $\mathsf{abrt} = 1$).

Definition 24 (Non-Frameability). *For every PPT $\mathcal{A}$ and $\lambda \in \mathbb{N}$, the success probability in $\mathsf{Exp}^{\mathrm{TagNF}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$ (Fig. 10) is $\Pr\left[\mathsf{Exp}^{\mathrm{TagNF}}_{\mathcal{A},\mathsf{DAPT}}(\lambda) = win\right] \leq \mathsf{negl}(\lambda)$.*

6 Our Tetris

Before giving our construction, we define some useful poly-time relations. We first define the disjunctive relation R_{D}. The statement contains a commitment c, a DAPT public key pk_{T}, a topic τ, a message m, and a trapdoor theorem x_{trap}. A witness for such relation may be a tuple $(e, r, \mathsf{sk}_{\mathsf{T}})$ such that (e, r) is an opening of c and e is a correctly computed tag over the pair (τ, m) and the secret key sk_{T}. Alternatively, the witness

$\mathsf{Exp}^{\mathrm{TagPR}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$

1 : $\mathsf{L_{tag}} \leftarrow \emptyset, b \leftarrow\$ \{0,1\}$
2 : $\mathsf{SKcorr} = 0, \mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$
3 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{gk})$
4 : $b' \leftarrow \mathcal{A}^{\mathsf{OTag}(\mathsf{b},\mathsf{sk},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk})$
5 : **if** $\mathsf{SKcorr} = 1$:
6 : **return** *lose*
7 : $\forall(\tau', m', \cdot), (\tau, m, \cdot) \in \mathsf{L_{tag}}$:
8 : **if** $\tau' = \tau \wedge m' \neq m$:
9 : **return** *lose*
10 : **if** $b' = b$:
11 : **return** *win*
12 : **else** : **return** *lose*

$\mathsf{OTag}(b, \mathsf{sk}, \tau, m, \mathsf{abrt}, \mathsf{honest})$

1 : **if** $\mathsf{abrt} = 1$:
2 : $\mathsf{SKcorr} = 1$
3 : **return** sk
4 : **if** $\exists(\tau', m', e) \in \mathsf{L_{tag}} \wedge \tau' = \tau \wedge m' = m$:
5 : **return** e
6 : **if** $\mathsf{honest} = 1$:
7 : $e \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m)$
8 : $\mathsf{L_{tag}} \leftarrow \mathsf{L_{tag}} \cup \{(\tau, m, e)\}$
9 : **return** e
10 : **if** $b = 0$: $e \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m)$ **else** : $e \leftarrow\$ \mathcal{E}$
11 : $\mathsf{L_{tag}} \leftarrow \mathsf{L_{tag}} \cup \{(\tau, m, e)\}$
12 : **return** e

Fig. 9. Tag pseudo-randomness game of DAPT.

can just be w_{trap} such that $(x_{\mathsf{trap}}, w_{\mathsf{trap}}) \in R_{\mathcal{L}_{\mathsf{trap}}}$, where $\mathcal{L}_{\mathsf{trap}}$ is a language in NP$\cap$ co-NP. The introduction of x_{trap} is instrumental to prove security of our $\mathfrak{Tetris}$.

$$\begin{aligned} R_{\mathsf{D}} =& \{(\mathsf{ck}, x = (c, \mathsf{pk_T}, \tau, m, x_{\mathsf{trap}}), w) | (w = (e, r, \mathsf{sk_T}) \wedge e \leftarrow \mathsf{Tag}(\mathsf{sk_T}, \tau, m) \wedge \\ & c \leftarrow \mathsf{Com}_{\mathsf{ck}}(e; r) \wedge (\mathsf{pk_T}, \mathsf{sk_T}) \in R_{\mathsf{key}}) \vee (w = w_{\mathsf{trap}} \wedge (x_{\mathsf{trap}}, w_{\mathsf{trap}}) \in R_{\mathcal{L}_{\mathsf{trap}}})\}. \end{aligned} \tag{2}$$

Then, we define the threshold relation over R_{D} as follows.

$$\begin{aligned} R_{\mathsf{Dtr}} =& \{(\mathsf{ck}, x = (k, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}), w = (\phi, w_1, \ldots, w_k, \widetilde{r_1}, \ldots, \widetilde{r_n})) | \\ & \forall i \in [n] \; \exists \; \mathsf{bit}_i : \; \widetilde{c_i} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; \widetilde{r_i}) \wedge \mathsf{bit}_i \in \{0,1\} \wedge \textstyle\sum_{i=1}^n \mathsf{bit}_i = k \\ & \wedge \forall_{i=1}^k (\mathsf{bit}_{\phi(i)} = 1 \wedge (\mathsf{ck}, x_{\phi(i)}, w_i) \in R_{\mathsf{D}}) \wedge \phi \text{ is an injective map } [t] \to [n]\} \end{aligned} \tag{3}$$

Additionally, we define the extendable shuffle relation as

$$\begin{aligned} R_{\mathsf{SH}} = \{&(\mathsf{ck}, x = (k, c_1, \ldots, c_n, \tilde{c_1}, \ldots, \tilde{c_n}, e_1, \ldots, e_k), w = (\phi, r_1, \ldots, r_k, \tilde{r_1}, \ldots, \tilde{r_n})) | \\ & \forall i \in [n] \; \exists \; \mathsf{bit}_i : \; \tilde{c_i} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; \tilde{r_i}) \wedge \mathsf{bit}_i \in \{0,1\} \wedge \textstyle\sum_{i=1}^n \mathsf{bit}_i = k \wedge \\ & \forall_{i=1}^k (\mathsf{bit}_{\phi(i)} = 1 \wedge (c_{\phi(i)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(e_i, r_i))) \wedge \phi \text{ is an injective map } [k] \to [n]\}. \end{aligned} \tag{4}$$

The components of our $\mathfrak{Tetris}$ are the following: (i) a doubly-authentication-preventing tag DAPT; (ii) an extendable non-interactive proof system for R_{Dtr} that we call EP; (iii) an extendable non-interactive proof system for R_{SH} that we call SH; (iv) an IND-CPA public key encryption scheme PKE in which every public key has a unique secret key. Additionally, PKE must be homomorphic w.r.t. EP.AuxUpd and SH.AuxUpd. Notice that the algorithm $\mathsf{PKE.Eval}_1$ ($\mathsf{PKE.Eval}_2$) homomorphically evaluates EP.AuxUpd (SH.AuxUpd). We require EP and SH to work over the same commitment scheme $\mathsf{Com}_{\mathsf{ck}}$.

Our $\mathfrak{Tetris}$ is shown in Fig. 11 and in Fig. 12. We refer to Sec. 1.2 for an intuitive description of our $\mathfrak{Tetris}$. For the sake of space, we omit the setup algorithm from the description of our $\mathfrak{Tetris}$. $\mathfrak{Tetris}$.Setup just runs the setup algorithm of all the used building blocks. Additionally, it samples a random instance x_{trap} such that $x_{\mathsf{trap}} \notin \mathcal{L}_{\mathsf{trap}}$[12]. All the public parameters are generally omitted from the input of the algorithms. In our construction, it is crucial that EP and SH share the same lists of commitments $C = \{c_i\}_{i\in[n]}$ and $\widetilde{C} = \{\widetilde{c_i}\}_{i\in[n]}$.

[12] This can be done efficiently by sampling an instance in the complement of $\mathcal{L}_{\mathsf{trap}}$.

$\mathsf{Exp}^{\mathrm{TagNF}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$

$\mathsf{L_{tag}} \leftarrow \emptyset,\ \mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{gk})$
$(e_0, e_1, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{OTag}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})$
if $m_0 = m_1$:
  return *lose*
$b' \leftarrow \mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk})$
$\forall(\tau', m'), (\tau, m) \in \mathsf{L_{tag}}$:
  if $\tau' = \tau \wedge m' \neq m$:
    return *lose*
if $b' = 1$: **return** *win* **else** : **return** *lose*

$\mathsf{OTag}(\mathsf{sk}, \tau, m)$

$e \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m)$
$\mathsf{L_{tag}} \leftarrow \mathsf{L_{tag}} \cup \{(\tau, m)\}$
return e

Fig. 10. Tag non-frameability game of DAPT.

Moreover, the addition and the extension operations on EP and SH must preserve such connection. Therefore, every time we perform an addition or an extension with EP, we re-use the same transformation randomness for SH. In this way, we end up with two updated proofs that are still talking about two updated statements sharing the same common part (i.e., C' and $\widetilde{C}'$). We use the square brackets notation to denote an element of a list, using 1 as the index of the first element. We treat sets as ordered lists, meaning that the elements of a set can be accessed, via the square bracket operator, in the same order they were added to the set. The relation $R^{\mathfrak{Tetris}}_{\mathsf{key}}$ is defined as $R^{\mathfrak{Tetris}}_{\mathsf{key}} = \{\mathsf{pk} = (\mathsf{pk_T}, \mathsf{pk_e}), \mathsf{sk} = (\mathsf{sk_T}, \mathsf{sk_e}) | (\mathsf{pk_T}, \mathsf{sk_T}) \in R_{\mathsf{key}} \wedge (\mathsf{pk_e}, \mathsf{sk_e}) \in R^{\mathsf{PKE}}_{\mathsf{key}}\}$, where R_{key} and $R^{\mathsf{PKE}}_{\mathsf{key}}$ are the poly-time relation verifying well-formed DAPT and PKE keys respectively.

Instantiating our $\mathfrak{Tetris}$. We use the ElGamal encryption in $\hat{\mathbb{G}}$ and $\check{\mathbb{H}}$ as a perfectly binding commitment $\mathsf{Com_{ck}}$, in Sec. 8 we build an extendable shuffle for such commitments, in App. C we present a EP for relation R_{Dtr} that is a simple adaptation of [8] and we set $\mathcal{L}_{\mathsf{trap}}$ to be the language of Diffie-Hellman tuples. We point out that the property of perfect Δ-extraction is not defined in [8], therefore we prove it in App. C, intuitively we set $\Delta((\hat{e}_j, r_{\hat{z}_j}, \mathsf{sk}^j_{\mathsf{T}})_{j\in[k]}) = (\hat{e}_1, \ldots, \hat{e}_k)$, and the property holds thanks to the perfect F-extraction of GS (see [28, Def. 3]). Finally, in Sec. 7 we propose a DAPT. All the components are instantiated from the same bilinear groups and thus work well with each other. Additionally, since in both EP and SH, updating the auxiliary values (AuxUpd) simply consists of applying the group operation between two elements of $\hat{\mathbb{G}}$ or $\check{\mathbb{H}}$, we can use ElGamal instantiated in $\hat{\mathbb{G}}$ and $\check{\mathbb{H}}$ as PKE. We point out that since the setup of our SH bounds the length of the commitments list, our $\mathfrak{Tetris}$ supports rings of bounded size. Getting a linear-size $\mathfrak{Tetris}$ without this bound is an open problem. We rely on the SXDH (Ass. 4), Co-CDH (Ass. 5), subset permutation pairing (Ass. 6), and subset simultaneous pairing (Ass. 7) assumptions, along with the random oracle model (ROM).

6.1 Security of Our $\mathfrak{Tetris}$

In this section, we prove the security of our $\mathfrak{Tetris}$ via the following theorem.

Theorem 2. *Let* DAPT *be defined as in Sec. 5. Let* EP *be defined as in Sec. 4 for relation* R_{Dtr} *with WAI (Def. 17), FPWI (Def. 18), EWI (Def. 19), perfect* Δ*-extraction (Def. 16) where* Δ *is defined as* $\Delta((\hat{e}_j, r_{\hat{z}_j}, \mathsf{sk}^j_{\mathsf{T}})_{j\in[k]}) = (\hat{e}_1, \ldots, \hat{e}_k)$*. Let* SH *be defined as in Sec. 4 for relation* R_{SH} *with EZK (Def. 20). Let* PKE *be an* IND-CPA *public key encryption scheme which is homomorphic w.r.t.* EP.AuxUpd *and* SH.AuxUpd*. Then, the scheme in Fig. 11,12 is a* $\mathfrak{Tetris}$*.*

We prove Thm. 2 via the following lemmas. Verifiability of keys (Def. 2) follows from the verifiability of keys of underlying schemes (Def. 21 and Def. 27).

$\mathsf{Sign}(\tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \mathsf{sk})$

$\mathsf{A}_{\mathsf{EP}} \leftarrow \emptyset, \mathsf{A}_{\mathsf{SH}} \leftarrow \emptyset, \mathsf{E} \leftarrow \emptyset$
parse $\{\mathsf{pk}_i\}_{i\in\mathcal{R}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
$\forall\, i \in [n]$: **parse** $\mathsf{pk}_i = (\mathsf{pk}_{\mathsf{e}}^i, \mathsf{pk}_{\mathsf{T}}^i)$
parse $\mathsf{sk} = (\mathsf{sk}_{\mathsf{e}}, \mathsf{sk}_{\mathsf{T}})$
if $\nexists\, \mathsf{pk}_{\mathsf{T}}^j, j \in [n]$ s.t. $(\mathsf{pk}_{\mathsf{T}}^j, \mathsf{sk}_{\mathsf{T}}) \in R_{\mathsf{key}}$:
  return $\perp$
$\mathsf{E} \leftarrow \mathsf{E} \cup (e \leftarrow \mathsf{Tag}(\mathsf{sk}_{\mathsf{T}}, \tau, m))$
$r_j, \widetilde{r_j} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c_j \leftarrow \mathsf{Com}_{\mathsf{ck}}(e; r_j), \widetilde{c_j} \leftarrow \mathsf{Com}_{\mathsf{ck}}(1; \widetilde{r_j})$
for $i \neq j \wedge i \in [n]$:
  $r_i, \widetilde{r_i} \leftarrow\!\!\$\ \{0,1\}^\lambda$
  $c_i \leftarrow \mathsf{Com}_{\mathsf{ck}}(0; r_i), \widetilde{c_i} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0; \widetilde{r_i})$
$\forall i \in [n] : x_i \leftarrow (c_i, \mathsf{pk}_{\mathsf{T}}^i, \tau, m, x_{\mathsf{trap}})$
$w' \leftarrow (e, r_j, \mathsf{sk}_{\mathsf{T}})$
$C \leftarrow (c_i)_{i\in[n]}$, $\widetilde{C} \leftarrow (\widetilde{c_i})_{i\in[n]}$
$R \leftarrow (r_i)_{i\in[n]}$, $\widetilde{R} \leftarrow (\widetilde{r_i})_{i\in[n]}$
$x_{\mathsf{EP}} \leftarrow (1, x_1, \ldots, x_n, \widetilde{C})$
$x_{\mathsf{SH}} \leftarrow (1, C, \widetilde{C}, \mathsf{E})$
$\phi(1) \leftarrow j$
$(\Pi_1, \mathsf{AUX}_{\mathsf{EP}}) \leftarrow \mathsf{EP.Prv}(x_{\mathsf{EP}}, (\phi, w', \widetilde{R}))$
$(\Pi_2, \mathsf{AUX}_{\mathsf{SH}}) \leftarrow$
$\mathsf{SH.Prv}(x_{\mathsf{SH}}, (\phi, R, \widetilde{R}))$
for $i \in [n]$:
  if $i = j$:
    $\mathsf{A}_{\mathsf{EP}}[i] \leftarrow \mathsf{PKE.Enc}(\perp, \mathsf{pk}_{\mathsf{e}}^i)$
    $\mathsf{A}_{\mathsf{SH}}[i] \leftarrow \mathsf{PKE.Enc}(\perp, \mathsf{pk}_{\mathsf{e}}^i)$
  else :
    $\mathsf{A}_{\mathsf{EP}}[i] \leftarrow \mathsf{PKE.Enc}(\mathsf{AUX}_{\mathsf{EP}}[i], \mathsf{pk}_{\mathsf{e}}^i)$
    $\mathsf{A}_{\mathsf{SH}}[i] \leftarrow \mathsf{PKE.Enc}(\mathsf{AUX}_{\mathsf{SH}}[i], \mathsf{pk}_{\mathsf{e}}^i)$
$\sigma \leftarrow (1, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C, \widetilde{C}, \mathsf{E})$
return σ

$\mathsf{KeyGen}()$

$(\mathsf{pk}_{\mathsf{e}}, \mathsf{sk}_{\mathsf{e}}) \leftarrow \mathsf{PKE.KeyGen}()$
$(\mathsf{pk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}}) \leftarrow \mathsf{DAPT.KeyGen}()$
$(\mathsf{pk} \leftarrow (\mathsf{pk}_{\mathsf{e}}, \mathsf{pk}_{\mathsf{T}}), \mathsf{sk} \leftarrow (\mathsf{sk}_{\mathsf{e}}, \mathsf{sk}_{\mathsf{T}}))$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Join}(\tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \mathsf{sk}, \sigma)$

parse $\{\mathsf{pk}_i\}_{i\in\mathcal{R}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
$\forall\, i \in [n]$: **parse** $\mathsf{pk}_i = (\mathsf{pk}_{\mathsf{e}}^i, \mathsf{pk}_{\mathsf{T}}^i)$
parse $\mathsf{sk} = (\mathsf{sk}_{\mathsf{e}}, \mathsf{sk}_{\mathsf{T}})$
if $\nexists\, \mathsf{pk}_{\mathsf{T}}^j, j \in [n]$ s.t. $(\mathsf{pk}_{\mathsf{T}}^j, \mathsf{sk}_{\mathsf{T}}) \in R_{\mathsf{key}}$:
  return $\perp$
parse $\sigma = (k, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C, \widetilde{C}, \mathsf{E})$
$\mathsf{E} \leftarrow \mathsf{E} \cup (e \leftarrow \mathsf{Tag}(\mathsf{sk}_{\mathsf{T}}, \tau, m))$
$r_j, \widetilde{r_j} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c_j \leftarrow \mathsf{Com}_{\mathsf{ck}}(e; r_j), \widetilde{c_j} \leftarrow \mathsf{Com}_{\mathsf{ck}}(1; \widetilde{r_j})$
$\mathsf{aux}_{\mathsf{EP}} \leftarrow \mathsf{PKE.Dec}(\mathsf{A}_{\mathsf{EP}}[j], \mathsf{sk}_{\mathsf{e}})$
$\mathsf{aux}_{\mathsf{SH}} \leftarrow \mathsf{PKE.Dec}(\mathsf{A}_{\mathsf{SH}}[j], \mathsf{sk}_{\mathsf{e}})$
$x_i \leftarrow (C[i], \mathsf{pk}_{\mathsf{T}}^i, \tau, m, x_{\mathsf{trap}})\ \forall i \in [n]$
$w' \leftarrow (e, r_j, \mathsf{sk}_{\mathsf{T}}^j)$
$x_{\mathsf{EP}} \leftarrow (k, x_1, \ldots, x_n, \widetilde{C})$
$x_{\mathsf{SH}} \leftarrow (k, C, \widetilde{C}, \mathsf{E})$
Sample transformation randomness R_{Add}
$(\Pi'_{\mathsf{EP}}, x'_{\mathsf{EP}}, \mathsf{aux}'_j, R_{\mathsf{EP}}) \leftarrow$
$\mathsf{EP.PAdd}(x_{\mathsf{EP}}, w, j, \cdot, \widetilde{r_j}, \mathsf{aux}_{\mathsf{EP}}, \Pi_{\mathsf{EP}}, R_{\mathsf{Add}})$
$(\Pi'_{\mathsf{SH}}, (k+1, C', \widetilde{C}', \mathsf{E}), \mathsf{aux}'_j, R_{\mathsf{SH}}) \leftarrow$
$\mathsf{SH.PAdd}(x_{\mathsf{SH}}, r_j, j, e, \widetilde{r_j}, \mathsf{aux}_{\mathsf{SH}}, \Pi_{\mathsf{SH}}, R_{\mathsf{Add}})$
$\mathsf{A}_{\mathsf{EP}}[j] \leftarrow \mathsf{PKE.Enc}(\perp, \mathsf{pk}_{\mathsf{e}}^j)$
$\mathsf{A}_{\mathsf{SH}}[j] \leftarrow \mathsf{PKE.Enc}(\perp, \mathsf{pk}_{\mathsf{e}}^j)$
for $i \in [n]$:
  $\mathsf{A}_{\mathsf{EP}}[i] \leftarrow \mathsf{PKE.Eval}_1(\Pi_{\mathsf{EP}}, \mathsf{A}_{\mathsf{EP}}[i], R_{\mathsf{EP}}, \mathsf{pk}_{\mathsf{e}}^i)$
  $\mathsf{A}_{\mathsf{SH}}[i] \leftarrow \mathsf{PKE.Eval}_2(\Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{SH}}[i], R_{\mathsf{SH}}, \mathsf{pk}_{\mathsf{e}}^i)$
$\sigma \leftarrow (k+1, \Pi'_{\mathsf{EP}}, \Pi'_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C', \widetilde{C}', \mathsf{E})$
return σ

Fig. 11. KeyGen, Sign, and Join algorithms of our $\mathfrak{Tetris}$.

$\mathsf{Extend}(\tau, m, \sigma, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \mathsf{pk}^*)$

1 : **if** $\mathsf{pk}^* \in \{\mathsf{pk}_i\}_{i\in\mathcal{R}}$: **return** $\perp$
2 : **parse** $\{\mathsf{pk}_i\}_{i\in\mathcal{R}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
3 : $\forall\, i \in [n]$: **parse** $\mathsf{pk}_i = (\mathsf{pk}_\mathsf{e}^i, \mathsf{pk}_\mathsf{T}^i)$
4 : **parse** $\mathsf{pk}^* = (\mathsf{pk}_\mathsf{e}^{n+1}, \mathsf{pk}_\mathsf{T}^{n+1})$
5 : **parse** $\sigma = (k, \Pi_\mathsf{EP}, \Pi_\mathsf{SH}, \mathsf{A}_\mathsf{EP}, \mathsf{A}_\mathsf{SH}, C, \widetilde{C}, \mathsf{E})$
6 : $\forall i \in [n] : x_i \leftarrow (C[i], \mathsf{pk}_\mathsf{T}^i, \tau, m, x_\mathsf{trap})$
7 : $x_\mathsf{EP} \leftarrow (k, x_1, \ldots, x_n, \widetilde{C})$
8 : $r_{n+1}, \widetilde{r_{n+1}} \leftarrow\!\!\$\ \{0,1\}^\lambda$
9 : $c_{n+1} \leftarrow \mathsf{Com}_\mathsf{ck}(0; r_{n+1})$
10 : $\widetilde{c_{n+1}} \leftarrow \mathsf{Com}_\mathsf{ck}(0; \widetilde{r_{n+1}})$
11 : $x_\mathsf{SH} \leftarrow (k, C, \widetilde{C}, \mathsf{E})$
12 : Sample transformation randomness R_Ext
13 : $(\Pi'_\mathsf{EP}, x'_\mathsf{EP}, \mathsf{aux}_\mathsf{EP}, R_\mathsf{EP}) \leftarrow$
$\mathsf{EP.PExt}(x_\mathsf{EP}, x_{n+1}, \widetilde{r_{n+1}}, \Pi_\mathsf{EP}, R_\mathsf{Ext})$
14 : $(\Pi'_\mathsf{SH}, (k, C', \widetilde{C}', \mathsf{E}), \mathsf{aux}_\mathsf{SH}, R_2) \leftarrow$
$\mathsf{SH.PExt}(x_\mathsf{SH}, c_{n+1}, \widetilde{r_{n+1}}, \Pi_\mathsf{SH}, R_\mathsf{Ext})$
15 : $\mathsf{A}_\mathsf{EP}[n+1] \leftarrow \mathsf{PKE.Enc}(\mathsf{aux}_\mathsf{EP}, \mathsf{pk}_\mathsf{e}^{n+1})$
16 : $\mathsf{A}_\mathsf{SH}[n+1] \leftarrow \mathsf{PKE.Enc}(\mathsf{aux}_\mathsf{SH}, \mathsf{pk}_\mathsf{e}^{n+1})$
17 : **for** $i \in [n+1]$:
18 : $\quad \mathsf{A}_\mathsf{EP}[i] \leftarrow \mathsf{PKE.Eval}_1(\Pi_1, \mathsf{A}_\mathsf{EP}[i], R_1, \mathsf{pk}_\mathsf{e}^i)$
19 : $\quad \mathsf{A}_\mathsf{SH}[i] \leftarrow \mathsf{PKE.Eval}_2(\Pi_2, \mathsf{A}_\mathsf{SH}[i], R_2, \mathsf{pk}_\mathsf{e}^i)$
20 : $\sigma \leftarrow (k, \Pi'_\mathsf{EP}, \Pi'_\mathsf{SH}, \mathsf{A}_\mathsf{EP}, \mathsf{A}_\mathsf{SH}, C', \widetilde{C}', \mathsf{E})$
21 : **return** σ

$\mathsf{Verify}(t, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma)$

1 : **parse** $\{\mathsf{pk}_i\}_{i\in\mathcal{R}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
2 : $\forall\, i \in [n]$: **parse** $\mathsf{pk}_i = (\mathsf{pk}_\mathsf{e}^i, \mathsf{pk}_\mathsf{T}^i)$
3 : **parse** $\sigma = (k, \Pi_\mathsf{EP}, \Pi_\mathsf{SH}, \mathsf{A}_\mathsf{EP}, \mathsf{A}_\mathsf{SH}, C, \widetilde{C}, \mathsf{E})$
4 : **if** $k < t$:
5 : $\quad$ **return** 0
6 : **else** :
7 : $\quad \forall\, i \in [n]$: $x_i \leftarrow (C[i], \mathsf{pk}_\mathsf{T}^i, \tau, m, x_\mathsf{trap})$
8 : $\quad x_\mathsf{EP} \leftarrow (k, x_1, \ldots, x_n, \widetilde{C})$
9 : $\quad x_\mathsf{SH} \leftarrow (k, C, \widetilde{C}, \mathsf{E})$
10 : $\quad$ **return** $(\mathsf{EP.PVfy}(x_\mathsf{EP}, \Pi_\mathsf{EP}) \wedge$
$\mathsf{SH.PVfy}(x_\mathsf{SH}, \Pi_\mathsf{SH}))$

$\mathsf{Trace}(\tau, m_1, m_2, \sigma_1, \sigma_2, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_1}, \{\mathsf{pk}_i\}_{i\in\mathcal{R}_2})$

1 : $\mathsf{PK}_1 \leftarrow \{\mathsf{pk}_i\}_{i\in\mathcal{R}_1} \cap \{\mathsf{pk}_i\}_{i\in\mathcal{R}_2}$
2 : $\mathsf{PK}_2 \leftarrow \emptyset, n = 0$
3 : **parse** $\sigma_1 = (k_1, \Pi^1_\mathsf{EP}, \Pi^1_\mathsf{SH}, \mathsf{A}^1_\mathsf{EP}, \mathsf{A}^1_\mathsf{SH}, C_1, \widetilde{C}_1, \mathsf{E}_1)$
4 : **parse** $\sigma_2 = (k_2, \Pi^2_\mathsf{EP}, \Pi^2_\mathsf{SH}, \mathsf{A}^2_\mathsf{EP}, \mathsf{A}^2_\mathsf{SH}, C_2, \widetilde{C}_2, \mathsf{E}_2)$
5 : $\forall \mathsf{pk} \in \mathsf{PK}_1, e_1 \in \mathsf{E}_1, e_2 \in \mathsf{E}_2$:
6 : $\quad$ **parse** $\mathsf{pk} = (\mathsf{pk}_\mathsf{e}, \mathsf{pk}_\mathsf{T})$
7 : $\quad$ **if** $m_1 = m_2$:
8 : $\quad\quad$ **if** $e_1 = e_2$:
9 : $\quad\quad\quad n \leftarrow n + 1$
10 : $\quad$ **else if** $\mathsf{TagTrace}(e_1, e_2, m_1, m_2, \mathsf{pk}_\mathsf{T}) = 1$:
11 : $\quad\quad \mathsf{PK}_2 \leftarrow \mathsf{PK}_2 \cup \mathsf{pk}$
12 : **if** $|\mathsf{PK}_2| > 0$:
13 : $\quad$ **return** PK_2
14 : **if** $n > 0$:
15 : $\quad$ **return** (linked, n)
16 : **return** indep

Fig. 12. Extend, Verify, and Trace algorithms of our $\mathfrak{Tetris}$.

Lemma 3. *Let* EP *be defined as in Sec. 4 for relation* R_{Dtr}*, and let* SH *be defined as in Sec. 4 for relation* R_{SH}*. Then the scheme in Fig. 11,12 is correct (Def. 1).*

Proof. Let us assume that lad is not well-formed, then Proc (Fig. 3) would return $\perp$ as required by the definition. On the other end, if the ladder is well-formed, Proc will return a triple $(\Sigma, \mathsf{Th}, \mathcal{R})$, where $\Sigma = \{\sigma_1, \ldots, \sigma_\ell\}$, $\mathsf{Th} = \{t_1, \ldots, t_\ell\}$. Recall that for each element in $t_i \in \mathsf{Th}$ with $i \in [\ell]$, we have that $t_1 = 1$ and $t_i = t_{i-1}$ if $\mathsf{action}_i = \mathsf{Extend}$, and $t_i = t_{i-1} + 1$ if $\mathsf{action}_i = \mathsf{Join}$. We now argue that $\bigwedge_{j=1}^{\ell} \mathsf{Verify}(t_j, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma_j) = 1$ (Fig. 12). Let us now consider all the possible actions contained in the ladder:

1. action_1 of lad is always a Sign action, which produces a signature of the format $\sigma_1 \leftarrow (k = 1, \Pi'_{\mathsf{EP}}, \Pi'_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C', \widetilde{C}', \mathsf{E})$. Therefore, the check of line 4 (Fig. 12) of $\mathsf{Verify}(t_1, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma_1)$ will not return 0.
2. If action_i of lad is an Extend action, it produces a signature of the format $\sigma_i \leftarrow (k = t_{i-1}, \Pi'_{\mathsf{EP}}, \Pi'_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C', \widetilde{C}', \mathsf{E})$. Therefore, the check of line 4 (Fig. 12) of $\mathsf{Verify}(t_i, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma_1)$ will not return 0, since $t_i = t_{i-1}$.
3. If action_i of lad is a Join action, it produces a signature of the format $\sigma_i \leftarrow (k = t_{i-1} + 1, \Pi'_{\mathsf{EP}}, \Pi'_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C', \widetilde{C}', \mathsf{E})$. Therefore, the check of line 4 (Fig. 12) of $\mathsf{Verify}(t_i, \tau, m, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma_1)$ will not return 0, since $t_i = t_{i-1} + 1$.

It remains to notice that in all of the above cases, the proof generation (in Proc) and verification (in Verify) algorithms of EP and SH are used with the intended inputs. Therefore, the completeness of EP and SH guarantees that the check of line 10 of Verify (Fig. 12) returns 1.

Lemma 4. *Let* DAPT *be defined as in Sec. 5. Then, the scheme in Fig. 11,12 has tracing correctness (Def. 3).*

Proof. We have to consider the following cases:

1. if the set of overlapping signers is empty (i.e., $\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S} = \emptyset$), then Trace must return indep with overwhelming probability;
2. if $m_0 \neq m_1$, then Trace must return the set of overlapping signers $\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S}$ with overwhelming probability;
3. if $m_0 = m_1$, then Trace must return (linked, n) where n is the number of overlapping signers (i.e., $n = |\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S}|$) with overwhelming probability.

We now argue that the above requirements hold:

1 : Let us assume that there exists a PPT adversary $\mathcal{A}$ that returns $(\tau, m_0, m_1, \mathsf{lad}_0, \mathsf{lad}_1, \mathsf{L}_{\mathsf{keys}})$ such that $\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S} = \emptyset$, but Trace does not return indep. Trace returns a value different from indep if and only if $\mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}_{\mathsf{T}}) = 1$ for some pk_{T} (line 10 of Trace Fig. 12). Let pk_{T}, e_0, e_1, m_0, m_1 be the values in the loop at line 5 of Trace (Fig. 12) in the algorithm Trace for which $\mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}_{\mathsf{T}}) = 1$. Notice that, due to the check at line 6, of $\mathsf{Exp}^{\mathrm{Tcorr}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 5) if $\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S} = \emptyset$ then there is only one signer that has public key containing pk_{T}. Since all pk_{T} are different, we look for all $\mathsf{sk}^i_{\mathsf{T}}, \mathsf{sk}^j_{\mathsf{T}} \in \mathsf{L}_{\mathsf{keys}}$ such that $e_0 \leftarrow \mathsf{Tag}(\mathsf{sk}^i_{\mathsf{T}}, \tau, m_0)$ and $e_1 \leftarrow \mathsf{Tag}(\mathsf{sk}^j_{\mathsf{T}}, \tau, m_1)$. We use $\mathcal{A}$ to define an adversary $\mathcal{B}$ that breaks tag traceability of DAPT with the same probability of $\mathcal{A}$ winning in the tracing correctness game. $\mathcal{B}$ runs $\mathcal{A}$ and outputs $(\tau, m_0, m_1, \mathsf{pk}^i_{\mathsf{T}}, \mathsf{pk}^j_{\mathsf{T}}, \mathsf{sk}^i_{\mathsf{T}}, \mathsf{sk}^j_{\mathsf{T}})$, where $\mathsf{pk}^i_{\mathsf{T}}, \mathsf{pk}^j_{\mathsf{T}}$ are the public keys associated with $\mathsf{sk}^i_{\mathsf{T}}, \mathsf{sk}^j_{\mathsf{T}}$. Notice that $\mathcal{B}$ wins the tag traceability game, with the same advantage of $\mathcal{A}$, since the secret and public keys used by $\mathcal{A}$ belong to $R^{\mathfrak{Tetris}}_{\mathsf{key}}$, otherwise the check at line 4 of $\mathsf{Exp}^{\mathrm{Tcorr}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ (Fig. 5) fails.

2 : Let us assume that there exists a PPT adversary $\mathcal{A}$ that returns, with non-negligible probability, $(\tau, m_0, m_1, \mathsf{lad}_0, \mathsf{lad}_1, \mathsf{L}_{\mathsf{keys}})$ such that $m_0 \neq m_1$ and $\mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S} \neq \emptyset$, but Trace returns $\mathsf{PK} \neq \mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S}$. Let tinfo_b, for $b \in \{0, 1\}$, be the tracing information produced by $\mathsf{Proc}(\tau, m_b, \mathsf{L}_{\mathsf{keys}}, \mathsf{lad}_b)$. For each public key $\mathsf{pk}_{\mathsf{T}} \in \mathsf{lad}_0.\mathcal{S} \cap \mathsf{lad}_1.\mathcal{S}$ such that $\mathsf{pk}_{\mathsf{T}} \notin \mathsf{PK}$, let $e^b \leftarrow \mathsf{Tag}(\mathsf{sk}_{\mathsf{T}}, \tau, m_b)$, where

sk_T is the secret key associated with pk_T taken from $\mathsf{L}_{\mathsf{keys}}$, it must be that $e^b \in \mathsf{tinfo}_b$. Therefore, $\mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}_\mathsf{T}) = 0$ only with negligible probability due to tag traceability (Def. 22), implying that the check of line 10 of Trace (Fig. 12) can fail on input $(e_0, e_1, m_0, m_1, \mathsf{pk}_\mathsf{T})$ only with negligible probability. Thus, $\mathcal{A}$ can win only with the same probability, reaching a contradiction.

3 : The analysis is identical to the previous case.

Lemma 5. *Let* DAPT *be defined as in Sec. 5. Let* EP *be defined as in Sec. 4 for relation* R_{Dtr} *with perfect* Δ*-extraction (Def. 16) where* Δ *is defined as* $\Delta((\hat{e}_j, r_{\hat{z}_j}, \mathsf{sk}^j_\mathsf{T})_{j\in[k]}) = (\hat{e}_1, \ldots, \hat{e}_k)$*. Let* SH *be defined as in Sec. 4 for relation* R_{SH}*. Then, the scheme in Fig. 11,12 has signers extraction (Def. 4).*

Proof. Let Δ be defined as follows:

$$(e_1, \ldots, e_k) = \Delta(\phi, (e_j, r_j, \mathsf{sk}^j_\mathsf{T})_{j\in[k]}, (\widetilde{r}_i)_{j\in[n]}) \tag{5}$$

Let $(\mathsf{pp} = (x_{\mathsf{trap}}, \mathsf{crs}_{\mathsf{SH}}, \mathsf{crs}_{\mathsf{EP}}), \mathsf{td} = (\mathsf{td}_{\mathsf{SH}}, \mathsf{td}_{\mathsf{EP}}))$ be the public parameters and trapdoor of our $\mathfrak{Tetris}$. The extractor $\mathsf{Ext}(\mathsf{td}, \tau, m, t, \{\mathsf{pk}_i\}_{i\in\mathcal{R}}, \sigma)$ is defined as follows:

- Parse $\sigma = (\mu = (t, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C, \widetilde{C}), \mathsf{tinfo} = \mathsf{E})$.
- Let $x_{\mathsf{EP}} = (t, x_1, \ldots, x_n, \widetilde{C})$ where $x_i = (C[i], \mathsf{pk}^i_\mathsf{T}, \tau, m, x_{\mathsf{trap}})$ be the statement of EP.
- $(\phi, \mathsf{E}' = (e'_1, \ldots, e'_k)) \leftarrow \mathsf{EP.Ext}(\mathsf{crs}_{\mathsf{EP}}, \mathsf{td}_{\mathsf{EP}}, \mathsf{ck}, \Pi_{\mathsf{EP}}, x)$ (executed with function Δ).
- If E' is a permutation of E do the following:
  - Initialize $\mathsf{PK}, \mathsf{Ind} \leftarrow \emptyset$; for all $i \in [t]$ $\mathsf{Ind} \leftarrow \mathsf{Ind} \cup \phi(i)$ and $\mathsf{PK} \leftarrow \mathsf{PK} \cup \mathsf{pk}_{\phi(i)}$.
  - Return $(\mathsf{PK}, \mathsf{Ind})$.
- Else return $\perp$.

First, let us analyse the case in which Ext returns $(\mathsf{PK}, \mathsf{Ind})$. According to relations R_{D} (see (2)) and R_{Dtr} (see (3)) Π_{EP} proves that:

1. for all $i \in [n]$, $\widetilde{C}[i]$ is either a commitment to 1 (if i is a signer), or to 0 (if i is not a signer);
2. the sum of the values committed in $\widetilde{C}$ is t;
3. if i is a signer, $C[i]$ is a commitment of a tag e with $e \leftarrow \mathsf{Tag}(\mathsf{sk}^i_\mathsf{T}, \tau, m)$, where sk^i_T is the DAPT secret key of the signer i.

This follows from the fact that $x_{\mathsf{trap}} \notin \mathcal{L}_{\mathsf{trap}}$, thus the disjunctive relation (2) can only be satisfied by correctly generated tags. Recall that EP has perfect Δ-extraction and $\mathsf{Com}_{\mathsf{ck}}$ is perfectly binding. Therefore, the range of the extracted ϕ is uniquely determined and corresponds to the indices of the active signers. Observe that each element in E is produced by the *deterministic* algorithm Tag and every DAPT public key has only one corresponding secret key, therefore each element e in E (i.e., tinfo) is uniquely determined by $(\mathsf{pk}, \phi, \tau, m)$ for some $\mathsf{pk} \in \mathsf{PK}$. We can therefore conclude that every accepting signature $\sigma = (\mu, \mathsf{tinfo})$ has a *unique* tinfo determined by t, τ, m, and the set of signers. Thus, there must exist a ladder lad producing a final signature $\sigma' = (\mu', \mathsf{tinfo}')$ with $\mathsf{tinfo}' = \mathsf{E} = \mathsf{tinfo}$. This ladder contains sign and join operations which are done using, in an appropriate order, the signers with indices in Ind.

Let us now consider the case in which Ext returns $\perp$ (i.e., Π_{EP} and Π_{SH} verifies, but E' is not a permutation of E that happens if the elements in the SH statements are not the one committed in EP). Assume that there exists a PPT $\mathcal{A}$ producing an accepting signature that makes Ext return $\perp$ with non-negligible probability. According to relation R_{SH} (see relation (4)), Π_{SH} proves that the following tuple $x_{\mathsf{SH}} = (k, C, \widetilde{C}, \mathsf{E})$ has the following properties:

1. for all $i \in [n]$, $\widetilde{C}[i]$ is either a commitment to 1 (if i is a signer), or to 0 (if i is not a signer);
2. the sum of the values committed in $\widetilde{C}$ is t;
3. $\phi : [t] \to [n]$ is injective;
4. for all $i \in [t]$, $C[\phi(i)]$ is a commitment to $\mathsf{E}[i]$.

Observe that Ext returns $\perp$ if and only if the point 4 above is not satisfied. However, this means that $x_{\mathsf{SH}} \notin \mathcal{L}_{\mathsf{SH}}$ but Π_{SH} is accepting, contradicting the (computational) soundness of SH.

Lemma 6. *Let* DAPT *be defined as in Sec. 5. Let* EP *be defined as in Sec. 4 for relation* R_{Dtr} *with WAI (Def. 17), FPWI (Def. 18), EWI (Def. 19). Let* SH *be defined as in Sec. 4 for relation* R_{SH} *with EZK (Def. 20). Let* PKE *be an* IND-CPA *scheme which is homomorphic w.r.t.* $\mathsf{EP.AuxUpd}$ *and* $\mathsf{SH.AuxUpd}$*. Then the scheme in Fig. 11,12 is anonymous (Def. 6).*

Proof. Let $\mathcal{B}$ be the reduction that internally runs $\mathcal{A}$. Every $\mathcal{H}_i$ is identical to $\mathcal{H}_{i-1}$ except the changes that are explicitly reported in the description of $\mathcal{H}_i$.

$\mathcal{H}_0$: This is exactly the anonymity game of Fig. 6 with $b = 0$.

$\mathcal{H}_1$: Instead of sampling x_{trap} such that $x_{\mathsf{trap}} \notin \mathcal{L}_{\mathsf{trap}}$, $\mathcal{B}$ samples $(x_{\mathsf{trap}}, w_{\mathsf{trap}}) \in R_{\mathcal{L}_{\mathsf{trap}}}$. This hybrid is computationally indistinguishable from the previous one thanks to the hardness of $\mathcal{L}_{\mathsf{trap}}$.

$\mathcal{H}_2$: $\mathcal{B}$ switches the common reference string $\mathsf{crs}_{\mathsf{SH}}$ to $(\mathsf{crs}_{\mathsf{SH}}, \mathsf{td}_{\mathsf{crs}_{\mathsf{SH}}}) \leftarrow \mathsf{SH.Sim}_0(\mathsf{gk})$. This hybrid is computationally indistinguishable from the previous one because of the computational indistinguishability of honest and simulated $\mathsf{crs}_{\mathsf{SH}}$ imposed by EZK (Def. 20) of SH. $\mathcal{B}$ includes the $\mathsf{crs}_{\mathsf{SH}}$ to be distinguished in the public parameters given to $\mathcal{A}$. Then, $\mathcal{B}$ replies to all queries of $\mathcal{A}$ by simply running $\mathsf{SH.Prv}$. Notice that $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids, thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the EZK game with the same success probability.

$\mathcal{H}_3$: $\mathcal{B}$ replies to $\mathsf{OSign}(\tau, m, i, \mathcal{R})$ queries using w_{trap} instead of $\mathsf{sk}^i_{\mathsf{T}}$ as a witness to compute the proof Π_{EP}. This hybrid is computationally indistinguishable from the previous one thanks to the FPWI property (Def. 18) of EP. Let $\mathcal{C}$ be the challenger of Def. 18. Upon a sign query, $\mathcal{B}$ creates $c_i \leftarrow \mathsf{Com}_{\mathsf{ck}}(e_i; r_i)$ with $e_i \leftarrow \mathsf{Tag}(\mathsf{sk}^i_{\mathsf{T}}, \tau, m)$, $\widetilde{c_i} \leftarrow \mathsf{Com}_{\mathsf{ck}}(1; \widetilde{r_i})$, $\widetilde{c_j} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0; \widetilde{r_j})$ and $c_j \leftarrow \mathsf{Com}_{\mathsf{ck}}(0; r_j)$ for $j \in [n] \setminus i$. $\mathcal{B}$ then sets $x_j = (c_j, \mathsf{pk}^j_{\mathsf{T}}, \tau, m, x_{\mathsf{trap}})$ with $j \in [n]$. Since each OSign queries are performed only on not corrupted indices i, it holds that $\mathcal{B}$ can choose the secret keys for the signer with index i. $\mathcal{B}$ sends $(1, x_1, \ldots, x_n, \widetilde{c_1}, \ldots, \widetilde{c_n}, w_0, w_1)$ to $\mathcal{C}$ with $w_0 = (\phi, (e_i, r_i, \mathsf{sk}^i_{\mathsf{T}}), \widetilde{r_1}, \ldots, \widetilde{r_n})$, $w_1 = (\phi, w_{\mathsf{trap}}, \widetilde{r_1}, \ldots, \widetilde{r_n})$, and $\phi = \{1 \to i\}$. $\mathcal{C}$ replies with a proof Π_{EP}, and a list of auxiliary values AUX. $\mathcal{B}$ can now construct a reply to the sign query $\sigma = (1, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C, \widetilde{C}, \{e_i\})$ where Π_{EP} is the one received by $\mathcal{C}$, the auxiliary values related to Π_{EP} received from $\mathcal{C}$ are encrypted by $\mathcal{B}$ in A_{EP}, C and $\widetilde{C}$ contain the commitments computed as above, Π_{SH} with its encrypted auxiliary values are regularly computed by $\mathcal{B}$, while $e_i \leftarrow \mathsf{Tag}(\mathsf{sk}^i_{\mathsf{T}}, \tau, m)$. $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids and thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the FPWI game with the same success probability.

$\mathcal{H}_4$: $\mathcal{B}$ replies to $\mathsf{OJoin}(\tau, m, \mathcal{R}, i, \sigma)$ queries using w_{trap} instead of $\mathsf{sk}^i_{\mathsf{T}}$ as a witness to compute the add operation over the proof Π_{EP} contained in σ.
This hybrid is computationally indistinguishable from the previous one thanks to the WAI (Def. 17) of EP. Let $\mathcal{C}$ be the challenger of the WAI game. Upon a join query, $\mathcal{B}$ parses the queried $\sigma = (k, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C, \widetilde{C}, \mathsf{E})$ and decrypts the auxiliary value aux_i in $\mathsf{A}_{\mathsf{EP}}[i]$ (recall that signer with index i is not corrupted, therefore $\mathcal{B}$ knows his secret keys). Then $\mathcal{B}$ defines $x = (k, x_1, \ldots, x_n, \widetilde{C})$ with $x_j = (c_j, \mathsf{pk}^j_{\mathsf{T}}, \tau, m, x_{\mathsf{trap}})$ where c_j is the j-th element of C, for all $j \in [n]$. $\mathcal{B}$ samples a randomness $\widetilde{r'_i}$ and transformation randomness R_{Add} and sends $(x, \Pi_{\mathsf{EP}}, i, \mathsf{aux}_i, e_i, w_0, w_1, \widetilde{r'_i}, R_{\mathsf{Add}})$ to $\mathcal{C}$, where $w_0 = (e_i, r_i, \mathsf{sk}^i_{\mathsf{T}})$ and $w_1 = w_{\mathsf{trap}}$. Then, $\mathcal{B}$ gets $x', \Pi'_{\mathsf{EP}}, \mathsf{aux}'_i, (r_1, \ldots, r_n)$ from $\mathcal{C}$. $\mathcal{B}$ now computes the reply to the query $\sigma' = (k+1, \Pi'_{\mathsf{EP}}, \Pi'_{\mathsf{SH}}, \mathsf{A}'_{\mathsf{EP}}, \mathsf{A}'_{\mathsf{SH}}, C', \widetilde{C}', \mathsf{E}')$ in the following way. It sets Π'_{EP} as the one received from $\mathcal{C}$, it updates the auxiliary values in A_{EP} using $(r_1, \ldots, r_n)$ and $\mathsf{PKE.Eval}_1$, it sets C' and $\widetilde{C}'$ as the one contained in x'. It computes all the elements related to the shuffle proofs as in the previous hybrid, using commitment randomness $\widetilde{r'_i}$ and transformation randomness R_{Add}. $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids and thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the WAI game with the same success probability.

$\mathcal{H}_5$: When processing lad_0, $\mathcal{B}$ processes the sign action in the ladder with the same modification of $\mathcal{H}_3$. Indistinguishability from the previous hybrid can be argued as above.

$\mathcal{H}_6$: When processing the ladder, $\mathcal{B}$ processes the join actions in the ladder with the same modification of $\mathcal{H}_4$. Indistinguishability from the previous hybrid can be argued as above.

$\mathcal{H}_7$: Let s be the index of the first signer in lad_0. After that $\mathcal{A}$ returns $(\tau^*, m^*, \mathsf{lad}^*_0, \mathsf{lad}^*_1)$ (line 4 of $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A}, \mathfrak{Tetris}}(\lambda)$), the first action of lad_0 that is $\mathsf{Sign}(\tau^*, m, s, \mathcal{R})$ is processed by $\mathcal{B}$ setting e as $e \leftarrow\!\!\$\ \mathcal{E}$ instead of $e \leftarrow \mathsf{Tag}(\mathsf{sk}^s_{\mathsf{T}}, \tau^*, m)$. This hybrid is computationally indistinguishable from the previous one

thanks to the pseudo-randomness property of the DAPT (Def. 23). Let $\mathcal{C}$ be the challenger of Def. 23. Every time $\mathcal{A}$ makes a KeyGen query with index i, $\mathcal{B}$ generates the $\mathsf{pk}_{\mathsf{T}}^{i}$ starting a new interaction with a pseudo-randomness tag game. Every time $\mathcal{A}$ makes a Sign or a Join query on an index i with public key $\mathsf{pk}_{\mathsf{T}}^{i}$ for a topic τ at line 4 of $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$, $\mathcal{B}$ queries OTag (Fig. 9) to obtain the tag for $(\mathsf{sk}_{\mathsf{T}}^{i}, \tau, m)$ to compute the signature. Notice that for each different index i, $\mathcal{B}$ is interacting with a different OTag in order to obtain the output of $\mathsf{OTag}(\mathsf{sk}_{\mathsf{T}}^{i}, \tau, m, \mathsf{abrt} = 0, \mathsf{honest} = 1)$. Once $\mathcal{A}$ returns $(\tau^*, m^*, \mathsf{lad}_0^*, \mathsf{lad}_1^*)$, $\mathcal{B}$ sets s as the index of the first signer in lad_0. $\mathcal{B}$ generates Σ in line 5 of $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ interacting with OTag as follows: for each index $i \neq s$, $\mathcal{A}$ obtains the tag calling $\mathsf{OTag}(\mathsf{sk}_{\mathsf{T}}^{i}, \tau^*, m^*, \mathsf{abrt} = 0, \mathsf{honest} = 1)$; while for each action of type $\mathsf{Join}(\tau^*, m^*, \mathcal{R}, s, \sigma)$ and $\mathsf{Sign}(\tau^*, m^*, s, \mathcal{R})$, $\mathcal{A}$ obtains the tag calling $\mathsf{OTag}(\mathsf{sk}_{\mathsf{T}}^{s}, \tau^*, m^*, \mathsf{abrt} = 0, \mathsf{honest} = 0)$. $\mathcal{B}$ returns Σ to $\mathcal{A}$ and replies all queries in line 6 of $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ from $\mathcal{A}$ as follows. For Sign and Join queries on all positions $i \neq s$, $\mathcal{B}$ uses a tag obtained by $\mathsf{OTag}(\mathsf{sk}_{\mathsf{T}}^{i}, \tau, m, \mathsf{abrt} = 0, \mathsf{honest} = 1)$. For sign and join operations on position s, $\mathcal{B}$ uses a tag obtained by $\mathsf{OTag}(\mathsf{sk}_{\mathsf{T}}^{s}, \tau, m, \mathsf{abrt} = 0, \mathsf{honest} = 1)$. For each corruption query on $i \neq s$, $\mathcal{B}$ calls $\mathsf{OTag}(\bot, \bot, \bot, \mathsf{abrt} = 1, \bot)$ and returns $\mathsf{sk}_{\mathsf{T}}^{i}$ to $\mathcal{A}$. Notice that by construction, an admissible $\mathcal{A}$ makes $\mathcal{B}$ ask exclusively admissible queries in the tag pseudo-randomness game w.r.t. $\mathsf{pk}_{\mathsf{T}}^{s}$. $\mathcal{B}$ perfectly simulates $\mathcal{H}_6$ if $\mathcal{C}$ always honestly generated tags (i.e., $b = 0$ in $\mathsf{Exp}^{\mathrm{TagPR}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$ w.r.t. $\mathsf{pk}_{\mathsf{T}}^{s}$ Fig. 9) and perfectly simulates $\mathcal{H}_7$ if $\mathcal{C}$ always returns (when $\mathsf{honest} = 0$) random tags (i.e., $b = 1$ in $\mathsf{Exp}^{\mathrm{TagPR}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$ w.r.t. $\mathsf{pk}_{\mathsf{T}}^{s}$ Fig. 9). Thus, a successful distinguisher between the two hybrids can be used to break the tag pseudo-randomness of DAPT with the same success probability.

$\mathcal{H}_8$: $\mathcal{B}$ modifies the tag for the second signer in lad_0 to be $e \leftarrow\!\!\$\, \mathcal{E}$ for the Join action done while computing the signatures in Σ associated to lad_0. Moreover, we implicitly add another set of hybrids, one for each of the additional signers in lad_0. For each of these hybrids, $\mathcal{B}$ sets $e \leftarrow\!\!\$\, \mathcal{E}$ for the current signer. All these hybrids are indistinguishable for the same argument of indistinguishability between $\mathcal{H}_7$ and $\mathcal{H}_6$.

$\mathcal{H}_9$: When processing the ladder, for every join action on an index i, $\mathcal{B}$ uses $(\mathsf{EP.AddT}_x, \mathsf{EP.AddT}_w, \mathsf{EP.Prv})$ instead of $\mathsf{EP.PAdd}$. Additionally, instead of performing AuxUpd, each element of A_{EP} is replaced with a fresh encryption of the auxiliary values output by the Prv algorithm, except in the signer's ciphertexts where $\mathcal{B}$ keeps encrypting $\bot$. This hybrid is computationally indistinguishable from the previous one thanks to the addition privacy (Def. 14) of EP. Whenever $\mathcal{B}$ has to perform a Join action on a signature $\sigma = (k, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A}_{\mathsf{EP}}, \mathsf{A}_{\mathsf{SH}}, C, \widetilde{C}, \mathsf{E})$, $\mathcal{B}$ sets $x = (k, x_1, \ldots, x_n, \widetilde{C})$ with $x_j = (c_j, \mathsf{pk}_{\mathsf{T}}^{j}, \tau, m, x_{\mathsf{trap}})$ where c_j is the j-th element of C. $\mathcal{B}$ sets $w = (\phi, \{w_{\mathsf{trap}}\}^k, \widetilde{r_1}, \ldots, \widetilde{r_n})$ that is the witness that was used to compute Π_{EP}. $\mathcal{B}$ sets $w^* = w_{\mathsf{trap}}$ and sends $(x, w, w^*, i, \widetilde{r'_i}, \Pi_{\mathsf{EP}}, \mathsf{AUX}_{\mathsf{EP}}, R_{\mathsf{Add}})$ where $\mathsf{AUX}_{\mathsf{EP}}$ is known to $\mathcal{B}$ since it processed all the ladders by itself, while $\widetilde{r'_i}$ and R_{Add} are uniformly sampled commitment randomness and transformation randomness. $\mathcal{B}$ now computes the next signature in the ladder $\sigma' = (k+1, \Pi'_{\mathsf{EP}}, \Pi'_{\mathsf{SH}}, \mathsf{A}'_{\mathsf{EP}}, \mathsf{A}'_{\mathsf{SH}}, C', \widetilde{C}', \mathsf{E}')$ in the following way. It sets Π'_{EP} as the one received from $\mathcal{C}$, it updates the auxiliary values in $\mathsf{A}'_{\mathsf{EP}}$ by encrypting the auxiliary values received by $\mathcal{C}$ in the non-signers' positions (and $\bot$ in the signers' ones), sets C' and $\widetilde{C}'$ as the one contained in x'. It computes all the elements related to the shuffle proofs as in the previous hybrid, using commitment randomness $\widetilde{r'_i}$ and transformation randomness R_{Add}. $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids and thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the addition privacy game with the same success probability.

$\mathcal{H}_{10}$: When processing the ladder, $\mathcal{B}$ uses $(\mathsf{EP.ExtT}_x, \mathsf{EP.ExtT}_w, \mathsf{EP.Prv})$ instead of $\mathsf{EP.PExt}$. Additionally, instead of performing AuxUpd, each element of A_{EP} is replaced with a fresh encryption of the auxiliary values in output of Prv algorithm. This hybrid is computationally indistinguishable from the previous one thanks to the extension privacy of EP. Indistinguishability can be argued in a similar way to as was done in $\mathcal{H}_9$.

$\mathcal{H}_{11}$: $\mathcal{B}$ uses $(\mathsf{SH.AddT}_x, \mathsf{SH.AddT}_w, \mathsf{SH.Prv})$ instead of $\mathsf{SH.PAdd}$. Additionally, instead of performing AuxUpd, each element of A_{SH} is replaced with a fresh encryption of the auxiliary values in output of $\mathsf{SH.Prv}$ algorithm. This hybrid is computationally indistinguishable from the previous one thanks to the addition privacy of SH. Indistinguishability can be argued in a similar way to as was done in $\mathcal{H}_9$.

$\mathcal{H}_{12}$: $\mathcal{B}$ uses $(\mathsf{SH.ExtT}_x, \mathsf{SH.ExtT}_w, \mathsf{SH.Prv})$ instead of $\mathsf{SH.PExt}$. Additionally, instead of performing AuxUpd, each element of $\mathsf{A_{SH}}$ is replaced with a fresh encryption of the auxiliary values in output of Prv algorithm. This hybrid is computationally indistinguishable from the previous one thanks to the extension privacy of SH. Indistinguishability can be argued in a similar way to as was done in $\mathcal{H}_9$.

$\mathcal{H}_{13}$: When processing the ladders, $\mathcal{B}$ encrypts $\perp$ in all the signers' ciphertexts of $\mathsf{A_{EP}}$ and $\mathsf{A_{SH}}$. Notice that, at this point, $\mathcal{B}$ is already not using any auxiliary value. Recall that an admissible $\mathcal{A}$ cannot corrupt any of the signers in lad_0 or lad_1. This hybrid is computationally indistinguishable from the previous one thanks to the IND-CPA property of the encryption scheme.

$\mathcal{H}_{14}$: When processing the ladder, $\mathcal{B}$ uses $\mathsf{SH.Sim}_1$ instead of $\mathsf{SH.Prv}$ when producing Π_{SH} and associated auxiliary values. This hybrid is computationally indistinguishable from the previous one thanks to the EZK property (Def. 20) of SH. Let $\mathcal{C}$ be the challenger of the EZK game. Whenever $\mathcal{B}$ has to compute a shuffle proof to process the ladder, it simply forwards $(x_{\mathsf{SH}}, w_{\mathsf{SH}})$ to $\mathcal{C}$ and puts in the next signature in the ladder Π_{SH} received from $\mathcal{C}$, $\mathsf{A_{SH}}$ is computed by encrypting the *non*-signers' auxiliary values AUX received from $\mathcal{C}$ and encrypting $\perp$ in the signers' auxiliary values. The remaining parts of the signatures are computed as in the previous hybrid. $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids and thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the EZK game with the same success probability.

$\mathcal{H}_{15}$: Let C_i be the list of commitments to tags contained in σ_i (i.e. the i-th signature in Σ). For all $i \in [\ell]$, where ℓ is the length of the ladder, $\mathcal{B}$ now creates C_i by committing to 0 in all positions. This hybrid is computationally indistinguishable from the previous one thanks to the computational hiding of $\mathsf{Com_{ck}}$. Let $\mathcal{C}$ be the challenger of the computational hiding game. For every Sign or Join action in the ladder that adds an additional signer with index j, $\mathcal{B}$, instead of directly creating the j-th commitment within C_i, for $i \in \ell$, by directly committing to e_j, it queries $\mathcal{C}$ with $(e_j, 0)$. Then, $\mathcal{B}$ puts the received commitment in $C_i[j]$. Notice that at this point $\mathcal{B}$ does not need the opening of any of the commitments to compute any of the proofs. $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids and thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the computational hiding game with the same success probability.

$\mathcal{H}_{16}$: Let C_i be the list of commitments to tags contained in σ_i. For all $i \in [\ell]$, $\mathcal{B}$ now creates the C_i by committing to the tags implicitly specified by lad_1. Indistinguishability can be shown as done in $\mathcal{H}_{15}$.

$\mathcal{H}_{17}$: Let E_i be the list of clear-text tags contained in σ_i. $\mathcal{B}$ now computes the clear-text tag to be (if needed) added to each E_i as the corresponding tags from lad_1. This hybrid is indistinguishable from the previous one thanks to the tag pseudo-randomness of DAPT. Indistinguishability can be argued in the same way as $\mathcal{H}_7$ and $\mathcal{H}_8$.

$\mathcal{H}_{18}$: $\mathcal{B}$ uses the DAPT secret keys related to lad_1 to compute the EP proofs. In particular, at the i-th step of the ladder with $i \in [\ell]$, $\mathcal{B}$ creates the commitments in $\widetilde{C}$ are according to the signers of the i-th step in lad_1. This hybrid is computationally indistinguishable from the previous one thanks to the EWI of EP (Def. 19). Let $\mathcal{C}$ be the challenger of the EWI game. Let us consider the first step y in which the signers are different in lad_0 and lad_1. $\mathcal{B}$ sets $x = (k, x_1, \ldots, x_n, \{\perp\}^k)$, with $x_j = (c_j, \mathsf{pk}^j_{\mathsf{T}}, \tau, m, x_{\mathsf{trap}})$ where c_j is the j-th element of C computed as in the previous hybrid. $\mathcal{B}$ sets $w_0 = (\phi_0, \{w_{\mathsf{trap}}\}^k)$, and $w_1 = (\phi_1, \{(e_j, \mathsf{sk}_{\mathsf{T}}^{\phi_1(j)}, r_{\phi_1(j)})\}_{j \in [k]})$ where ϕ_0 and ϕ_1 are the mappings induced by lad_0 and lad_1 respectively. $\mathcal{B}$ sends (x, w_0, w_1) to $\mathcal{C}$ and gets back $\Pi_{\mathsf{EP}}, \mathsf{AUX_{EP}}, \widetilde{C}$. $\mathcal{B}$ then computes the signature $\sigma_y = (k, \Pi_{\mathsf{EP}}, \Pi_{\mathsf{SH}}, \mathsf{A_{EP}}, \mathsf{A_{SH}}, C, \widetilde{C}, \mathsf{E})$ where Π_{EP} and $\widetilde{C}$ are the ones received from $\mathcal{C}$, $\mathsf{AUX_{EP}}$ includes all the auxiliary values of non-signers' positions which can be encrypted by $\mathcal{B}$ and included in $\mathsf{A_{EP}}$, while the other positions are still encrypting $\perp$. The shuffle proofs can be computed as before as they are simulated and $\mathcal{B}$ does not need the openings of $\widetilde{C}$. $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in both hybrids and thus $\mathcal{B}$ can use the output of a successful distinguisher to win in the extended witness indistinguishability game with the same success probability.

$\mathcal{H}_{19}$: We start a sequence of hybrids that progressively removes the use of $\mathsf{SH.Sim}, w_{\mathsf{trap}}$ and that switches back to using the extension and addition algorithms for Extend and Join queries/operations. To do that, we undo the changes from $\mathcal{H}_{14}$ to $\mathcal{H}_9$ and from $\mathcal{H}_6$ to $\mathcal{H}_1$. All of these hybrids are indistinguishable for the same reasons argued above. The final hybrid is exactly $\mathsf{Exp}^{\mathrm{Anon}}_{\mathcal{A},\mathfrak{Tetris}}(\lambda)$ with $b = 1$.

Lemma 7. *Let* DAPT *be defined as in Sec. 5. Let* EP *be defined as in Sec. 4 for relation* R_{Dtr} *with WAI (Def. 17) and FPWI (Def. 18). Then, the scheme in Fig. 11,12 is exculpable (Def. 7).*

Proof. Let us call $\mathcal{B}$ the reduction that runs $\mathcal{A}$ internally. The proof uses the following indistinguishable hybrids.

$\mathcal{H}_0$: This is exactly the experiment of Fig. 7.

$\mathcal{H}_1$: Instead of sampling a x_{trap} such that $x_{\mathsf{trap}} \notin \mathcal{L}_{\mathsf{trap}}$, $\mathcal{B}$ samples $(x_{\mathsf{trap}}, w_{\mathsf{trap}}) \in R_{\mathcal{L}_{\mathsf{trap}}}$. This hybrid is computationally indistinguishable from the previous one thanks to the hardness of $\mathcal{L}_{\mathsf{trap}}$.

$\mathcal{H}_2$: This is equivalent to $\mathcal{H}_1$ except that $\mathcal{B}$ guesses an index i^* such that $\mathsf{pk}_{i^*} \in \mathsf{PK}$ at the end of exculpability experiment (see Fig. 7). The index i^* must exist, otherwise $\mathcal{A}$ would not win the exculpability game. From now on, every corruption query on i^* results in an abort of the experiment. Indeed, under the condition that i^* is correctly guessed, $\mathcal{A}$ cannot corrupt pk_{i^*} to win the exculpability game. Thus, the probability that $\mathcal{B}$ does not abort in this hybrid is at least $\frac{1}{q_{KG}+1}$, where q_{KG} is a polynomial bound on the number of key generation queries $\mathcal{A}$ can do.

$\mathcal{H}_3$: $\mathcal{B}$ replies to $\mathsf{Sign}(\tau, m, i^*, \mathcal{R})$ queries using w_{trap} instead of $\mathsf{sk}_{\mathsf{T}}^{i^*}$ as a witness to compute the proof Π_{EP}. This hybrid is indistinguishable from the previous thanks to the FPWI of EP (Def. 18). The proof is identical to the one of $\mathcal{H}_3$ of the proof of anonymity.

$\mathcal{H}_4$: $\mathcal{B}$ replies to $\mathsf{Join}(\tau, m, \mathcal{R}, i^*, \sigma)$ queries using w_{trap} instead of $\mathsf{sk}_{\mathsf{T}}^{i^*}$ as a witness to compute the add operation over the proof Π_{EP} contained in σ. This hybrid is indistinguishable from the previous thanks to the WAI of EP (Def. 17). The proof is identical to the one of $\mathcal{H}_4$ of the proof of anonymity (Lem. 6).

$\mathcal{H}_5$: Let $\mathcal{C}$ be the challenger of the tag non-frameability game $\mathsf{Exp}^{\mathrm{TagNF}}_{\mathcal{A},\mathsf{DAPT}}(\lambda)$ (Fig. 10). Whenever asked to generate a key for index i^*, $\mathcal{B}$ forwards such KeyGen query to $\mathcal{C}$ and uses its reply to answer $\mathcal{A}$. Whenever $\mathcal{A}$ asks for a Sign or Join query over index i^*, $\mathcal{B}$ queries $\mathcal{C}$ to get the corresponding tag needed to answer the queries. Notice that at this point, $\mathcal{B}$ does not need $\mathsf{sk}_{\mathsf{T}}^{i^*}$ to answer any query involving i^*. Whenever $\mathcal{A}$ outputs $(\tau, t_0, t_1, \mathcal{R}_0, \mathcal{R}_1, m_0, m_1, \sigma_0, \sigma_1)$, $\mathcal{B}$ looks for a pair e_0, e_1 (respectively contained in tinfo_0 of σ_0 and tinfo_1 of σ_1) such that $\mathsf{TagTrace}(e_0, e_1, m_0, m_1, \mathsf{pk}_{\mathsf{T}}^{i^*}) = 1$ and sends (e_0, e_1, m_0, m_1) to $\mathcal{C}$. Conditioned on correctly guessing i^*, $\mathcal{B}$ wins the tag non-frameability game with the same probability with which $\mathcal{A}$ wins the exculpability game. Notice that all the queries forwarded to $\mathcal{C}$ are admissible for a $\mathcal{B}$ playing the tag non-frameability game.

Lemma 8. *Let* DAPT *be defined as in Sec. 5. L Let* EP *be defined as in Sec. 4 for relation* R_{Dtr} *with perfect* Δ*-extraction (Def. 16) where* Δ *is defined as* $\Delta((\hat{e}_j, r_{\hat{z}_j}, \mathsf{sk}_{\mathsf{T}}^j)_{j\in[k]}) = (\hat{e}_1, \ldots, \hat{e}_k)$*. Let* SH *be defined as in Sec. 4 for relation* R_{SH}*. Then, the scheme in Fig. 11,12 is traceable (Def. 5).*

Proof. It follows from Lem. 4, Lem. 5, and Lem. 2.

7 Our DAPT

Let $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h})$ where $\mathsf{gk} \leftarrow\!\!\$\ \mathcal{G}(1^\lambda)$. The topic space is $\{0,1\}^*$, the message space is $\mathbb{Z}_p$, and the tag space is $\hat{\mathbb{G}}$. The algorithms work as follows:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{gk})$: Sample random $x, y \leftarrow\!\!\$\ \mathbb{Z}_p$, compute $\mathsf{pk} = (\hat{\mathsf{pk}}_1 = \hat{g}x, \hat{\mathsf{pk}}_2 = y\hat{g}, \check{\mathsf{pk}}_2 = \check{h}y)$ and set $\mathsf{sk} = (x, y)$.
- $\hat{e} \leftarrow \mathsf{Tag}(\mathsf{sk}, \tau, m)$: Let $\hat{\tau} \leftarrow H(\tau)$ output $\hat{e} = \hat{\tau}x + \hat{\mathsf{pk}}_2 mx$.
- $0/1 \leftarrow \mathsf{TagTrace}(\hat{e}_0, \hat{e}_1, m_0, m_1, \mathsf{pk})$: If $(\hat{e}_0 - \hat{e}_1) \cdot \check{h} = \hat{\mathsf{pk}}_1 \cdot (m_0 - m_1)\check{\mathsf{pk}}_2$ return 1, else return 0.
- $R_{\mathsf{key}} = \{((\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \check{\mathsf{pk}}_2), (x, y)) | \hat{\mathsf{pk}}_1 = x\hat{g}, \hat{\mathsf{pk}}_2 = \hat{g}y, \check{\mathsf{pk}}_2 = \check{h}y\}$.

Theorem 3. *If the DDH and the Co-CDH assumptions hold relative to* $\mathcal{G}$*, then the scheme above is a* DAPT *in the ROM.*

Proof. We prove the above theorem via the following three lemmas.

Lemma 9. *The* DAPT *described above enjoys verifiability of keys (cfr, Def. 21).*

Proof. Given the poly-time relation R_{key}, one can check by inspection that the following three conditions hold.

1. For every $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{gk})$, it hold that $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}$.
2. Every $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}$ lies in the output space of $\mathsf{KeyGen}(\mathsf{gk})$.
3. For a given pk, there exists a unique sk such that $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}$. This is because the exponentiation is a unique map.

Lemma 10. *The* DAPT *described above enjoys tag traceability (cfr, Def. 22).*

Proof. First, let us consider two honestly generated tags $\hat{e}_0, \hat{e}_1$ w.r.t. the same key $(\mathsf{pk}, \mathsf{sk})$, the same topic τ, and messages m_0, m_1. We have that $\hat{e}_0 = \hat{\tau}x + \hat{\mathsf{pk}}_1 m_0 x$ and $\hat{e}_1 = \hat{\tau}x + \hat{\mathsf{pk}}_1 m_1 x$. We have that $(\hat{e}_0 - \hat{e}_1) \cdot \check{h} = \hat{\mathsf{pk}}_1 x(m_0 - m_1) \cdot \check{h} = xy\hat{g}(m_0 - m_1) \cdot \check{h}$. On the other hand, $\hat{\mathsf{pk}}_0 \cdot (m_0 - m_1)\check{\mathsf{pk}}_1 = \hat{g}x \cdot (m_0 - m_1)\check{h}y$.

Second, let us consider an $\mathcal{A}$ winning the game of Def. 22 with non-negligible probability. $\mathcal{A}$ outputs $(\tau, m_0, m_1, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{sk}_1)$ such that $\mathsf{sk}_0 = (x_0, y_0)$, $\mathsf{sk}_1 = (x_1, y_1)$, $\mathsf{pk}_0 = (x_0\hat{g}, y_0\hat{g}, \check{h}y_0)$ and $\mathsf{pk}_1 = (x_1\hat{g}, y_1\hat{g}, \check{h}y_1)$. Additionally, given $\hat{e}_0 = \hat{\tau}x_0 + \hat{g}y_0 m_0 x$ and $\hat{e}_1 = \hat{\tau}x_1 + \hat{g}y_1 m_1 x$, where $\hat{\tau} = H(\tau)$, it holds that either $\mathsf{TagTrace}(\hat{e}_0, \hat{e}_1, m_0, m_1, \mathsf{pk}_0) = 1$ or $\mathsf{TagTrace}(\hat{e}_0, \hat{e}_1, m_0, m_1, \mathsf{pk}_1) = 1$. Let us assume, without loss of generality, that $\mathsf{TagTrace}(\hat{e}_0, \hat{e}_1, m_0, m_1, \mathsf{pk}_0) = 1$. This means that:

$$
\begin{gathered}
(\hat{\tau}x_0 + \hat{g}y_0 m_0 x_0 - \hat{\tau}x_1 - \hat{g}y_1 m_1 x_1) \cdot \check{h} = \hat{g}x_0 y_0 (m_0 - m_1) \cdot \check{h} \\
\hat{\tau}(x_0 - x_1) \cdot \check{h} = \hat{g}m_1(x_0 y_0 - x_1 y_1) \cdot \check{h} \\
u\hat{g}(x_0 - x_1) \cdot \check{h} = \hat{g}m_1(x_0 y_0 - x_1 y_1) \cdot \check{h} \\
u = m_1(x_0 y_0 - x_1 y_1)(x_0 - x_1)^{-1}
\end{gathered}
$$

Recall that we know that $x_0 \neq x_1 \wedge y_0 \neq y_1$. Basically, given an $\mathcal{A}$ winning with non-negligible probability, it is possible to find the discrete logarithm (DL) of $H(\tau)$ with the same probability. It is straightforward to build a reduction breaking the DL problem by making it program oracle queries with DL challenges.

Lemma 11. *If the EDDH assumption holds in $\mathbb{G}$ (Ass. 2), the* DAPT *described above enjoys pseudo-randomness (Def. 23) in the random oracle model.*

Proof. To prove this lemma we will transition, through a series of indistinguishable hybrids, from the experiment run with $b = 0$ to the experiment run with $b = 1$.

$\mathcal{H}_0$: This is exactly the pseudo-randomness experiment of Fig. 9 with $b = 0$.

$\mathcal{H}_1$: This is equivalent to $\mathcal{H}_0$ except that, when $\mathcal{B}$ is queried to produce a tag on the pair (τ, m), instead of computing it as $\hat{e} = H(\tau)x + \hat{\mathsf{pk}}_1 my$, the reduction outputs $\hat{e} = \hat{r}_\tau + \hat{\mathsf{pk}}_1 my$ with $\hat{r}_\tau \leftarrow\!\!\$\ \mathbb{G}$. The reduction uses the same $\hat{r}_\tau$ in case the same τ is queried again.

$\mathcal{H}_2$: This is equivalent to $\mathcal{H}_1$ except that, when $\mathcal{B}$ is queried to produce a tag on the pair (τ, m), the reduction samples a random $\hat{e} \leftarrow\!\!\$\ \mathbb{G}$. This is exactly the pseudo-randomness experiment of Fig. 9 with $b = 1$.

We now show that all the consecutive hybrids are indistinguishable:

– $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable under the Type 2 EDDH assumption (which is implied by the EDDH assumption). Indeed, we can use a distinguisher $\mathcal{D}$ that, on input the view of $\mathcal{A}$, distinguishes between $\mathcal{H}_0$ and $\mathcal{H}_1$ with probability ϵ, to build an adversary $\mathcal{B}$ that breaks the EDDH assumption with the same probability. $\mathcal{B}$ creates $\mathsf{pk} = (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \check{\mathsf{pk}}_2)$ as follows. It sets $\hat{\mathsf{pk}}_1 = \hat{a}$, with $\hat{a}$ that it gets from the EDDH game, while it samples $y \leftarrow\!\!\$\ \mathbb{Z}_p$ to create $\hat{\mathsf{pk}}_2 = y\hat{g}$ and $\check{\mathsf{pk}}_2 = \check{h}y$. Whenever $\mathcal{A}$ calls OTag with input a not previously queried (τ_i, m_i) and $\mathsf{honest} = 1$, $\mathcal{B}$ calls the ODDH with $\mathsf{honest} = 1$ and gets $(\hat{t}_i, \hat{z}_i)$. Then, it programs the random oracle on input τ_i to give $\hat{t}_i$ as output. Finally, it replies to the query computing $\hat{e}_i = \hat{z}_i + \hat{\mathsf{pk}}_1 m_i y$. Similarly, whenever $\mathcal{A}$ calls OTag with with input a not previously queried (τ_i, m_i) and $\mathsf{honest} = 0$, $\mathcal{B}$ calls the ODDH oracle of the EDDH game and gets $(\hat{t}_i, \hat{z}_i)$. Then, it programs the random oracle on input τ_i to give $\hat{t}_i$ as output. Finally, it replies to the query computing $\hat{e}_i = \hat{z}_i + \hat{\mathsf{pk}}_1 m_i y$. $\mathcal{B}$ outputs whatever $\mathcal{D}$ outputs. Notice that $\mathcal{B}$ perfectly simulates $\mathcal{H}_0$ whenever the type 2 EDDH experiment is run with $b = 0$, and it perfectly simulates $\mathcal{H}_1$ otherwise.

– $\mathcal{H}_1$ and $\mathcal{H}_2$ are perfectly indistinguishable if $\mathcal{A}$ never queries (τ, m_1) and (τ, m_2) with $m_1 \neq m_2$. Indeed, in this case, $\hat{r}_\tau$ acts as a random mask that is only used once and perfectly hides the group element that is added to it.

Lemma 12. *If the EDDH (Assumption 2) and the Co-CDH (Assumption 5) assumptions hold relative to $\mathcal{G}$, the* DAPT *described above enjoys tag non-frameability (Def. 24) in the random oracle model.*

Proof. To prove this lemma we will transition through a series of indistinguishable hybrids. Then, in the final hybrid we will use the output of the adversary $\mathcal{A}$ of the tag non-frameability experiment to break the Co-CDH assumption.

$\mathcal{H}_0$: This is exactly the tag non-frameability experiment of Fig. 10.
$\mathcal{H}_1$: This is equivalent to $\mathcal{H}_0$ except that, when queried to produce a tag on the pair (τ, m), instead of computing it as $\hat{e} = H(\tau)x + \hat{\mathsf{pk}}_1 my$, the reduction outputs $\hat{e} = \hat{r}_\tau + \hat{\mathsf{pk}}_1 my$ with $\hat{r}_\tau \leftarrow\!\!\$\ \mathbb{G}$. The reduction uses the same $\hat{r}_\tau$ in case the same τ is queried again.
$\mathcal{H}_2$: This is equivalent to $\mathcal{H}_1$ except that the reduction computes the tags simply as random elements of $\mathbb{G}$. The reduction keeps track of these values and returns the same value if the same pair (τ, m) is queried again.

The indistinguishability of all the hybrids can be argued as it was shown while proving the tag indistinguishability property. The key point now is that in $\mathcal{H}_2$, $\mathcal{B}$ can reply to the tag generation queries without knowing any part of the secret key. Therefore, $\mathcal{B}$ will set the public key to give to $\mathcal{A}$ as the Co-CDH challenge. Then whenever $\mathcal{A}$ outputs $(\hat{e}_0, \hat{e}_1, m_0, m_1)$ s.t. $(\hat{e}_0 - \hat{e}_1) \cdot \check{h} = \hat{\mathsf{pk}}_1 \cdot (m_0 - m_1)\check{\mathsf{pk}}_2$, $\mathcal{B}$ will output $\frac{(\hat{e}_0 - \hat{e}_1)}{m_0 - m_1}$. It follows that $\mathcal{B}$ wins the Co-CDH game with the same probability with which $\mathcal{A}$ wins the tag non-frameability game.

Remark 1. As shown in the above security proof, given two tags $\hat{e}_0$ and $\hat{e}_1$ w.r.t. the same topic τ and $m_0 \neq m_1$ it is possible to compute $\hat{t} = \frac{(\hat{e}_0 - \hat{e}_1)}{m_0 - m_1}$. It is also possible to compute $\hat{\tau}x = \hat{e}_0 - \hat{t}m_0$. Given this information, it is easy to issue tags for the topic τ and any message m by computing $\hat{t}m + \hat{\tau}x$.

8 Our Extendable Shuffle Argument

Our extendable shuffle argument (Sec. 1.2) is an EP with EZK for relation R_{SH}:

$$\begin{aligned} R_{\mathsf{SH}} = \{&(\mathsf{ck}, x = (k, \hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_n, \check{\boldsymbol{m}}_1, \ldots, \check{\boldsymbol{m}}_n, \hat{e}_1, \ldots, \hat{e}_k), w = (\phi, r_1, \ldots, r_k, \\ &r_{\check{m}_1}, \ldots, r_{\check{m}_n}))|\forall i \in [n]\ \check{\boldsymbol{m}}_i \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; r_{\check{m}_i}) \wedge \mathsf{bit}_i \in \{0,1\} \wedge \textstyle\sum_{i=1}^n \mathsf{bit}_i = k \wedge \\ &\forall_{i=1}^k (\mathsf{bit}_{\phi(i)} = 1 \wedge (\hat{\boldsymbol{z}}_{\phi(i)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\hat{e}_i; r_i))) \wedge \phi \text{ is an injective map } [k] \to [n]\}. \end{aligned}$$

As commitment scheme, we use ElGamal with commitment keys $\hat{\mathsf{ck}}$ and $\check{\mathsf{ck}}$:

– To commit to a group element $\hat{E}$ using randomness $r_{\hat{z}}$, compute $\hat{\boldsymbol{z}} = (\hat{\boldsymbol{z}}^1, \hat{\boldsymbol{z}}^2) = (r_{\hat{z}}\hat{g}, \hat{E} + r_{\hat{z}}\hat{\mathsf{ck}})$.
– To commit to a bit bit with randomness $r_{\check{m}}$, compute $\check{\boldsymbol{m}} = (\check{\boldsymbol{m}}^1, \check{\boldsymbol{m}}^2) = (r_{\check{m}}\check{h}, \mathsf{bit}\check{h} + r_{\check{m}}\check{\mathsf{ck}})$.

We construct Π_{SH} for relation R_{SH} by defining a set of equations and proving their satisfiability with the GS proof system. Let ψ be a committed variable that is 1 on a regular $\mathsf{crs}_{\mathsf{SH}}$ and can be equivocated to 0 with the simulation trapdoor; such a variable is implicitly available in the crs of GS [36]. The variable ψ is only needed to achieve EZK, so it can be ignored in the following overview in which we describe all the equations we prove and why they give a shuffle argument.

In order to prove the correctness of the switch variables $\{\mathsf{bit}_i : i \in [n]\}$, i.e., that $\forall i \in [n] : \mathsf{bit}_i \in \{0,1\} \wedge \sum_{i=1}^n \mathsf{bit}_i = k$, we define the equations $\mathcal{B}_i : \mathsf{bit}_i(1 - \mathsf{bit}_i) = 0$ and $\mathcal{K} : \sum_{i=1}^n \mathsf{bit}_i = k\psi$.

The main challenge is to devise a set of equations whose satisfiability, proven with GS, implies the statement in the third line of R_{SH}. Towards this goal, we first introduce new auxiliary variables $\hat{E}_i, f_i, r_{\hat{z}_i}$ and define the following equations (with public constants $\hat{\mathsf{ck}}, \check{\mathsf{ck}}, \hat{g}, \check{h}, \hat{\boldsymbol{z}}_i, \check{\boldsymbol{m}}_i$):

$$(\forall i)\ \mathcal{D}^1_{i,1} : f_i \hat{\boldsymbol{z}}_{\boldsymbol{i}}{}^1 = r_{\hat{z}_i}\hat{g},\ \ \mathcal{D}^1_{i,2} : f_i \hat{\boldsymbol{z}}^2_i = f_i \hat{E}_i + r_{\hat{z}_i}\hat{\mathsf{ck}},\ \ \mathcal{F}_i : f_i = \mathsf{bit}_i \psi$$

$$(\forall i)\ \mathcal{D}^2_{i,1} : \psi\check{\boldsymbol{m}}^1_i = r_{\check{m}_i}\check{h},\ \ \mathcal{D}^2_{i,2} : \psi\check{\boldsymbol{m}}^2_1 = \mathsf{bit}_i\check{h} + r_{\check{m}_i}\check{\mathsf{ck}}.$$

The equations in the first line guarantee that the prover can open k commitments to tags, while the equations in the second line guarantee that the i-th switch variable bit_i is correctly committed in $\check{\boldsymbol{m}}_i$.

In order to prove the shuffle w.r.t. the k public values $(\hat{e}_1, \ldots, \hat{e}_k)$, we extend the techniques of Groth and Lu [36] as follows. We introduce new variables $\{\check{a}_i, \check{a'}_i, \check{b}_i, \check{b'}_i, \hat{c}_i : i \in [n]\}$ and define the following equations (with public constants $\hat{g}, \check{h}, \check{h}_i, \check{\delta}_i$), where all $\check{h}_i, \check{\delta}_i$ with $i \in [n]$ are part of $\mathsf{crs}_{\mathsf{SH}}$ and are generated as stated in Ass. 6.

$$\mathcal{Q}^1_i : \check{a}_i = \mathsf{bit}_i\check{a'}_i \quad (\forall i)\ \mathcal{Q}^2_i : \check{b}_i = \mathsf{bit}_i\check{b'}_i$$

$$\mathcal{S}^1 : \textstyle\sum_{i=1}^n \check{a}_i = \sum_{i=1}^k \psi\check{h}_i \quad \mathcal{S}^2 : \sum_{i=1}^n \check{b}_i = \sum_{i=1}^k \psi\check{\delta}_i$$

$$(\forall i)\ \mathcal{V}^1_i : \hat{g} \cdot \check{a}_i = \hat{c}_i \cdot \check{h} \quad (\forall i)\ \mathcal{V}^2_i : \hat{g} \cdot \check{b}_i = \hat{c}_i \cdot \check{a}_i$$

The above equations formulate a subset permutation pairing problem over the variables $\{\check{a}_i, \check{b}_i\}$ as stated in Assumption 6. In particular, for all $i \in [n]$ the equations $\mathcal{Q}^1_i, \mathcal{Q}^2_i$, together with equations $\mathcal{B}_i$ and $\mathcal{K}$ guarantee that at least k of the variables $\check{a}_i$ and $\check{b}_i$ are set to $\check{0}$. Additionally, if the equations $\mathcal{S}^1, \mathcal{S}^2, \mathcal{V}^1, \mathcal{V}^2$ are satisfied then, thanks to the subset permutation assumption, there exists except with negligible probability a set of indices $\mathcal{J} = \{\alpha_1, \ldots, \alpha_k\}$ with $1 \le \alpha_i \le n$ s.t. $\{(\check{a}_{\alpha_i}, \check{b}_{\alpha_i})\}_{i\in[k]}$ is a permutation of $\{(\check{h}_i, \check{\delta}_i)\}_{i\in[k]}$.

Namely, the switch variables bit_i modify the equations in such a way that setting $\mathsf{bit}_i = 1$ implies that $i \in \mathcal{J}$. Finally, we define the following equations with variables $\hat{o}, \hat{o'}, \psi, \hat{E}_i, \check{a}_i, \check{b}_i$ and public constants $\hat{e}_i, \check{h}, \check{\delta}_i, \check{h}_i$.

$$\mathcal{E}^1 : \hat{o} \cdot \check{h} + \textstyle\sum_{i=1}^n \hat{E}_i \cdot \check{a}_i = \sum_{i=1}^k \hat{e}_i \cdot \check{h}_i,\ \ \mathcal{E}^2 : \hat{o'} \cdot \check{h} + \sum_{i=1}^n \hat{E}_i \cdot \check{b}_i = \sum_{i=1}^k \hat{e}_i \cdot \check{\delta}_i$$

$$\mathcal{P}^2 : \psi\hat{o} = \hat{0} \quad \mathcal{P}^3 : \psi\hat{o'} = \hat{0}$$

As for ψ, the introduction of the variables $\hat{o}$ and $\hat{o'}$ is only useful to the simulator. Let us look at the equations in the first line, we have that $\sum_{i=1}^n \hat{E}_i \cdot \check{a}_i = \sum_{i=1}^k \hat{e}_i \cdot \check{h}_i$ and $\sum_{i=1}^n \hat{E}_i \cdot \check{b}_i = \sum_{i=1}^k \hat{e}_i \cdot \check{\delta}_i$. Given that $\{(\check{a}_{\alpha_i}, \check{b}_{\alpha_i})\}_{i\in[k]}$ is a permutation of $\{(\check{h}_i, \check{\delta}_i)\}_{i\in[k]}$, there must exist a permutation π such that $\sum_{i=1}^k (\hat{E}_{\alpha_{\pi(i)}} - \hat{e}_i) \cdot \check{h}_i = \hat{0}$ and $\sum_{i=1}^k (\hat{E}_{\alpha_{\pi(i)}} - \hat{e}_i) \cdot \check{\delta}_i = \hat{0}$. This constitutes a subset simultaneous pairing problem (see Assumption 7), and assuming its computational hardness, we can infer that $\hat{E}_{\alpha_{\pi(i)}} = \hat{e}_i$ for all $i \in [k]$. To prove all the above equations the prover assigns the value to the variables in the following way:

- For all $i \in [n]$ set $r_{\check{m}_i}$ as the value given in the witness.
- Set $\mathcal{I} \leftarrow \emptyset$. For all $j \in [k]$ let $\phi(j) = i$, set $\check{a}_i = \check{a'}_i = \check{h}_j, \check{b}_i = \check{b'}_i = \check{\delta}_j, \hat{c}_i = \hat{g}_j, \hat{E}_i = \hat{e}_j, \mathsf{bit}_i = 1, f_1 = 1, r_{\hat{z}_i} = r_j, \mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$.
- For all $i \in [n] \setminus \mathcal{I}$, set $\check{a}_i = \check{a'}_i = \check{0}, \check{b}_i = \check{b'}_i = \check{0}, \hat{c}_i = \hat{0}, \hat{E}_i = \hat{0}, \mathsf{bit}_i = 0, f_i = 0, r_{\hat{z}_i} = 0$.

The simulation trapdoor consists of the trapdoor of the GS proof system, that allows equivocating ψ to 0, and the discrete logs $\{p_i\}_{i\in[n]}$ of the elements $\{\hat{g}_i\}_{i\in[n]}$. Indeed, to simulate a proof without knowledge of the witness it suffices to set variables $\hat{o} = \sum_{i=1}^k p_i\hat{e}_i, \hat{o'} = \sum_{i=1}^k 2p_i\hat{e}_i$, $(\forall i)\ \check{a}_i = \check{a'}_i = \check{0}, \check{b}_i = \check{b'}_i = \check{0}, \hat{c}_i = \hat{0}, \hat{E}_i = \hat{0}, \mathsf{bit}_i = 0, f_i = 0, r_{\hat{z}_i} = 0, r_{\check{m}_i} = 0$. The extended zero knowledge property follows from the witness indistinguishability of GS proofs and from the fact that the variables corresponding to the inactive positions are assigned in the same way both for real and simulated proofs.

Extend and add operations can be implemented with the same techniques of [8] since, as for their ENIWI, they only involve adding/removing the contribution of certain variables within a set of equations (i.e., $\mathcal{K}, \mathcal{S}^1, \mathcal{S}^2, \mathcal{E}^1, \mathcal{E}^2$). Whenever an extend/add operation is performed, the lists of commitments $\{\hat{\boldsymbol{z}}_{\boldsymbol{i}}\}_{i\in[n]}$ to group elements and commitments $\{\check{\boldsymbol{m}}_{\boldsymbol{i}}\}_{i\in[n]}$ to switch variables are also re-randomized (ElGamal is re-randomizable). Applying this re-randomization and updating the GS proofs consequently is straightforward given the malleability of the ElGamal encryption and of GS proofs. We defer to App. B for a detailed description of the construction and its security proofs.

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (Dec 2002). https://doi.org/10.1007/3-540-36178-2_26
2. Aguilar Melchor, C., Cayrel, P.L., Gaborit, P.: A new efficient threshold ring signature scheme based on coding theory. In: Buchmann, J., Ding, J. (eds.) Post-quantum cryptography, second international workshop, PQCRYPTO 2008. pp. 1–16. Springer, Heidelberg (Oct 2008). https://doi.org/10.1007/978-3-540-88403-3_1
3. Aranha, D.F., Baum, C., Gjøsteen, K., Silde, T., Tunge, T.: Lattice-based proof of shuffle and applications to electronic voting. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 227–251. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_10
4. Aranha, D.F., Hall-Andersen, M., Nitulescu, A., Pagnin, E., Yakoubov, S.: Count me in! extendability for threshold ring signatures. Cryptology ePrint Archive, Paper 2021/1240 (2021), `https://eprint.iacr.org/2021/1240`, `https://eprint.iacr.org/2021/1240`
5. Aranha, D.F., Hall-Andersen, M., Nitulescu, A., Pagnin, E., Yakoubov, S.: Count me in! Extendability for threshold ring signatures. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part II. LNCS, vol. 13178, pp. 379–406. Springer, Heidelberg (Mar 2022). https://doi.org/10.1007/978-3-030-97131-1_13
6. Attema, T., Cramer, R., Rambaud, M.: Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 526–556. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_18
7. Au, M.H., Liu, J.K., Susilo, W., Yuen, T.H.: Secure id-based linkable and revocable-iff-linked ring signature with constant-size construction. Theor. Comput. Sci. **469**, 1–14 (2013)
8. Avitabile, G., Botta, V., Fiore, D.: Extendable threshold ring signatures with enhanced anonymity. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 281–311. Springer, Heidelberg (May 2023). https://doi.org/10.1007/978-3-031-31368-4_11
9. Avitabile, G., Botta, V., Friolo, D., Visconti, I.: Efficient proofs of knowledge for threshold relations. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) ESORICS 2022, Part III. LNCS, vol. 13556, pp. 42–62. Springer, Heidelberg (Sep 2022). https://doi.org/10.1007/978-3-031-17143-7_3
10. Backes, M., Döttling, N., Hanzlik, L., Kluczniak, K., Schneider, J.: Ring signatures: Logarithmic-size, no setup - from standard assumptions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 281–311. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17659-4_10
11. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 108–125. Springer, Heidelberg (Aug 2009). https://doi.org/10.1007/978-3-642-03356-8_7
12. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_38
13. Bettaieb, S., Schrek, J.: Improved lattice-based threshold ring signature scheme. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013. pp. 34–51. Springer, Heidelberg (Jun 2013). https://doi.org/10.1007/978-3-642-38616-9_3
14. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 464–492. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_16
15. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_26
16. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J.: Foundations of fully dynamic group signatures. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16. LNCS, vol. 9696, pp. 117–136. Springer, Heidelberg (Jun 2016). https://doi.org/10.1007/978-3-319-39555-5_7
17. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: Pernul, G., Ryan, P.Y.A., Weippl, E.R. (eds.) ESORICS 2015, Part I. LNCS, vol. 9326, pp. 243–265. Springer, Heidelberg (Sep 2015). https://doi.org/10.1007/978-3-319-24174-6_13
18. Branco, P., Mateus, P.: A traceable ring signature scheme based on coding theory. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography. pp. 387–403. Springer International Publishing, Cham (2019)

19. Branco, P., Mateus, P.: A traceable ring signature scheme based on coding theory. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019. pp. 387–403. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-25510-7_21
20. Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 465–480. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_30
21. Catalano, D., Fuchsbauer, G., Soleimanian, A.: Double-authentication-preventing signatures in the standard model. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 338–358. Springer, Heidelberg (Sep 2020). https://doi.org/10.1007/978-3-030-57990-6_17
22. Chakraborty, S., Hofheinz, D., Langrehr, R., Nielsen, J.B., Striecks, C., Venturi, D.: Malleable SNARKs and their applications. Cryptology ePrint Archive, Paper 2025/311 (2025), https://eprint.iacr.org/2025/311
23. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_18
24. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Succinct malleable NIZKs and an application to compact shuffles. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 100–119. Springer, Heidelberg (Mar 2013). https://doi.org/10.1007/978-3-642-36594-2_6
25. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT'91. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (Apr 1991). https://doi.org/10.1007/3-540-46416-6_22
26. Derler, D., Ramacher, S., Slamanig, D.: Generic double-authentication preventing signatures and a post-quantum instantiation. In: Baek, J., Susilo, W., Kim, J. (eds.) ProvSec 2018. LNCS, vol. 11192, pp. 258–276. Springer, Heidelberg (Oct 2018). https://doi.org/10.1007/978-3-030-01446-9_15
27. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 609–626. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_36
28. Escala, A., Groth, J.: Fine-tuning Groth-Sahai proofs. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 630–649. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_36
29. Fujisaki, E.: Sub-linear size traceable ring signatures without random oracles. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 393–415. Springer, Heidelberg (Feb 2011). https://doi.org/10.1007/978-3-642-19074-2_25
30. Fujisaki, E., Suzuki, K.: Traceable ring signature. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 181–200. Springer, Heidelberg (Apr 2007). https://doi.org/10.1007/978-3-540-71677-8_13
31. Fujisaki, E., Suzuki, K.: Traceable ring signature. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **91-A**(1), 83–93 (2008). https://doi.org/10.1093/IETFEC/E91-A.1.83, https://doi.org/10.1093/ietfec/e91-a.1.83
32. Ghadafi, E.: Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 327–347. Springer, Heidelberg (Sep 2015). https://doi.org/10.1007/978-3-319-16295-9_18
33. Ghadafi, E., Smart, N.P., Warinschi, B.: Groth-Sahai proofs revisited. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 177–192. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-13013-7_11
34. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 458–487. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_16
35. González, A.: Shorter ring signatures from standard assumptions. In: Lin, D., Sako, K. (eds.) PKC 2019, Part I. LNCS, vol. 11442, pp. 99–126. Springer, Heidelberg (Apr 2019). https://doi.org/10.1007/978-3-030-17253-4_4
36. Groth, J., Lu, S.: A non-interactive shuffle with pairing based verifiability. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 51–67. Springer, Heidelberg (Dec 2007). https://doi.org/10.1007/978-3-540-76900-2_4
37. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_24
38. Haque, A., Scafuro, A.: Threshold ring signatures: New definitions and post-quantum security. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 423–452. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45388-6_15
39. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 04. LNCS, vol. 3108, pp. 325–335. Springer, Heidelberg (Jul 2004). https://doi.org/10.1007/978-3-540-27800-9_28

40. Malavolta, G., Schröder, D.: Efficient ring signatures in the standard model. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 128–157. Springer, Heidelberg (Dec 2017). https://doi.org/10.1007/978-3-319-70697-9_5
41. Munch-Hansen, A., Orlandi, C., Yakoubov, S.: Stronger notions and a more efficient construction of threshold ring signatures. In: Longa, P., Ràfols, C. (eds.) LATINCRYPT 2021. LNCS, vol. 12912, pp. 363–381. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-88238-9_18
42. Naor, M.: Deniable ring authentication. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 481–498. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_31
43. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001. pp. 116–125. ACM Press (Nov 2001). https://doi.org/10.1145/501983.502000
44. Petzoldt, A., Bulygin, S., Buchmann, J.: A multivariate based threshold ring signature scheme. Appl. Algebra Eng. Commun. Comput. **24**(3-4), 255–275 (2013)
45. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_32
46. Russo, A., Anta, A.F., Vasco, M.I.G., Romano, S.P.: Chirotonia: A Scalable and Secure e-Voting Framework based on Blockchains and Linkable Ring Signatures. In: 2021 IEEE International Conference on Blockchain (Blockchain). pp. 417–424 (2021)
47. Scafuro, A., Zhang, B.: One-time traceable ring signatures. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021, Part II. LNCS, vol. 12973, pp. 481–500. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-88428-4_24
48. Thyagarajan, S.A.K., Malavolta, G., Schmid, F., Schröder, D.: Verifiable timed linkable ring signatures for scalable payments for monero. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) ESORICS 2022, Part II. LNCS, vol. 13555, pp. 467–486. Springer, Heidelberg (Sep 2022). https://doi.org/10.1007/978-3-031-17146-8_23
49. Tsang, P.P., Wei, V.K.: Short linkable ring signatures for e-voting, e-cash and attestation. In: Deng, R.H., Bao, F., Pang, H., Zhou, J. (eds.) Information Security Practice and Experience, First International Conference, ISPEC 2005, Singapore, April 11-14, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3439, pp. 48–60. Springer (2005), https://doi.org/10.1007/978-3-540-31979-5_5
50. Xu, S., Yung, M.: Accountable ring signatures: A smart card approach. In: Quisquater, J.J., Paradinas, P., Deswarte, Y., El Kalam, A.A. (eds.) Smart Card Research and Advanced Applications VI. pp. 271–286. Springer US, Boston, MA (2004)
51. Yuen, T.H., Esgin, M.F., Liu, J.K., Au, M.H., Ding, Z.: DualRing: Generic construction of ring signatures with efficient instantiations. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 251–281. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_10
52. Yuen, T.H., Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Efficient Linkable and/or Threshold Ring Signature Without Random Oracles. Comput. J. **56**(4), 407–421 (2013)

Appendices

A Standard Tools and Definitions

A.1 Commitment Schemes

A non-interactive commitment is a pair of PPT algorithms $(\mathsf{Setup}, \mathsf{Com})$, where Setup takes as input 1^λ and returns a commitment key ck and Com take as input a message m, and outputs a commitment $\mathsf{Com} = \mathsf{Com}_{\mathsf{ck}}(m; r)$, where r is the randomness used to generate Com. The pair (m, r) is called the opening. Intuitively, a commitment satisfies two properties called binding and hiding. The first property says that it is hard to open a commitment in two different ways. The second property says that a commitment hides the underlying message.

Definition 25 (Perfect Binding). *We say that a non-interactive commitment is* perfectly binding *if* $\nexists\, m_0 \neq m_1, r_0, r_1$ *s.t.* $\mathsf{Com}_{\mathsf{ck}}(m_0; r_0) = \mathsf{Com}_{\mathsf{ck}}(m_1; r_1)$*, where* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$.

Definition 26 (Computational Hiding). *We say that a non-interactive commitment is* computationally hiding *if for all* PPT *adversaries* $\mathcal{A}$ *the following quantity is negligible*

$$\left|\Pr\left[\mathcal{A}^{\mathsf{O}(\mathsf{ck},0,\cdot,\cdot)}(1^\lambda, \mathsf{ck}) = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{O}(\mathsf{ck},1,\cdot,\cdot)}(1^\lambda, \mathsf{ck}) = 1\right]\right|,$$

where the oracle $\mathsf{O}(\mathsf{ck}, b, \cdot, \cdot)$ *with hard-wired* $b \in \{0, 1\}$ *and* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ *takes as input a pair of messages* m_0, m_1*, and outputs* $\mathsf{Com}_{\mathsf{ck}}(m_b)$.

A.2 Public Key Encryption

A public key encryption scheme is a set of PPT algorithms $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: outputs public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$: generates a new public and secret key pair.
- $\mathsf{a} \leftarrow \mathsf{Enc}(m, \mathsf{pk})$: on input a message m, and a public key pk, output a ciphertext a.
- $m \leftarrow \mathsf{Dec}(\mathsf{a}, \mathsf{sk})$ on input a ciphertext a, and a secret key sk, output a message m.

Additionally, a public key encryption scheme is homomorphic w.r.t a function f, if there exists a PPT algorithm that works as follows.

- $\mathsf{a}' \leftarrow \mathsf{Eval}(\mathsf{a}, x, \mathsf{pk})$: on input a ciphertext a, a message x, and the public key pk. Let $y \leftarrow \mathsf{Dec}(\mathsf{a}, \mathsf{sk})$, it returns a ciphertext a' s.t. $f(y, x) = \mathsf{Dec}(\mathsf{a}', \mathsf{sk})$.

A public key encryption scheme is is IND-CPA secure if the probability that a PPT adversary $\mathcal{A}$ wins the following game is negligibly close to $\frac{1}{2}$. The game involves the following steps: (i) $\mathcal{A}$ has access to the public key and outputs two messages m_0 and m_1; (ii) the challenger encrypts one of the two messages; (iii) $\mathcal{A}$ has to guess which message was encrypted.

ElGamal Encryption. The ElGamal encryption scheme is a public key encryption scheme with the following algorithms. The public parameters pp produced by Setup are implicitly available to all other algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: on input the security parameter, sample a cyclic group $\hat{\mathbb{G}}$ of prime order p, a generator $\hat{g}$. Output $\mathsf{pp} = (\hat{\mathbb{G}}, \hat{g})$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$: sample an element $\zeta \leftarrow\!\!\$\ \mathbb{Z}_p^*$. Define public key as $\mathsf{pk} = \hat{\boldsymbol{v}} = (\zeta\hat{g}, \hat{g})^\top \in \hat{\mathbb{G}}^{2\times 1}$ and $\mathsf{sk} = \boldsymbol{\zeta} = (-\zeta^{-1}, 1)$. Output $(\mathsf{pk}, \mathsf{sk})$.

– $\hat{\mathbf{a}} \leftarrow \mathsf{Enc}(\hat{m}, \mathsf{pk})$: with input the public key and a message $\hat{m} \in \hat{\mathbb{G}}$, sample $r \leftarrow\!\$\ \mathbb{Z}_p$ and output ciphertext $\hat{\mathbf{a}} = \boldsymbol{e}^\top \hat{m} + \hat{\boldsymbol{v}} r \in \hat{\mathbb{G}}^{2\times 1}$, where $\boldsymbol{e} = (0, 1)$.
– $\hat{m} \leftarrow \mathsf{Dec}(\hat{\mathbf{a}}, \mathsf{sk})$: with input the secret key and a ciphertext $\mathbf{a} \in \hat{\mathbb{G}}^{2\times 1}$, output $\hat{m} = \boldsymbol{\zeta}\hat{\mathbf{a}}$.

The ElGamal encryption scheme is also homomorphic, with the function f being the group operation. In more detail:

– $\mathbf{a}' \leftarrow \mathsf{Eval}(\mathbf{a}_1, \hat{m}_2, \mathsf{pk})$: compute $\mathbf{a_2} = \mathsf{Enc}(\hat{m}_2, \mathsf{pk})$, output $\mathbf{a}' = \mathbf{a}_1 + \mathbf{a}_2$. If the ciphertexts contained messages $\hat{m}_1$ and $\hat{m}_2$, the output ciphertext will contain message $\hat{m}_1 + \hat{m}_2$.

The ElGamal encryption is IND-CPA secure if the DDH assumption holds in $\hat{\mathbb{G}}$. Additionally, ciphertexts updated with Eval are identically distributed to freshly generated ciphertexts.

We also add the following definition.

Definition 27 (Verifiability of Keys). *For all $\lambda \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, it holds that $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}()$ iff $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathsf{PKE}}$. We also require that for all pk there exists a* unique *sk s.t. $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathsf{PKE}}$.*

Note that the ElGamal encryption scheme satisfies Def. 27 for $R_{\mathsf{key}}^{\mathsf{PKE}} = \{(\mathsf{pk}, \mathsf{sk}) | \exists \zeta \in \mathbb{Z}_p^* \text{ s.t. } \mathsf{pk} = (\zeta\hat{g}, \hat{g})^\top \wedge \mathsf{sk} = (-\zeta^{-1}, 1)\}$. Indeed, it holds that:

1. For every $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$, it hold that $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathsf{PKE}}$.
2. Every $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathsf{PKE}}$ lies in the output space of $\mathsf{KeyGen}()$.
3. For a given pk, there exists a unique sk such that $(\mathsf{pk}, \mathsf{sk}) \in R_{\mathsf{key}}^{\mathsf{PKE}}$. This is because the exponentiation is a unique map.

A.3 Non-Interactive Proof Systems

Let us consider an NP language $\mathcal{L}$ with associated poly-time relation $R_{\mathcal{L}}$. A non-interactive proof system for $R_{\mathcal{L}}$ consists of the following algorithms. The group key $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$ is considered as an implicit input to all algorithms.

– $(\mathsf{crs}, \mathsf{xk}) \leftarrow\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$: on input the group key, output a common reference string $\mathsf{crs} \in \{0,1\}^\lambda$, and an extraction trapdoor xk.
– $\Pi \leftarrow \mathsf{Prv}(\mathsf{crs}, x, w)$: on input statement x and witness w s.t. $(x, w) \in R_{\mathcal{L}}$, output a proof Π.
– $0/1 \leftarrow \mathsf{PVfy}(\mathsf{crs}, x, \Pi)$: on input statement x and proof Π, output either 1 to accept or 0 to reject.
– $\Pi' \leftarrow \mathsf{RandPr}(\mathsf{crs}, x, \Pi)$: on input statement x and proof Π for $x \in \mathcal{L}$, output a randomized proof Π'.

A non-interactive proof system is said to be witness indistinguishable (NIWI) if all the properties below are satisfied.

Definition 28 (Completeness). *A proof system for $R_{\mathcal{L}}$ is complete if $\forall \lambda \in \mathbb{N}$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{crs} \leftarrow\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$, $(x, w) \in R_{\mathcal{L}}$, and $\Pi \leftarrow \mathsf{Prv}(\mathsf{crs}, x, w)$ it holds that $\Pr[\mathsf{PVfy}(\mathsf{crs}, x, \Pi) = 1] = 1$.*

Definition 29 (Witness Indistinguishability). *We say that the proof system is witness indistinguishable (WI) if the following holds. For all (x, w_1, w_2) such that $(x, w_1), (x, w_2) \in R_{\mathcal{L}}$, the tuples (crs, Π_1) and (crs, Π_2), where $\mathsf{crs} \leftarrow\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$ and for $i \in [2]$, $\Pi_i \leftarrow \mathsf{Prv}(\mathsf{crs}, x, w_i)$, are computationally indistinguishable. If the two tuples are identically distributed, we say that the proof system is perfect WI.*

Definition 30 (Soundness). *For all PPT $\mathcal{A}$, and for $\mathsf{crs} \leftarrow\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$, $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, the probability that $\mathcal{A}(\mathsf{crs})$ outputs (x, Π) such that $x \notin \mathcal{L}$ but $\mathsf{PVfy}(\mathsf{crs}, x, \Pi) = 1$, is negligible.*

Additionally, a NIWI is said to be a NIWI proof of knowledge (PoK) if the property below is also satisfied.

Definition 31 (Adaptive Extractable Soundness). *There exists a polynomial-time extractor $\mathsf{Ext} = (\mathsf{Ext}_0, \mathsf{Ext}_1)$, for all $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$, with the following properties:*

– $\mathsf{Ext}_0(\mathsf{gk})$ *outputs* $(\mathsf{crs}_{\mathsf{Ext}}, \mathsf{xk})$ *such that* $\mathsf{crs}_{\mathsf{Ext}}$ *is indistinguishable from* crs *obtained running* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$.
– *For all* PPT $\mathcal{A}$*, the probability that* $\mathcal{A}(\mathsf{crs}_{\mathsf{Ext}}, \mathsf{xk})$ *outputs* (x, Π) *such that* $\mathsf{PVfy}(\mathsf{crs}_{\mathsf{Ext}}, x, \Pi) = 1$ *and* $(x, w) \notin R_{\mathcal{L}}$ *where* $w \leftarrow \mathsf{Ext}_1(\mathsf{crs}_{\mathsf{Ext}}, \mathsf{xk}, x, \Pi)$ *is negligible.*

We can also consider a stronger notion than witness indistinguishability called zero knowledge.

Definition 32 (Zero Knowledge). *There exists a polynomial-time simulator algorithm* $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ *such that* $\mathsf{Sim}_0(\mathsf{gk})$ *outputs* $(\mathsf{crs}, \mathsf{td})$*, and* $\mathsf{Sim}_1(\mathsf{crs}, \mathsf{td}, x)$ *outputs a value* π *such that for all* $(x, w) \in R_{\mathcal{L}}$ *and* PPT *adversaries* $\mathcal{A}$*, the following two interactions are indistinguishable: in the first, we compute* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$ *and give* $\mathcal{A}$ *both* crs *and oracle access to* $\mathsf{Prv}(\mathsf{crs}, \cdot, \cdot)$ *(where* Prv *will output* $\perp$ *on input* $(x, w) \notin R_{\mathcal{L}}$*); in the second, we compute* crs *as* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(\mathsf{gk})$*, and give* $\mathcal{A}$ *such* crs *and oracle access to* $\mathsf{Sim}_2(\mathsf{crs}, \mathsf{td}, \cdot, \cdot)$*, with the exception that it outputs* $\perp$ *whenever* $(x, w) \notin R_{\mathcal{L}}$.

An extra property of a proof system is re-randomizability, which we state below. In our paper, when talking about a proof system we usually refer to a re-randomizable proof system.

Definition 33 (Re-Randomizable Proof System). *Consider the following experiment:*

– $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$
– $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$
– $(x, w, \Pi) \leftarrow \mathcal{A}(\mathsf{crs})$
– *If either* $\mathsf{PVfy}(\mathsf{crs}, x, \Pi) = 0$ *or* $(x, w) \notin R_{\mathcal{L}}$ *output* $\perp$ *and abort. Otherwise, sample* $b \leftarrow\!\!\$\ \{0, 1\}$
  • *If* $b = 0$ $\Pi' \leftarrow \mathsf{Prv}(\mathsf{crs}, x, w)$
  • *If* $b = 1$ $\Pi' \leftarrow \mathsf{RandPr}(\mathsf{crs}, x, \Pi)$
– $b' \leftarrow \mathcal{A}(\Pi')$.

We say that the proof system is re-randomizable if for every PPT $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$*, such that* $\Pr[b = b'] \leq 1/2 + \nu(\lambda)$.

A.4 Groth-Sahai Proofs

The Groth-Sahai (GS) proof system [37] is a proof system for the language of satisfiable equations (of types listed below) over a bilinear group $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h}) \leftarrow \mathcal{G}(1^\lambda)$. The prover wants to show that there exists an assignment of all the variables that satisfies the equation. Such equations can be of four types:

Pairing-product equations (PPE): For public constants $\hat{a}_j \in \hat{\mathbb{G}}$, $\check{b}_i \in \check{\mathbb{H}}$, $\gamma_{ij} \in \mathbb{Z}_p$, $t_{\mathbb{T}} \in \mathbb{T}$: $\sum_i \hat{x}_i \cdot \check{b}_i + \sum_j \hat{a}_j \cdot \check{y}_j + \sum_i \sum_j \gamma_{ij} \hat{x}_i \cdot \check{y}_j = t_{\mathbb{T}}$.

Multi-scalar multiplication equation in $\hat{\mathbb{G}}$ ($\mathbf{ME}_{\hat{\mathbb{G}}}$): For public constants $\hat{a}_j \in \hat{\mathbb{G}}$, $b_i \in \mathbb{Z}_p$, $\gamma_{ij} \in \mathbb{Z}_p$, $\hat{t} \in \hat{\mathbb{G}}$: $\sum_i \hat{x}_i b_i + \sum_j \hat{a}_j y_j + \sum_i \sum_j \gamma_{ij} \hat{x}_i y_j = \hat{t}$.

Multi-scalar multiplication equation in $\check{\mathbb{H}}$ ($\mathbf{ME}_{\check{\mathbb{H}}}$): For public constants $a_j \in \mathbb{Z}_p$, $\check{b}_i \in \check{\mathbb{H}}$, $\gamma_{ij} \in \mathbb{Z}_p$, $\check{t} \in \check{\mathbb{H}}$: $\sum_i x_i \check{b}_i + \sum_j a_j \check{y}_j + \sum_i \sum_j \gamma_{ij} x_i \check{y}_j = \check{t}$.

Quadratic equation in $\mathbb{Z}_p$ (QE): For public constants $a_j \in \mathbb{Z}_p$, $b_i \in \mathbb{Z}_p$, $\gamma_{ij} \in \mathbb{Z}_p$, $t \in \mathbb{Z}_p$: $\sum_i x_i b_i + \sum_j a_j y_j + \sum_i \sum_j \gamma_{ij} x_i y_j = t$.

We use the notation and formalization of the GS proof system proposed in [28]. The GS proof system is a commit-and-prove system. Each committed variable is also provided with a public label that specifies the type of input (i.e., scalar or group element). Accordingly, the prover algorithm takes as input a label L which indicates the type of equation to be proved (i.e., $L \in \{\mathrm{PPE}, \mathrm{ME}_{\hat{\mathbb{G}}}, \mathrm{ME}_{\check{\mathbb{H}}}, \mathrm{QE}\}$). GS features the following PPT algorithms, the common reference string crs and the group key gk are considered as implicit input of all the algorithms.

- $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSSetup}(\mathsf{gk})$: on input the group key, output the common reference string. The common reference string defines the parameters of the commitment scheme.
- $(l, c) \leftarrow \mathsf{Com}(l, w; r)$: return a commitment (l, c) to message w according to the label l and randomness r.
- $\pi \leftarrow \mathsf{Prv}(L, x, (l_1, w_1, r_1), \ldots, (l_n, w_n, r_n))$: consider statement x as an equation of type specified by L, and on input a list of commitment openings produce a proof π.
- $0/1 \leftarrow \mathsf{PVfy}(x, (l_1, c_1), \ldots, (l_n, c_n), \pi)$: given committed variables, statement x, and proof π, output 1 to accept and 0 to reject.
- $((l_1, c'_1), \ldots, (l_n, c'_n), \pi') \leftarrow \mathsf{RandPr}(L, (l_1, c_1), \ldots, (l_n, c_n), \pi; r)$: on input equation type specified by L, a list of commitments, a proof π, and a randomness r, output a re-randomized proof along with the corresponding list of re-randomized commitments.

GS can also be used to prove that a set of equations S, with possibly shared variables across the equations, has a satisfying assignment. To do so, the prover reuses the same commitments for the shared variables while executing the Prv algorithm for each individual equation. Notice that a commit-and-prove system can always fit the interface of a regular proof system by including both the commitments and proof elements in the proof itself.

The GS proof system is a NIWI for all types of the above equations under the SXDH assumption. In addition, it is a PoK for all equations involving solely group elements. To be more specific, Escala and Groth formulated the notion of F-knowledge [28] for a commit-and-prove system (a variation of adaptive extractable soundness). In a nutshell, it requires the existence of an Ext_2 algorithm that, on input a valid commitment and the extraction key produced by an algorithm Ext_1 which is also producing the crs, outputs a function F of the committed value. They prove that GS enjoys F-knowledge. For commitments to group elements, F is the identity function. Regarding commitments to scalars, F is a one-way function (i.e., exponentiation) that uniquely determines the committed value. Additionally, GS proofs are zero knowledge for equations of type $\mathrm{ME}_{\hat{\mathbb{G}}}, \mathrm{ME}_{\check{\mathbb{H}}}$, QE, and for pairing-product equations in which $t_{\mathbb{T}} = 0_{\mathbb{T}}$ and no public constants are paired with each other. However, zero knowledge can be achieved if a representation of $t_{\mathbb{T}}$ in terms of the source groups is known. If this representation is known in terms of $\hat{g}$ and $\check{h}$, zero knowledge comes basically at no extra cost.

An important feature of GS is that it follows the dual-mode paradigm. Basically, the CRS can be set in two modes that are indistinguishable under the SXDH assumption. In perfect binding mode, the CRS can be sampled with a trapdoor that allows for extraction and thus defines a perfectly binding commitment scheme giving a perfect proof of knowledge with computational zero knowledge. On the contrary, the perfect hiding mode CRS comes with a trapdoor that allows simulation and thus defines a perfectly hiding commitment scheme giving perfect WI (or ZK) and computational extractable soundness.

Internals of GS proofs. In [28], the authors provide a very fine-grained description of GS proofs. In this description, we report only the aspects that are relevant to our constructions. It is possible to write the equations of Sec. A.4 in a more compact way. Consider $\hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_m)$ and $\check{\boldsymbol{y}} = (\check{y}_1, \ldots, \check{y}_n)$, which may be both public constants (i.e., written before as $\hat{a}_j, \check{b}_i$) or secret values. Let $\Gamma = \{\gamma_{ij}\}_{i=1,j=1}^{m,n} \in \mathbb{Z}_p^{m \times n}$. We can now write a PPE as $\hat{\boldsymbol{x}} \Gamma \check{\boldsymbol{y}} = t_{\mathbb{T}}$. Similarly, a $\mathrm{ME}_{\hat{\mathbb{G}}}$, a $\mathrm{ME}_{\check{\mathbb{H}}}$, and a QE can be written as $\hat{\boldsymbol{x}} \Gamma \boldsymbol{y} = \hat{t}$, $\boldsymbol{x} \Gamma \check{\boldsymbol{y}} = \check{t}$, and $\boldsymbol{x} \Gamma \boldsymbol{y} = t$. This holds for $\hat{\boldsymbol{x}} \in \hat{\mathbb{G}}^{1 \times m}, \check{\boldsymbol{y}} \in \check{\mathbb{H}}^{n \times 1}, \boldsymbol{x} \in \mathbb{Z}_p^{1 \times m}, \boldsymbol{y} \in \mathbb{Z}_p^{n \times 1}$. The structure of the crs is clear from Fig. 13, where the generation of perfectly binding crs (i.e., via Ext_0) and of a perfectly hiding crs (i.e., via Sim_0) are shown.

In Fig. 14, we report the commitment labels and corresponding commit algorithm that are of interest for this work. Commitments of type $\mathsf{unit}_{\hat{\mathbb{G}}}, \mathsf{base}_{\hat{\mathbb{G}}}, \mathsf{unit}_{\check{\mathbb{H}}}$, and $\mathsf{base}_{\check{\mathbb{H}}}$ have a distinctive feature that is crucial for simulation. The simulation trapdoor specifies ρ such that $\hat{\boldsymbol{u}} = \rho \hat{\boldsymbol{v}}$ and $\boldsymbol{e}^\top \hat{g} = \rho \hat{\boldsymbol{v}} - \hat{\boldsymbol{w}}$. This means that commitments of types $\mathsf{base}_{\hat{\mathbb{G}}}$ and $\mathsf{unit}_{\hat{\mathbb{G}}}$ can be equivocated as either commitments to $\hat{g}$ and 1 or as commitments to $\hat{0}$ and 0. The same holds for commitments of type $\mathsf{unit}_{\check{\mathbb{H}}}$ and $\mathsf{base}_{\check{\mathbb{H}}}$.

In Fig. 15 and in Fig. 16, we report the prover and verifier algorithm respectively. In Fig. 17 we report the proof re-randomization algorithm.

$(\mathsf{crs}, \mathsf{xk}) \leftarrow \mathsf{Ext}_0(\mathsf{gk})$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(\mathsf{gk})$
1 : Parse $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h})$	1 : Parse $\mathsf{gk} = (p, \hat{\mathbb{G}}, \check{\mathbb{H}}, \mathbb{T}, e, \hat{g}, \check{h})$
2 : $\rho \leftarrow\!\$\, \mathbb{Z}_p, \xi \leftarrow\!\$\, \mathbb{Z}_p^*$ and $\sigma \leftarrow\!\$\, \mathbb{Z}_p, \psi \leftarrow\!\$\, \mathbb{Z}_p^*$	2 : $\rho \leftarrow\!\$\, \mathbb{Z}_p, \xi \leftarrow\!\$\, \mathbb{Z}_p^*$ and $\sigma \leftarrow\!\$\, \mathbb{Z}_p, \psi \leftarrow\!\$\, \mathbb{Z}_p^*$
3 : $\hat{\boldsymbol{v}} = (\xi\hat{g}, \hat{g})^\top$ and $\check{\boldsymbol{v}} = (\psi\check{h}, \check{h})$	3 : $\hat{\boldsymbol{v}} = (\xi\hat{g}, \hat{g})^\top$ and $\check{\boldsymbol{v}} = (\psi\check{h}, \check{h})$
4 : $\hat{\boldsymbol{w}} = \rho\hat{\boldsymbol{v}}$ and $\check{\boldsymbol{w}} = \sigma\check{\boldsymbol{v}}$	4 : $\hat{\boldsymbol{w}} = \rho\hat{\boldsymbol{v}} - (\hat{0}, \hat{g})^\top$ and $\check{\boldsymbol{w}} = \sigma\check{\boldsymbol{v}} - +(\check{0}, \hat{g})$
5 : $\hat{\boldsymbol{u}} = \hat{\boldsymbol{w}} + (\hat{0}, \hat{g})^\top$ and $\check{\boldsymbol{u}} = \check{\boldsymbol{w}} + (\check{0}, \hat{g})$	5 : $\hat{\boldsymbol{u}} = \hat{\boldsymbol{w}} + (\hat{0}, \hat{g})^\top$ and $\check{\boldsymbol{u}} = \check{\boldsymbol{w}} + (\check{0}, \hat{g})$
6 : $\boldsymbol{\xi} = (-\xi^{-1} \mod p, 1)$ and	6 : $\mathsf{crs} = (\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{w}}, \check{\boldsymbol{u}}, \check{\boldsymbol{v}}, \check{\boldsymbol{w}})$
7 : $\boldsymbol{\psi} = (-\psi^{-1} \mod p, 1)^\top$	7 : $\mathsf{td} = (\rho, \sigma)$
8 : $\mathsf{crs} = (\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{w}}, \check{\boldsymbol{u}}, \check{\boldsymbol{v}}, \check{\boldsymbol{w}})$	8 : **return** $(\mathsf{crs}, \mathsf{td})$
9 : $\mathsf{xk} = (\boldsymbol{\xi}, \boldsymbol{\psi})$	
10 : **return** $(\mathsf{crs}, \mathsf{xk})$	

Fig. 13. Generation of the CRS along with either the extraction key or the trapdoor key in the GS proof system.

Input	Randomness	Output
$\mathsf{pub}_{\hat{\mathbb{G}}}, \hat{x}$	$r = 0, s = 0$	$\hat{\boldsymbol{c}} = \boldsymbol{e}^\top \hat{x}$
$\mathsf{com}_{\hat{\mathbb{G}}}, \hat{x}$	$r, s \leftarrow\!\$\, \mathbb{Z}_p$	$\hat{\boldsymbol{c}} = \boldsymbol{e}^\top \hat{x} + \hat{\boldsymbol{v}} r + \hat{\boldsymbol{w}} s$
$\mathsf{base}_{\hat{\mathbb{G}}}, \hat{g}$	$r = 0, s = 0$	$\hat{\boldsymbol{c}} = \boldsymbol{e}^\top \hat{g}$
$\mathsf{sca}_{\hat{\mathbb{G}}}, x$	$r \leftarrow\!\$\, \mathbb{Z}_p, s = 0$	$\hat{\boldsymbol{c}} = \hat{\boldsymbol{u}} x + \hat{\boldsymbol{v}} r$
$\mathsf{unit}_{\hat{\mathbb{G}}}, 1$	$r = 0, s = 0$	$\hat{\boldsymbol{c}} = \hat{\boldsymbol{u}}$

Input	Randomness	Output
$\mathsf{pub}_{\check{\mathbb{H}}}, \check{y}$	$r = 0, s = 0$	$\check{\boldsymbol{d}} = \check{y}\boldsymbol{e}$
$\mathsf{com}_{\check{\mathbb{H}}}, \check{x}$	$r, s \leftarrow\!\$\, \mathbb{Z}_p$	$\check{\boldsymbol{d}} = \check{y}\boldsymbol{e} + r\check{\boldsymbol{v}} + s\check{\boldsymbol{w}}$
$\mathsf{base}_{\check{\mathbb{H}}}, \check{h}$	$r = 0, s = 0$	$\check{\boldsymbol{d}} = \check{h}\boldsymbol{e}$
$\mathsf{sca}_{\check{\mathbb{H}}}, y$	$r \leftarrow\!\$\, \mathbb{Z}_p, s = 0$	$\check{\boldsymbol{d}} = y\check{\boldsymbol{u}} + r\check{\boldsymbol{v}}$
$\mathsf{unit}_{\check{\mathbb{H}}}, 1$	$r = 0, s = 0$	$\check{\boldsymbol{d}} = \check{\boldsymbol{u}}$

Fig. 14. GS commit labels and corresponding commit algorithm, $\boldsymbol{e} = (0, 1)$.

For the sake of simplicity, we omit the explicit description of Sim_1 which can be found in [28]. In a nutshell, the simulator sets all of the committed variables to the corresponding neutral element with the goal of using a trivial assignment to satisfy the equation. Having the trapdoor, the simulator can equivocate the commitments of type $\mathsf{base}_{\hat{\mathbb{G}}}, \mathsf{unit}_{\hat{\mathbb{G}}}, \mathsf{base}_{\check{\mathbb{H}}}, \mathsf{unit}_{\check{\mathbb{H}}}$ to the neutral element. This gives the simulator a satisfying witness, and using this simulated witness it can now create the proof as an honest prover would do.

$\mathsf{Prv}(L, \Gamma, \{(l_{x_i}, x_i, (r_{x_i}, s_{x_i}))\}_{i=1}^m, \{(l_{y_j}, y_j, (r_{y_j}, s_{y_j}))\}_{j=1}^n)$

if $\boldsymbol{x} \in \hat{\mathbb{G}}^m$ define $\hat{C} = \boldsymbol{e}^\top \boldsymbol{x} + \hat{\boldsymbol{v}} \boldsymbol{r}_x + \hat{\boldsymbol{w}} \boldsymbol{s}_x$ **else if** $\boldsymbol{x} \in \mathbb{Z}_p^m$ define $\hat{C} = \hat{\boldsymbol{u}} \boldsymbol{x} + \hat{\boldsymbol{v}} \boldsymbol{r}_x$

if $\boldsymbol{y} \in \check{\mathbb{H}}^n$ define $\check{D} = \boldsymbol{e}^\top \boldsymbol{y} + \boldsymbol{r}_y \check{\boldsymbol{v}} + \boldsymbol{s}_y \check{\boldsymbol{w}}$ **else if** $\boldsymbol{y} \in \mathbb{Z}_p^n$ define $\check{D} = \check{\boldsymbol{u}} \boldsymbol{y} + \boldsymbol{r}_y \check{\boldsymbol{v}}$

Set $\alpha = \beta = \gamma = \delta = 0$

if $L = \mathrm{PPE}$ $\alpha, \beta, \gamma, \delta \leftarrow\!\$\, \mathbb{Z}_p$

if $L = \mathrm{ME}_{\hat{\mathbb{G}}}$ $\alpha, \beta \leftarrow\!\$\, \mathbb{Z}_p$

if $L = \mathrm{ME}_{\check{\mathbb{H}}}$ $\alpha, \gamma \leftarrow\!\$\, \mathbb{Z}_p$

if $L = \mathrm{QE}$ $\alpha \leftarrow\!\$\, \mathbb{Z}_p$

$\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} = \boldsymbol{r}_x \Gamma \check{D} + \alpha \check{\boldsymbol{v}} + \beta \check{\boldsymbol{w}}$ $\quad$ $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} = (\hat{C} - \hat{\boldsymbol{v}} \boldsymbol{r}_x - \hat{\boldsymbol{w}} \boldsymbol{s}_x) \Gamma \boldsymbol{r}_y - \hat{\boldsymbol{v}} \alpha - \hat{\boldsymbol{w}} \gamma$

$\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} = \boldsymbol{s}_x \Gamma \check{D} + \gamma \check{\boldsymbol{v}} + \delta \check{\boldsymbol{w}}$ $\quad$ $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} = (\hat{C} - \hat{\boldsymbol{v}} \boldsymbol{r}_x - \hat{\boldsymbol{w}} \boldsymbol{s}_x) \Gamma \boldsymbol{s}_y - \hat{\boldsymbol{v}} \beta - \hat{\boldsymbol{w}} \delta$

return $\boldsymbol{\pi} = (\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$

Fig. 15. Prover algorithm of the GS proof system.

$\mathsf{PVfy}(L, \Gamma, \{(l_{x_i}, \hat{\boldsymbol{c}}_i)\}_{i=1}^m, \{(l_{y_j}, \check{\boldsymbol{d}}_j)\}_{j=1}^n), \boldsymbol{\pi}, t)$

Check that the equation has a valid format.

Check $\hat{C} = (\hat{\boldsymbol{c}}_1 \dots \hat{\boldsymbol{c}}_m) \in \hat{\mathbb{G}}^{2\times m}$ and $\check{D} = (\check{\boldsymbol{d}}_1 \dots \check{\boldsymbol{d}}_n)^\top \in \check{\mathbb{H}}^{n\times 2}$

Check $\boldsymbol{\pi} = (\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}) \in \check{\mathbb{H}}^{2\times 1} \times \check{\mathbb{H}}^{2\times 1} \times \hat{\mathbb{G}}^{1\times 2} \times \hat{\mathbb{G}}^{1\times 2}$

if $L = \text{PPE}$ Check that $t \in \mathbb{T}$ and compute $\boldsymbol{l}_{\mathbb{T}} = \boldsymbol{e}^\top t \boldsymbol{e}$

if $L = \text{ME}_{\hat{\mathbb{G}}}$ Check that $t \in \hat{\mathbb{G}}$ and compute $\boldsymbol{l}_{\mathbb{T}} = \boldsymbol{e}^\top t \check{\boldsymbol{u}}$

if $L = \text{ME}_{\check{\mathbb{H}}}$ Check that $t \in \check{\mathbb{H}}$ and compute $\boldsymbol{l}_{\mathbb{T}} = \hat{\boldsymbol{u}} t \boldsymbol{e}$

if $L = \text{QE}$ Check that $t \in \mathbb{Z}_p$ and compute $\boldsymbol{l}_{\mathbb{T}} = \hat{\boldsymbol{u}} t \check{\boldsymbol{u}}$

Check $\hat{C}\Gamma\check{D} = \hat{\boldsymbol{v}}\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} + \hat{\boldsymbol{w}}\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} + \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}\check{\boldsymbol{v}} + \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}\check{\boldsymbol{w}} + \boldsymbol{l}_{\mathbb{T}}$

return 1 if and only if all checks pass and 0 otherwise.

Fig. 16. Verifier algorithm of the GS proof system.

$(\hat{C}', \check{D}', \pi') \leftarrow \mathsf{RandPr}(L, \Gamma, \{(l_{x_i}, \hat{\boldsymbol{c}}_i)\}_{i=1}^m, \{(l_{y_j}, \check{\boldsymbol{d}}_j)\}_{j=1}^n, \boldsymbol{\pi}; \boldsymbol{r})$

Parse $\boldsymbol{r} = (\boldsymbol{r}_x, \boldsymbol{s}_x, \boldsymbol{r}_y, \boldsymbol{s}_y)$

Define $\hat{C} = (\hat{\boldsymbol{c}}_1 \dots \hat{\boldsymbol{c}}_m) \in \hat{\mathbb{G}}^{2\times m}, \check{D} = (\check{\boldsymbol{d}}_1 \dots \check{\boldsymbol{d}}_n)^\top \in \check{\mathbb{H}}^{n\times 2}$ and parse $\boldsymbol{\pi} = (\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$

if $\boldsymbol{x} \in \hat{\mathbb{G}}^m$ define $\hat{C}' = \hat{C} + \hat{\boldsymbol{v}}\boldsymbol{r}_x + \hat{\boldsymbol{w}}\boldsymbol{s}_x$ **else if** $\boldsymbol{x} \in \mathbb{Z}_p^m$ define $\hat{C}' = \hat{C} + \hat{\boldsymbol{v}}\boldsymbol{r}_x$

if $\boldsymbol{y} \in \check{\mathbb{H}}^n$ define $\check{D}' = \check{D} + \boldsymbol{r}_y\check{\boldsymbol{v}} + \boldsymbol{s}_y\check{\boldsymbol{w}}$ **else if** $\boldsymbol{y} \in \mathbb{Z}_p^n$ define $\check{D}' = \check{D} + \boldsymbol{r}_y\check{\boldsymbol{v}}$

if $L = \text{PPE}$ $\alpha, \beta, \gamma, \delta \leftarrow\!\!\$\ \mathbb{Z}_p$

if $L = \text{ME}_{\hat{\mathbb{G}}}$ $\alpha, \beta \leftarrow\!\!\$\ \mathbb{Z}_p$

if $L = \text{ME}_{\check{\mathbb{H}}}$ $\alpha, \gamma \leftarrow\!\!\$\ \mathbb{Z}_p$

if $L = \text{QE}$ $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$

$\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}{}' = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} + \boldsymbol{r}_x\Gamma\check{D}' + \alpha\check{\boldsymbol{v}} + \beta\check{\boldsymbol{w}} \qquad \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}{}' = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} + \hat{C}\Gamma\boldsymbol{r}_y - \hat{\boldsymbol{v}}\alpha - \hat{\boldsymbol{w}}\gamma$

$\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}{}' = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} + \boldsymbol{s}_x\Gamma\check{D}' + \gamma\check{\boldsymbol{v}} + \delta\check{\boldsymbol{w}} \qquad \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}{}' = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} + \hat{C}\Gamma\boldsymbol{s}_y - \hat{\boldsymbol{v}}\beta - \hat{\boldsymbol{w}}\delta$

return $(\hat{C}', \check{D}', \pi' = (\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}{}', \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}{}', \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}{}', \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}{}'))$

Fig. 17. Proof re-randomization algorithm of the GS proof system.

B Details on Our Extendable Shuffle

We build our extendable shuffle argument building upon the shuffle argument of [36], which is based on Groth-Sahai proofs. In particular, we commit to elements $\hat{e}_i$ and to the bits bit_i using as a commitment scheme the ElGamal encryption in $\hat{\mathbb{G}}$ and $\check{\mathbb{H}}$ respectively. In the following, given a vector $\boldsymbol{v}$ we indicate is i-th component as $\boldsymbol{v}^i$. Additionally, we use the following notation to refer to the GS commitment to a variable v. If the GS commitment to v is in group $\hat{\mathbb{G}}$ we write $\hat{\boldsymbol{c}}_v$. If the GS commitment to v is in group $\check{\mathbb{H}}$ we write $\check{\boldsymbol{d}}_v$. We use the same convention for the randomness r and s used to generate the GS commitment. The algorithms of our extendable shuffle argument are the following:

– $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk}, N)$: run $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{GS.Ext}_0(\mathsf{gk})$. Sample $p_1, \ldots, p_N \leftarrow\$ \mathbb{Z}_p$, compute sets $\{\hat{g}_i\} = \{\hat{g}p_i\}, \{\hat{\gamma}_i\} = \{\hat{g}2p_i\}, \{\check{h}_i\} = \{p_i\check{h}\}, \{\check{\delta}_i\} = \{2p_i\check{h}\}$. Set $\mathsf{crs}_2 = (\{\hat{g}_i\}, \{\hat{\gamma}_i\}, \{\check{h}_i\}, \{\check{\delta}_i\})$. Return $(\mathsf{crs} = (\mathsf{crs}_1, \mathsf{crs}_2), \mathsf{td} = \mathsf{xk}_1)$.
– $\mathsf{ck} = \mathsf{ComSetup}(\mathsf{gk})$: sample $\hat{\mathsf{ck}} \leftarrow\$ \hat{\mathbb{G}}$ and $\check{\mathsf{ck}} \leftarrow\$ \check{\mathbb{H}}$. Return $\mathsf{ck} = (\hat{\mathsf{ck}}, \check{\mathsf{ck}})$.
– $\hat{\boldsymbol{z}} \leftarrow \mathsf{Com}_{\hat{ck}}(\hat{e}; r)$: return $\hat{\boldsymbol{z}} = (\hat{g}r, \hat{e} + \hat{\mathsf{ck}}r)$.
– $\check{\boldsymbol{m}} \leftarrow \mathsf{Com}_{\check{ck}}(\mathsf{bit}; r)$: Return $\check{\boldsymbol{m}} = (r\check{h}, \mathsf{bit}\check{h} + r\check{\mathsf{ck}})$.
– $(\Pi, \mathsf{aux}_1, \ldots, \mathsf{aux}_n) \leftarrow \mathsf{ShPrv}(\mathsf{crs}, \mathsf{ck}, x, w)$: See Fig. 18 for a detailed description of the prover algorithm.
– $0/1 \leftarrow \mathsf{ShVerify}(\mathsf{crs}, \mathsf{ck}, x, \Pi)$: reconstruct equations as shown in Fig. 18, appropriately parse Π, and for every equation run $\mathsf{GS.PVfy}$ with the obvious inputs.
– $(\Pi', x', \mathsf{aux}_{n+1}, \boldsymbol{r}) \leftarrow \mathsf{ShExtend}(\mathsf{crs}, \mathsf{ck}, x, \hat{\boldsymbol{z}}_{\boldsymbol{n+1}}, r_{\check{m}_{n+1}}, \Pi)$:
  1. Parse Π in terms of its commitments and proof elements lists.
  2. Parse x as $(k, \hat{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}_{\boldsymbol{n}}, \check{\boldsymbol{m}}_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}_{\boldsymbol{n}}, \hat{e}_1, \ldots, \hat{e}_k)$.
  3. For each of the equation types $\mathcal{D}^1, \mathcal{D}^2, \mathcal{F}, \mathcal{B}, \mathcal{Q}^1, \mathcal{Q}^2, \mathcal{V}^1, \mathcal{V}^2$, add a new equation by defining the corresponding new independent variables, $r_{\hat{z}_{n+1}} = 0, f_{n+1} = 0, \mathsf{bit}_{n+1} = 0, \check{a'}_{n+1} = \check{a}_{n+1} = \check{0}, \check{b'}_{n+1} = \check{b}_{n+1} = \check{0}, \hat{c}_{n+1} = \hat{0}, \hat{E}_{n+1} = \hat{0}$. The value of the variable $r_{\check{m}_{n+1}}$ is set as the one given in input.
  4. Compute $\check{\boldsymbol{m}}_{\boldsymbol{n+1}} = (r_{\check{m}_{n+1}}\check{h}, \mathsf{bit}_{n+1}\check{h} + r_{\check{m}_{n+1}}\check{\mathsf{ck}})$.
  5. Compute GS commitments to new variables and appropriately add them to the corresponding commitment lists.
  6. Compute the related new GS proofs and add them to the corresponding proof element lists.
  7. Compute auxiliary value $\mathsf{aux}_{n+1} = (\mathsf{aux}_{\mathcal{K}}^{n+1}, \mathsf{aux}_{\mathcal{S}^1}^{n+1}, \mathsf{aux}_{\mathcal{S}^2}^{n+1}, \mathsf{aux}_{\mathcal{E}^1}^{n+1}, \mathsf{aux}_{\mathcal{E}^2}^{n+1})$
     - $\mathsf{aux}_{\mathcal{K}}^{n+1} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}})$ with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}} = (r_{\mathsf{bit}_{n+1}}\check{\boldsymbol{u}}^1, r_{\mathsf{bit}_{n+1}}\check{\boldsymbol{u}}^2)$.
     - $\mathsf{aux}_{\mathcal{S}^1}^{n+1} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (r_{\check{a}_{n+1}}\hat{\boldsymbol{u}}^1, r_{\check{a}_{n+1}}\hat{\boldsymbol{u}}^2), \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (s_{\check{a}_{n+1}}\hat{\boldsymbol{u}}^1, s_{\check{a}_{n+1}}\hat{\boldsymbol{u}}^2)$.
     - $\mathsf{aux}_{\mathcal{S}^2}^{n+1} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (r_{\check{b}_{n+1}}\hat{\boldsymbol{u}}^1, r_{\check{b}_{n+1}}\hat{\boldsymbol{u}}^2), \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (s_{\check{b}_{n+1}}\hat{\boldsymbol{u}}^1, s_{\check{b}_{n+1}}\hat{\boldsymbol{u}}^2)$.
     - $\mathsf{aux}_{\mathcal{E}^1}^{n+1} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}} = (r_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{a}}_{\boldsymbol{n+1}}}, r_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{a}}_{\boldsymbol{n+1}}}), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (\hat{0}, \hat{0}), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}} = (\hat{0}, \hat{0}), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (s_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{a}}_{\boldsymbol{n+1}}}, s_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{a}}_{\boldsymbol{n+1}}})$.
     - $\mathsf{aux}_{\mathcal{E}^2}^{n+1} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}} = (r_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{b}}_{\boldsymbol{n+1}}}, r_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{b}}_{\boldsymbol{n+1}}}), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (\hat{0}, \hat{0}), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}} = (\hat{0}, \hat{0}), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (s_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{b}}_{\boldsymbol{n+1}}}, s_{\hat{E}_{n+1}}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{b}}_{\boldsymbol{n+1}}})$.
  8. Parse $\pi_{\mathcal{K}}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$ and update $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}$ as $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} + \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}$, with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}$ taken from $\mathsf{aux}_{\mathcal{K}}^{n+1}$.
  9. Parse $\pi_{\mathcal{S}^1}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$ and update $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}$, $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ π with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ taken from $\mathsf{aux}_{\mathcal{S}^1}^{n+1}$.
  10. Parse $\pi_{\mathcal{S}^2}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$ and update $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}$, $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ π with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ taken from $\mathsf{aux}_{\mathcal{S}^2}^{n+1}$.
  11. Parse $\pi_{\mathcal{E}^1}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$ and update $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} + \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} + \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ taken from $\mathsf{aux}_{\mathcal{E}^1}^{n+1}$.
  12. Parse $\pi_{\mathcal{E}^2}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}})$ and update $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}} + \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}} + \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}}$ taken from $\mathsf{aux}_{\mathcal{E}^2}^{n+1}$.
  13. Randomize the statement and adjust the related GS commitments as follows for all $i \in [n+1]$:
     - Compute $\hat{\boldsymbol{z}}'_{\boldsymbol{i}} = \hat{\boldsymbol{z}}_{\boldsymbol{i}} + (\hat{g}r'_{\hat{z}_i}, \hat{\mathsf{ck}}r'_{\hat{z}_i})$ with uniformly sampled $r'_{\hat{z}_i}$ (i.e, obtain a commitment to the same value with randomness $r_{\hat{z}_i} + r'_{\hat{z}_i}$).
     - Compute $\check{\boldsymbol{m}}'_{\boldsymbol{i}} = \check{\boldsymbol{m}}_{\boldsymbol{i}} + (r'_{\check{m}_i}\check{h}, r'_{\check{m}_i}\check{\mathsf{ck}})$ with uniformly sampled $r'_{\check{m}_i}$.
     - Adjust GS commitments $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}$ to variables $r_{\hat{z}_i}$ as $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}} = \check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}} + \hat{\boldsymbol{u}}r'_{\hat{z}_i}$ (i.e, obtain a GS commitment with the same randomness but to value $r_{\hat{z}_i} + r'_{\hat{z}_i}$).
     - Adjust GS commitments $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\check{\boldsymbol{m}}_{\boldsymbol{i}}}}$ to variables $r_{\check{m}_i}$ as $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\check{\boldsymbol{m}}_{\boldsymbol{i}}}}} = \check{\boldsymbol{d}}_{\boldsymbol{r}_{\check{\boldsymbol{m}}_{\boldsymbol{i}}}}} + \hat{\boldsymbol{u}}r'_{\check{m}_i}$.

     Set $x' = (k, \hat{\boldsymbol{z}}'_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}'_{\boldsymbol{n+1}}, \check{\boldsymbol{m}}'_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}'_{\boldsymbol{n+1}}, \hat{e}_1, \ldots, \hat{e}_k)$.
  14. Run $\mathsf{GS.RandPr}$ on each of the proofs, appropriately fixing the random coins when re-randomizing proofs related to equations involving shared variables (i.e., s.t. we end up again with shared variables having the exact same commitments). Let $\boldsymbol{r} = (r_1, \ldots, r_{n+1})$ with $r_i = (r_{\check{a}_i}, s_{\check{a}_i}, r_{\check{b}_i}, s_{\check{b}_i}, r_{\hat{E}_i}, s_{\hat{E}_i}, r_{\mathsf{bit}_i})$. Namely, r_i contains all the randomnesses used to re-randomize the GS commitments to the corresponding i-th variables. Let randomized proof elements and commitments be contained in Π'.

15. Output $(\Pi', x', \mathsf{aux}_{n+1}, \boldsymbol{r})$.

– $(\Pi', x', \mathsf{aux}'_\alpha, \boldsymbol{r}) \leftarrow \mathsf{ShAdd}(\mathsf{crs}, \mathsf{ck}, x, r_{\hat{z}_\alpha}, \alpha, \hat{e}_{k+1}, \mathsf{aux}_\alpha, \Pi)$:

1. Parse Π in terms of its commitments and proof elements lists.
2. Parse x as $(k, \hat{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}_{\boldsymbol{n}}, \check{\boldsymbol{m}}_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}_{\boldsymbol{n}}, \hat{e}_1, \ldots, \hat{e}_k)$.
3. Replace $\check{\boldsymbol{m}}_{\boldsymbol{\alpha}}$ with a freshly computed $\check{\boldsymbol{m}}_{\boldsymbol{\alpha}} = (r_{\check{m}_\alpha}\check{h}, \mathsf{bit}_\alpha\check{h} + r_{\check{m}_\alpha}\check{\mathsf{ck}})$ with $\mathsf{bit}_\alpha = 1$ and uniformly sampled $r_{\check{m}_\alpha}$.
4. Replace $\hat{\boldsymbol{z}}_{\boldsymbol{\alpha}}$ with a freshly computed $\hat{\boldsymbol{z}}_{\boldsymbol{\alpha}} = (\hat{g}r_{\hat{z}_\alpha}, \hat{e} + \hat{\mathsf{ck}}r_{\hat{z}_\alpha})$.
5. For each of the equation types $\mathcal{D}^1, \mathcal{D}^2, \mathcal{F}, \mathcal{B}, \mathcal{Q}^1, \mathcal{Q}^2, \mathcal{V}^1, \mathcal{V}^2$, replace the variables in equations related to position α (i.e, $\mathcal{D}^1_{\cdot,\alpha}, \mathcal{D}^2_{\cdot,\alpha}, \mathcal{F}_\alpha, \mathcal{B}_\alpha, \mathcal{Q}^1_\alpha, \mathcal{Q}^2_\alpha, \mathcal{V}^1_\alpha, \mathcal{V}^2_\alpha$) as follows: $f_\alpha = 1, \mathsf{bit}_\alpha = 1, \check{a}_\alpha = \check{a'}_\alpha = \check{h}_{k+1}, \check{b}_\alpha = \check{b'}_\alpha = \check{\delta}_{k+1}, \hat{c}_\alpha = \hat{g}_{k+1}, \hat{E}_\alpha = \hat{e}_{k+1}$. The value of the variable $r_{\hat{z}_\alpha}$ is set as the one taken in input. The value of the variable $r_{\check{m}_\alpha}$ is set as the one sampled at step 3.
6. Replace the GS commitments related to equations $\mathcal{D}^1_{\cdot,\alpha}, \mathcal{D}^2_{\cdot,\alpha}, \mathcal{F}_\alpha, \mathcal{B}_\alpha, \mathcal{Q}^1_\alpha, \mathcal{Q}^2_\alpha, \mathcal{V}^1_\alpha, \mathcal{V}^2_\alpha$ with freshly generated ones updating the corresponding commitment lists accordingly.
7. Replace the GS proof elements related to equations $\mathcal{D}^1_{\cdot,\alpha}, \mathcal{D}^2_{\cdot,\alpha}, \mathcal{F}_\alpha, \mathcal{B}_\alpha, \mathcal{V}^1_\alpha, \mathcal{V}^2_\alpha$ with freshly generated ones updating the corresponding proof element lists accordingly.
8. Parse $\mathsf{aux}_\alpha = (\mathsf{aux}^\alpha_{\mathcal{K}}, \mathsf{aux}^\alpha_{\mathcal{S}^1}, \mathsf{aux}^\alpha_{\mathcal{S}^2}, \mathsf{aux}^\alpha_{\mathcal{E}^1}, \mathsf{aux}^\alpha_{\mathcal{E}^2})$
9. Compute $\mathsf{aux}'_\alpha = (\mathsf{aux}^{\alpha'}_{\mathcal{K}}, \mathsf{aux}^{\alpha'}_{\mathcal{S}^1}, \mathsf{aux}^{\alpha'}_{\mathcal{S}^2}, \mathsf{aux}^{\alpha'}_{\mathcal{E}^1}, \mathsf{aux}^{\alpha'}_{\mathcal{E}^2})$
   - $\mathsf{aux}^{\alpha'}_{\mathcal{K}} = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}})$ with $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}} = (r_{\mathsf{bit}_i}\check{\boldsymbol{u}}^1, r_{\mathsf{bit}_i}\check{\boldsymbol{u}}^2)$.
   - $\mathsf{aux}^{\alpha'}_{\mathcal{S}^1} = (\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}})$ with $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} = (r_{\check{a}_i}\hat{\boldsymbol{u}}^1, r_{\check{a}_i}\hat{\boldsymbol{u}}^2), \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} = (s_{\check{a}_i}\hat{\boldsymbol{u}}^1, s_{\check{a}_i}\hat{\boldsymbol{u}}^2)$.
   - $\mathsf{aux}^{\alpha'}_{\mathcal{S}^2} = (\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}})$ with $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} = (r_{\check{b}_i}\hat{\boldsymbol{u}}^1, r_{\check{b}_i}\hat{\boldsymbol{u}}^2), \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} = (s_{\check{b}_i}\hat{\boldsymbol{u}}^1, s_{\check{b}_i}\hat{\boldsymbol{u}}^2)$.
   - $\mathsf{aux}^{\alpha'}_{\mathcal{E}^1} = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}})$ with $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}} = (r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{a}}_{\boldsymbol{i}}}, r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{a}}_{\boldsymbol{i}}}), \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} = (0, r_{\check{a}_i}\hat{E}_i), \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}} = (0, s_{\check{a}_i}\hat{E}_i), \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} = (s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{a}}_{\boldsymbol{i}}}, s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{a}}_{\boldsymbol{i}}})$.
   - $\mathsf{aux}^{\alpha'}_{\mathcal{E}^2} = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}})$ with $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}} = (r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{b}}_{\boldsymbol{i}}}, r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{b}}_{\boldsymbol{i}}}), \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} = (0, r_{\check{b}_i}\hat{E}_i), \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}} = (0, s_{\check{b}_i}\hat{E}_i), \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} = (s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{b}}_{\boldsymbol{i}}}, s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{b}}_{\boldsymbol{i}}})$.
10. Parse $\pi_{\mathcal{K}}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}})$ and update $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}$ as $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}$, with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}$ taken from $\mathsf{aux}^i_{\mathcal{K}}$ and $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}$ taken from $\mathsf{aux}^{\alpha'}_{\mathcal{K}}$.
11. Parse $\pi_{\mathcal{S}^1}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}})$ and update $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$, with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^i_{\mathcal{S}^1}$ and $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^{\alpha'}_{\mathcal{S}^1}$.
12. Parse $\pi_{\mathcal{S}^2}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}})$ and update $\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$, with $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^i_{\mathcal{S}^2}$ and $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^{\alpha'}_{\mathcal{S}^2}$.
13. Parse $\pi_{\mathcal{E}^1}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}})$ and update $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}} - \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}} + \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^i_{\mathcal{E}^1}$ and $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^{\alpha'}_{\mathcal{E}^1}$.
14. Parse $\pi_{\mathcal{E}^2}$ as $(\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}})$ and update $\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}} = \check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}} - \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}} + \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} = \hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}} - \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}} + \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^i_{\mathcal{E}^2}$ and $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\upsilon}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{\omega}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{\omega}}}}$ taken from $\mathsf{aux}^{\alpha'}_{\mathcal{E}^2}$.
15. Randomize the statement and adjust the related GS commitments as follows for all $i \in [n+1]$:
   - Compute $\hat{\boldsymbol{z}}'_{\boldsymbol{i}} = \hat{\boldsymbol{z}}_{\boldsymbol{i}} + (\hat{g}r'_{\hat{z}_i}, \hat{\mathsf{ck}}r'_{\hat{z}_i})$ with uniformly sampled $r'_{\hat{z}_i}$ (i.e, obtain a commitment to the same value with randomness $r_{\hat{z}_i} + r'_{\hat{z}_i}$).
   - Compute $\check{\boldsymbol{m}}'_{\boldsymbol{i}} = \check{\boldsymbol{m}}_{\boldsymbol{i}} + (r'_{\check{m}_i}\check{h}, r'_{\check{m}_i}\check{\mathsf{ck}})$ with uniformly sampled $r'_{\check{m}_i}$.
   - Adjust GS commitments $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}$ to variables $r_{\hat{z}_i}$ as $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}} = \check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}} + \hat{\boldsymbol{u}}r'_{\hat{z}_i}$ (i.e, obtain a GS commitment with the same randomness but to value $r_{\hat{z}_i} + r'_{\hat{z}_i}$).
   - Adjust GS commitments $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\check{\boldsymbol{m}}_{\boldsymbol{i}}}}$ to variables $r_{\check{m}_i}$ as $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\check{\boldsymbol{m}}_{\boldsymbol{i}}}}} = \check{\boldsymbol{d}}_{\boldsymbol{r}_{\check{\boldsymbol{m}}_{\boldsymbol{i}}}}} + \hat{\boldsymbol{u}}r'_{\check{m}_i}$.

   Set $x' = (k+1, \hat{\boldsymbol{z}}'_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}'_{\boldsymbol{n}}, \check{\boldsymbol{m}}'_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}'_{\boldsymbol{n}}, \hat{e}_1, \ldots, \hat{e}_{k+1})$.
16. Run $\mathsf{GS.RandPr}$ on each of the proofs, appropriately fixing the random coins when re-randomizing proofs related to equations involving shared variables (i.e., s.t. we end up again with shared variables having the exact same commitments). Let $\boldsymbol{r} = (r_1, \ldots, r_n)$ with $r_i = (r_{\check{a}_i}, s_{\check{a}_i}, r_{\check{b}_i}, s_{\check{b}_i}, r_{\hat{E}_i}, s_{\hat{E}_i}, r_{\mathsf{bit}_i})$. Namely, r_i contains all the randomnesses used to re-randomize the commitments to the corresponding i-th variables. Let randomized proof elements and commitments be contained in Π'.

17. Output $(\Pi', x', \mathsf{aux}'_i, \boldsymbol{r})$.

– $\mathsf{aux}'_i \leftarrow \mathsf{ShAuxUpd}(\mathsf{crs}, x, \Pi, \mathsf{aux}_i, r_i)$: Parse $r_i = (r_{\check{a}_i}, s_{\check{a}_i}, r_{\check{b}_i}, s_{\check{b}_i}, r_{\hat{E}_i}, s_{\hat{E}_i}, r_{\mathsf{bit}_i})$. Parse Π to get commitments $\check{\boldsymbol{d}}_{\check{\boldsymbol{a}}_i}, \check{\boldsymbol{d}}_{\check{\boldsymbol{b}}_i}, \hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_i}, \hat{\boldsymbol{c}}_{\mathsf{bit}_i}$. Set $\check{\boldsymbol{d}}'_{\check{\boldsymbol{a}}_i} = \check{\boldsymbol{d}}_{\check{\boldsymbol{a}}_i} + r_{\check{a}_i}\check{\boldsymbol{v}} + s_{\check{a}_i}\check{\boldsymbol{w}}, \check{\boldsymbol{d}}'_{\check{\boldsymbol{b}}_i} = \check{\boldsymbol{d}}_{\check{\boldsymbol{b}}_i} + r_{\check{b}_i}\check{\boldsymbol{v}} + s_{\check{b}_i}\check{\boldsymbol{w}}$. Then run the same procedure of $\mathsf{GS.RandPr}$ but component-wise.

1. Parse $\mathsf{aux}_i = (\mathsf{aux}^i_{\mathcal{K}}, \mathsf{aux}^i_{\mathcal{S}^1}, \mathsf{aux}^i_{\mathcal{S}^2}, \mathsf{aux}^i_{\mathcal{E}^1}, \mathsf{aux}^i_{\mathcal{E}^2})$.
2. Parse $\mathsf{aux}^i_{\mathcal{K}} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}})$ and compute $\mathsf{aux}_{\mathcal{K}} i' = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}})$ with $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + r_{\mathsf{bit}_i}\check{\boldsymbol{u}}$.
3. Parse $\mathsf{aux}^i_{\mathcal{S}^1} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and compute $\mathsf{aux}^{i'}_{\mathcal{S}^1} = (\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ with
   - $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}} + \hat{\boldsymbol{u}} r_{\check{a}_i}$.
   - $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} + \hat{\boldsymbol{u}} s_{\check{a}_i}$.
4. Parse $\mathsf{aux}^i_{\mathcal{S}^2} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and compute $\mathsf{aux}^{i'}_{\mathcal{S}^2} = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ with
   - $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + \hat{\boldsymbol{u}} r_{\check{b}_i}$.
   - $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} + \hat{\boldsymbol{u}} s_{\check{b}_i}$.
5. Parse $\mathsf{aux}^i_{\mathcal{E}^1} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and compute $\mathsf{aux}^{i'}_{\mathcal{E}^1} = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ with
   - $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + r_{\hat{E}_i}\check{\boldsymbol{d}}'_{\check{\boldsymbol{a}}_i}$.
   - $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + s_{\hat{E}_i}\check{\boldsymbol{d}}'_{\check{\boldsymbol{a}}_i}$.
   - $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}} + \hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_i} r_{\check{a}_i}$.
   - $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} + \hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_i} s_{\check{a}_i}$.
6. Parse $\mathsf{aux}^i_{\mathcal{E}^2} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and compute $\mathsf{aux}^{i'}_{\mathcal{E}^2} = (\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}, \mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ with
   - $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + r_{\hat{E}_i}\check{\boldsymbol{d}}'_{\check{\boldsymbol{b}}_i}$.
   - $\mathsf{aux}'_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} = \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + s_{\hat{E}_i}\check{\boldsymbol{d}}'_{\check{\boldsymbol{b}}_i}$.
   - $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}} + \hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_i} r_{\check{b}_i}$.
   - $\mathsf{aux}'_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}} + \hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_i} s_{\check{b}_i}$.

Output $\mathsf{aux}'_i = (\mathsf{aux}^{i'}_{\mathcal{K}}, \mathsf{aux}^{i'}_{\mathcal{S}^1}, \mathsf{aux}^{i'}_{\mathcal{S}^2}, \mathsf{aux}^{i'}_{\mathcal{E}^1}, \mathsf{aux}^{i'}_{\mathcal{E}^2})$.

– $0/1 \leftarrow \mathsf{ShAuxVerify}(\mathsf{crs}, x, w, \mathsf{aux}_1, \ldots, \mathsf{aux}_n, \Pi)$: Parse the proof Π in terms of its commitments and proof elements lists and parse each auxiliary value $\mathsf{aux}_i = (\mathsf{aux}^i_{\mathcal{K}}, \mathsf{aux}^i_{\mathcal{S}^1}, \mathsf{aux}^i_{\mathcal{S}^2}, \mathsf{aux}^i_{\mathcal{E}^1}, \mathsf{aux}^i_{\mathcal{E}^2})$. For all $i \in [n]$ set the variables $\mathsf{bit}_i, \check{a}_i, \check{b}_i, \hat{c}_i, \hat{E}_i$ as follows:

- Set $\mathcal{I} \leftarrow \emptyset$
- For all $j \in [k]$ let $\phi(j) = i$, set $\mathsf{bit}_i = 1, \check{a}_i = \check{h}_j, \check{b}_i = \check{\delta}_j, \hat{E}_i = \hat{e}_j, \mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$.
- For all $i \in [n] \setminus \mathcal{I}$, set $\mathsf{bit}_i = 0, \check{a}_i = \check{0}, \check{b}_i = \check{0}, \hat{E}_i = \hat{0}, \hat{c}_i = \hat{0}$.

Do the following checks, where all the $\check{\boldsymbol{d}}_{\check{\boldsymbol{a}}_i}, \check{\boldsymbol{d}}_{\check{\boldsymbol{b}}_i}, \hat{\boldsymbol{c}}_{\mathsf{bit}_i}$ are taken from the commitment lists in Π (i.e, run the verification of the GS proof system but for individual variables).

1. Equation $\mathcal{K}$: for all $i \in [n]$ parse $\mathsf{aux}^i_{\mathcal{K}} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}})$ and check that $\hat{\boldsymbol{c}}_{\mathsf{bit}_i}\check{\boldsymbol{u}} = \hat{\boldsymbol{v}}\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + \mathsf{bit}_i\hat{\boldsymbol{u}}\check{\boldsymbol{u}}$.
2. Equation $\mathcal{S}^1$: for all $i \in [n]$ parse $\mathsf{aux}^i_{\mathcal{S}^1} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and check that $\hat{\boldsymbol{u}}\check{\boldsymbol{d}}_{\check{\boldsymbol{a}}_i} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}\check{\boldsymbol{v}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}\check{\boldsymbol{w}} + (0_{\mathbb{T}}, \hat{\boldsymbol{u}}\check{a}_i)$.
3. Equation $\mathcal{S}^2$: for all $i \in [n]$ parse $\mathsf{aux}^i_{\mathcal{S}^2} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{w}}}})$ and check that $\hat{\boldsymbol{u}}\check{\boldsymbol{d}}_{\check{\boldsymbol{b}}_i} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}\check{\boldsymbol{v}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}\check{\boldsymbol{w}} + (0_{\mathbb{T}}, \hat{\boldsymbol{u}}\check{b}_i)$.
4. Equation $\mathcal{E}^1$: for all $i \in [n]$ parse $\mathsf{aux}^i_{\mathcal{E}^1} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and check that $\hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_i}\check{\boldsymbol{d}}_{\check{\boldsymbol{a}}_i} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}\check{\boldsymbol{v}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}\check{\boldsymbol{w}} + \hat{\boldsymbol{v}}\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + \hat{\boldsymbol{w}}\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{w}}}} + (0_{\mathbb{T}}, \hat{E}_i \cdot \check{a}_i)$.
5. Equation $\mathcal{E}^2$: for all $i \in [n]$ parse $\mathsf{aux}^i_{\mathcal{E}^2} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}})$ and check that $\hat{\boldsymbol{c}}_{\hat{\boldsymbol{E}}_2}\check{\boldsymbol{d}}_{\check{\boldsymbol{b}}_i} = \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{d}}}}\check{\boldsymbol{v}} + \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\mathfrak{w}}}}\check{\boldsymbol{w}} + \hat{\boldsymbol{v}}\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{d}}}} + \hat{\boldsymbol{w}}\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\mathfrak{w}}}} + (0_{\mathbb{T}}, \hat{E}_i \cdot \check{b}_i)$.

– $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(\mathsf{gk}, N)$: run $(\mathsf{crs}_1, \mathsf{td}_1) \leftarrow \mathsf{GS.Sim}_0(\mathsf{gk})$. Sample $p_1, \ldots, p_N \leftarrow\!\!\$\ \mathbb{Z}_p$, and set $\mathsf{td}_2 = (p_1, \ldots, p_n)$. Compute sets $\{\hat{g}_i\} = \{\hat{g}p_i\}, \{\hat{\gamma}_i\} = \{\hat{g}2p_i\}, \{\check{h}_i\} = \{p_i\check{h}\}, \{\check{\delta}_i\} = \{2p_i\check{\delta}\}$. Set $\mathsf{crs}_2 = (\{\hat{g}_i\}, \{\hat{\gamma}_i\}, \{\check{h}_i\}, \{\check{\delta}_i\})$. Return $(\mathsf{crs} = (\mathsf{crs}_1, \mathsf{crs}_2), \mathsf{td} = (\mathsf{td}_1, \mathsf{td}_2))$.

– $(\Pi, \mathsf{aux}_1, \ldots, \mathsf{aux}_n) \leftarrow \mathsf{Sim}_1(\mathsf{crs}, \mathsf{td}, x)$: The description of this algorithm is reported in Fig. 19.

– The transformation $\mathsf{AddT} = (\mathsf{AddT}_x, \mathsf{AddT}_w)$ works as follows:

- $x' \leftarrow \mathsf{AddT}_x(\mathsf{ck}, x, \hat{e}_{k+1}, \alpha, r_{\hat{z}_\alpha}, r_{\check{m}_\alpha}; \boldsymbol{r}')$:
  * Parse x as $(k, \hat{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}_{\boldsymbol{n}}, \check{\boldsymbol{m}}_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}_{\boldsymbol{n}}, \hat{e}_1, \ldots, \hat{e}_k)$.
  * Parse $\boldsymbol{r}'$ as $(r'_{\hat{z}_1}, \ldots, r'_{\hat{z}_n}, r'_{\check{m}_1}, \ldots, r'_{\check{m}_n})$.

  - $*$ Replace $\check{\boldsymbol{m}}_{\boldsymbol{\alpha}}$ with a freshly computed $\check{\boldsymbol{m}}_{\boldsymbol{\alpha}} = (r_{\check{m}_\alpha}\check{h}, \mathsf{bit}_\alpha\check{h} + r_{\check{m}_\alpha}\check{\mathsf{ck}})$. with $\mathsf{bit}_\alpha = 1$.
  - $*$ Replace $\hat{\boldsymbol{z}}_{\boldsymbol{\alpha}}$ with a freshly computed $\hat{\boldsymbol{z}}_{\boldsymbol{\alpha}} = (\hat{g}r_{\hat{z}_\alpha}, \hat{e}_{k+1} + \hat{\mathsf{ck}}r_{\hat{z}_\alpha})$.
  - $*$ For all $i \in [n]$, compute $\hat{\boldsymbol{z}}'_{\boldsymbol{i}} = \hat{\boldsymbol{z}}_{\boldsymbol{i}} + (\hat{g}r'_{\hat{z}_i}, \hat{\mathsf{ck}}r'_{\hat{z}_i})$.
  - $*$ For all $i \in [n]$, compute $\check{\boldsymbol{m}}'_{\boldsymbol{i}} = \check{\boldsymbol{m}}_{\boldsymbol{i}} + (r'_{\check{m}_i}\check{h}, r'_{\check{m}_i}\check{\mathsf{ck}})$.
  - $*$ Set $x' = (k+1, \hat{\boldsymbol{z}}'_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}'_{\boldsymbol{n}}, \check{\boldsymbol{m}}'_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}'_{\boldsymbol{n}}, \hat{e}_1, \ldots, \hat{e}_{k+1})$.
- $w' \leftarrow \mathsf{AddT}_w(\mathsf{ck}, w, r_{\hat{z}_\alpha}, \alpha, r_{\check{m}_\alpha}; \boldsymbol{r}')$
  - $*$ Parse $w = (\phi, r_1, \ldots, r_k, r_{\check{m}_1}, \ldots, r_{\check{m}_n})$.
  - $*$ Parse $\boldsymbol{r}'$ as $(r'_{\hat{z}_1}, \ldots, r'_{\hat{z}_n}, r'_{\check{m}_1}, \ldots, r'_{\check{m}_n})$.
  - $*$ For all $i \in [n]$, compute $r^*_{\check{m}_i} = r_{\check{m}_i} + r'_{\check{m}_i}$.
  - $*$ For all $i \in [k]$, compute $r'_i = r_i + r'_{\hat{z}_{\phi(i)}}$.
  - $*$ Compute $r'_{k+1} = r_{\hat{z}_\alpha} + r'_{\hat{z}_\alpha}$.
  - $*$ Compute $\phi' = \phi \cup \{k+1 \to \alpha\}$.
  - $*$ Set $w' = (\phi', r'_1, \ldots, r'_{k+1}, r^*_{\check{m}_1}, \ldots, r^*_{\check{m}_n})$.

– The transformation $\mathsf{ExtT} = (\mathsf{ExtT}_x, \mathsf{ExtT}_w)$ works as follows:
- $x' \leftarrow \mathsf{ExtT}_x(\mathsf{ck}, x, \hat{\boldsymbol{z}}_{\boldsymbol{n+1}}, r_{\check{m}_{n+1}}; \boldsymbol{r}')$:
  - $*$ Parse x as $(k, \hat{\boldsymbol{z}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}_{\boldsymbol{n}}, \check{\boldsymbol{m}}_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}_{\boldsymbol{n}}, \hat{e}_1, \ldots, \hat{e}_k)$.
  - $*$ Parse $\boldsymbol{r}'$ as $(r'_{\hat{z}_1}, \ldots, r'_{\hat{z}_n}, r'_{\check{m}_1}, \ldots, r'_{\check{m}_{n+1}})$.
  - $*$ Compute $\check{\boldsymbol{m}}_{\boldsymbol{n+1}} = (r_{\check{m}_{n+1}}\check{h}, r_{\check{m}_{n+1}}\check{\mathsf{ck}})$.
  - $*$ Compute $\hat{\boldsymbol{z}}'_{\boldsymbol{i}} = \hat{\boldsymbol{z}}_{\boldsymbol{i}} + (\hat{g}r'_{\hat{z}_i}, \hat{\mathsf{ck}}r'_{\hat{z}_i})$.
  - $*$ Compute $\check{\boldsymbol{m}}'_{\boldsymbol{i}} = \check{\boldsymbol{m}}_{\boldsymbol{i}} + (r'_{\check{m}_i}\check{h}, r'_{\check{m}_i}\check{\mathsf{ck}})$.
  - $*$ Set $x' = (k, \hat{\boldsymbol{z}}'_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{z}}'_{\boldsymbol{n+1}}, \check{\boldsymbol{m}}'_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}'_{\boldsymbol{n+1}}, \hat{e}_1, \ldots, \hat{e}_k)$.
- $w' \leftarrow \mathsf{ExtT}_w(\mathsf{ck}, w, r_{\check{m}_{n+1}}; \boldsymbol{r}')$:
  - $*$ Parse $w = (\phi, r_1, \ldots, r_k, r_{\check{m}_1}, \ldots, r_{\check{m}_n})$.
  - $*$ Parse $\boldsymbol{r}'$ as $(r'_{\hat{z}_1}, \ldots, r'_{\hat{z}_n}, r'_{\check{m}_1}, \ldots, r'_{\check{m}_{n+1}})$.
  - $*$ For all $i \in [n+1]$, compute $r^*_{\check{m}_i} = r_{\check{m}_i} + r'_{\check{m}_i}$.
  - $*$ For all $i \in [k]$, compute $r'_i = r_i + r'_{\hat{z}_{\phi(i)}}$.
  - $*$ Set $w' = (\phi, r'_1, \ldots, r'_{k+1}, r^*_{\check{m}_1}, \ldots, r^*_{\check{m}_{n+1}})$.

Parse Trapdoor: Parse $\mathsf{td} = (\mathsf{td}_1, \mathsf{td}_2)$ where $\mathsf{td}_1 = (\rho, \sigma)$ and $\mathsf{td}_2 = (p_1, \ldots, p_n)$.

Use the trapdoor to create a satisfying assignment: Set variables $\hat{o} = \sum_{i=1}^k p_i\hat{e}_i, \hat{o'} = \sum_{i=1}^k 2p_i\hat{e}_i$, $(\forall i)$ $\check{a'}_i = \check{a}_i = \check{0}, \check{b'}_i = \check{b}_i = \check{0}, \hat{c}_i = \hat{0}, \hat{E}_i = \hat{0}, \mathsf{bit}_i = 0, f_i = 0, r_{\hat{z}_i} = 0, r_{\check{m}_i} = 0$. To commit to ψ we use the unit commitments with trivial randomness $\hat{\boldsymbol{u}}$ or $\check{\boldsymbol{u}}$ already available in crs_1.

Proof and auxiliary values generation: Compute the proof and the auxiliary values with the above assignment as the prover algorithm would do with the following exceptions:

– Use the value $\psi = 1$ and the trivial commitment randomness while proving equation $\mathcal{P}^1$.
– Use the value $\psi = 0$ with randomness the GS simulation trapdoors (i.e., $(\rho, 0)$ or $(\sigma, 0)$) while proving all the other equations involving ψ.

Fig. 19. Simulator of our extendable shuffle argument.

Theorem 4. *The construction above is an extendable non-interactive proof system for R_{SH} with extended zero knowledge under the SXDH assumption[13], the subset permutation pairing assumption (Ass. 6), and the subset simultaneous pairing assumption (Ass. 7).*

[13] Under this assumption GS is non-interactive proof system (Sec. A.3) with a dual-mode CRS.

Set variables: Set the following variables $\forall i \in [n]$:
$r_{\hat{z}_i}, r_{\check{m}_i}, \mathsf{bit}_i, f_i, \breve{a}_i, \breve{a'}_i, \check{b}_i, \check{b'}_i, \hat{c}_i, \hat{E}_i$ and the following other variables $\psi, \hat{o}, \hat{o'}$.

Set equations: Set the following equations:

$$(\forall i)\ \mathcal{D}^1_{i,1} : f_i \hat{\boldsymbol{z}}_{\boldsymbol{i}}{}^1 = r_{\hat{z}_i}\hat{g},\ \ \mathcal{D}^1_{i,2} : f_i \hat{\boldsymbol{z}}^2_i = f_i \hat{E}_i + r_{\hat{z}_i}\hat{\mathsf{ck}},\ \ \mathcal{F}_i : f_i = \mathsf{bit}_i \psi$$

$$(\forall i)\ \mathcal{D}^2_{i,1} : \psi \check{\boldsymbol{m}}^1_i = r_{\check{m}_i}\check{h},\ \ \mathcal{D}^2_{i,2} : \psi \check{\boldsymbol{m}}^2_1 = \mathsf{bit}_i \check{h} + r_{\check{m}_i}\check{\mathsf{ck}}$$

$$(\forall i)\ \mathcal{B}_i : \mathsf{bit}_i(1-\mathsf{bit}_i) = 0 \quad (\forall i)\ \mathcal{Q}^1_i : \breve{a}_i = \mathsf{bit}_i \breve{a'}_i \quad (\forall i)\ \mathcal{Q}^2_i : \check{b}_i = \mathsf{bit}_i \check{b'}_i$$

$$\mathcal{K} : \textstyle\sum_{i=1}^n \mathsf{bit}_i = k\psi \quad \mathcal{S}^1 : \sum_{i=1}^n \breve{a}_i = \sum_{i=1}^k \psi \check{h}_i$$

$$\mathcal{S}^2 : \textstyle\sum_{i=1}^n \check{b}_i = \sum_{i=1}^k \psi \check{\delta}_i \quad (\forall i)\ \mathcal{V}^1_i : \hat{g} \cdot \breve{a}_i = \hat{c}_i \cdot \check{h} \quad (\forall i)\ \mathcal{V}^2_i : \hat{g} \cdot \check{b}_i = \hat{c}_i \cdot \breve{a}_i$$

$$\mathcal{E}^1 : \hat{o} \cdot \check{h} + \textstyle\sum_{i=1}^n \hat{E}_i \cdot \breve{a}_i = \sum_{i=1}^k \hat{e}_i \cdot \check{h}_i \quad \mathcal{E}^2 : \hat{o'} \cdot \check{h} + \sum_{i=1}^n \hat{E}_i \cdot \check{b}_i = \sum_{i=1}^k \hat{e}_i \cdot \check{\delta}_i$$

$$\mathcal{P}^1 : \psi = 1 \quad \mathcal{P}^2 : \psi \hat{o} = \hat{0} \quad \mathcal{P}^3 : \psi \hat{o'} = \hat{0}$$

Map witness to satisfying assignment: For all $i \in [n]$ set the variables $r_{\hat{z}_i}, r_{\check{m}_i}, \mathsf{bit}_i, f_i, \breve{a}_i, \breve{a'}_i, \check{b}_i, \check{b'}_i, \hat{c}_i, \hat{E}_i$ as follows:

- For all $i \in [n]$ set $r_{\check{m}_i}$ as the value given in input.
- Set $\mathcal{I} \leftarrow \emptyset$.
- For all $j \in [k]$ let $\phi(j) = i$, set $\breve{a'}_i = \breve{a}_i = \check{h}_j, \check{b'}_i = \check{b}_i = \check{\delta}_j, \hat{c}_i = \hat{g}_j, \hat{E}_i = \hat{e}_j, \mathsf{bit}_i = 1, f_1 = 1, r_{\hat{z}_i} = r_j, \mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$.
- For all $i \in [n] \setminus \mathcal{I}$, set $\breve{a'}_i = \breve{a}_i = \check{0}, \check{b'}_i = \check{b}_i = \check{0}, \hat{c}_i = \hat{0}, \hat{E}_i = \hat{0}, \mathsf{bit}_i = 0, f_i = 0, r_{\hat{z}_i} = 0$.

Set $\psi = 1, \hat{o} = \hat{0}$, and $\hat{o'} = \hat{0}$.
Proof and auxiliary values generation:

1. Generate GS proof elements and GS commitments for all the above equations. Let Π contain all the GS commitments and proofs elements related to each equation. Note that $\hat{\boldsymbol{u}}$ and $\check{\boldsymbol{u}}$ (available in the crs) are already a commitment to 1 with trivial randomness, therefore the prover does not have to compute GS commitments to ψ nor to explicitly prove equation $\mathcal{P}^1$.
2. Define $\mathsf{AUX} = (\mathsf{aux}_i)_{i\in[n]}$, with $\mathsf{aux}_i = (\mathsf{aux}^i_{\mathcal{K}}, \mathsf{aux}^i_{\mathcal{S}^1}, \mathsf{aux}^i_{\mathcal{S}^2}, \mathsf{aux}^i_{\mathcal{E}^1}, \mathsf{aux}^i_{\mathcal{E}^2})$ where:
   - $\mathsf{aux}^i_{\mathcal{K}} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}})$ with $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}} = (r_{\mathsf{bit}_i}\check{\boldsymbol{u}}^1, r_{\mathsf{bit}_i}\check{\boldsymbol{u}}^2)$
   - $\mathsf{aux}^i_{\mathcal{S}^1} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ where $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (r_{\breve{a}_i}\hat{\boldsymbol{u}}^1, r_{\breve{a}_i}\hat{\boldsymbol{u}}^2)$, $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (s_{\breve{a}_i}\hat{\boldsymbol{u}}^1, s_{\breve{a}_i}\hat{\boldsymbol{u}}^2)$.
   - $\mathsf{aux}^i_{\mathcal{S}^2} = (\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ where $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (r_{\check{b}_i}\hat{\boldsymbol{u}}^1, r_{\check{b}_i}\hat{\boldsymbol{u}}^2)$, $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (s_{\check{b}_i}\hat{\boldsymbol{u}}^1, s_{\check{b}_i}\hat{\boldsymbol{u}}^2)$.
   - $\mathsf{aux}^i_{\mathcal{E}^1} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}})$ where $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}} = (r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\breve{\boldsymbol{a}}_{\boldsymbol{i}}}, r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\breve{\boldsymbol{a}}_{\boldsymbol{i}}})$, $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (\hat{0}, r_{\breve{a}_i}\hat{E}_i)$, $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}} = (s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\breve{\boldsymbol{a}}_{\boldsymbol{i}}}, s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\breve{\boldsymbol{a}}_{\boldsymbol{i}}})$, $\mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (\hat{0}, s_{\breve{a}_i}\hat{E}_i)$.
   - $\mathsf{aux}^i_{\mathcal{E}^2} = (\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}}, \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}}, \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}})$ where $\mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{v}}}} = (r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\check{\boldsymbol{b}}_{\boldsymbol{i}}}, r_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\check{\boldsymbol{b}}_{\boldsymbol{i}}}), \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{v}}}} = (0, r_{\check{b}_i}\hat{E}_i), \mathsf{aux}_{\check{\boldsymbol{\pi}}_{\hat{\boldsymbol{w}}}} = (s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{1}}_{\breve{\boldsymbol{a}}_{\boldsymbol{i}}}, s_{\hat{E}_i}\check{\boldsymbol{d}}^{\boldsymbol{2}}_{\breve{\boldsymbol{a}}_{\boldsymbol{i}}}), \mathsf{aux}_{\hat{\boldsymbol{\pi}}_{\check{\boldsymbol{w}}}} = (0, s_{\check{b}_i}\hat{E}_i)$.
3. Output (Π, AUX).

Fig. 18. Prover algorithm of our extendable shuffle argument.

We prove the above theorem via the following lemmas. The specific assumptions used in each lemma can be deduced from its proofs.

Lemma 13. $\mathsf{Com}_{\mathsf{ck}}$ *is a perfectly binding and computationally hiding commitment scheme.*

Proof. It trivially follows from the IND-CPA security of the ElGamal encryption.

Lemma 14. SH *is complete (Def. 10).*

Proof. It follows from the completeness of the GS proof system and from the fact that correctly computed auxiliary values satisfy the verification equation checked by $\mathsf{AuxVerify}$.

Lemma 15. SH *has admissible transformations (Def. 11).*

Proof. AddT_x updates the statement by just replacing commitments to bits and elements in position α with freshly computed ones to 1 and freshly added $\hat{e}_{k+1}$ respectively. Then it re-randomizes all the commitments in the statement. Finally, it includes the freshly added $\hat{e}_{k+1}$ in the clear-text elements. AddT_w just updates the opening randomness of the commitments according to the re-randomization factors that were previously used. Finally, it just updates the mapping ϕ to include the newly added $\hat{e}_{k+1}$. Simple inspection and completeness of the re-randomization procedure shows that if $(\mathsf{ck}, x, w) \in R_{\mathsf{SH}}$ then $(\mathsf{ck}, x', w') \in R_{\mathsf{SH}}$. An analogous discussion applies to $(\mathsf{ExtT}_x, \mathsf{ExtT}_w)$.

Lemma 16. SH *is transformation complete (Def. 12).*

Proof. We now show that the first requirement of transformation completeness is always satisfied. The discussion for the second requirement is basically the same. PAdd does the following operations:

- It replaces old GS commitments and proof elements related to equations $\mathcal{D}^1_{\cdot,\alpha}, \mathcal{D}^2_{\cdot,\alpha}, \mathcal{F}_\alpha, \mathcal{B}_\alpha, \mathcal{Q}^1_\alpha, \mathcal{Q}^2_\alpha, \mathcal{V}^1_\alpha, \mathcal{V}^2_\alpha$ with new ones producing accepting GS commitments/proof elements for such equations. This follows from the completeness of GS (steps 1-8).
- It removes the contribution of old committed variables from the proof elements related to equations $\mathcal{K}, \mathcal{S}^1, \mathcal{S}^2, \mathcal{E}^1, \mathcal{E}^2$ and adds the contribution of the freshly committed variables. Then, it re-randomizes the commitments $\hat{\boldsymbol{z}}_{\boldsymbol{i}}, \check{\boldsymbol{m}}_{\boldsymbol{i}}$ with $i \in [n]$ (as AddT_x would do) and adjusts the GS commitments $\check{\boldsymbol{d}}_{\boldsymbol{r}_{\hat{\boldsymbol{z}}_{\boldsymbol{i}}}}, \check{\boldsymbol{d}}_{\boldsymbol{r},\boldsymbol{i}}$ with $i \in [n]$ to contain the updates randomness. Since in equations $\mathcal{D}^1_{\cdot,i}, \mathcal{D}^2_{\cdot,i}$ the variables $r_{\hat{z}_i}$ and $r_{\check{m}_i}$ only multiply public constants, there is no need to update any of the related proof elements. The overall process (step 9 to 15) produces accepting GS proofs w.r.t. the new statement x'. This follows from the linearity of the GS proof generation and verification equation.
- Finally, all the GS commitments and proof elements are re-randomized by calling $\mathsf{GS.RandPr}$. This produces accepting proofs w.r.t. x'. This follows from the completeness of GS.

Notice that by construction AddT_x and PAdd produce the same updated statement x' when they are run with the same transformation randomness. Then, AddT_w, as argued before, produces a valid witness for x'. Additionally, AuxUpd just shifts the auxiliary values in the same way the proof elements are shifted by $\mathsf{GS.RandPr}$. As a result, the checks done by $\mathsf{AuxVerify}$ will still pass with the updated information as the addition operation exactly matches the computation of a fresh proof over (x', w').

Lemma 17. SH *is sound (Def. 13).*

Proof. Recall that crs_1 is generated as a perfectly binding string of the GS proof system. Consequently, the proofs related to all equations are perfectly sound proofs certifying that the committed variables satisfy such equations. Recall that $\mathsf{Com}_{\mathsf{ck}}$ is perfectly binding, and thus there is only one possible satisfying assignment for variables $\hat{E}_i$ and bit_i. Additionally, we can perfectly extract all the variables that are group elements. Therefore, we can extract all the variables $\check{a}_i, \check{b}_i, \hat{c}_i, \hat{E}_i$ with $i \in [n]$. Then given such variables, we can compute the set of indexes $\mathcal{J} = \{\alpha_1, \ldots, \alpha_k\}$ such that $\sum_{i \in \mathcal{J}} \check{a}_i = \sum_{i=1}^k \check{h}_i$ and $\sum_{i \in \mathcal{J}} \check{b}_i = \sum_{i=1}^k \check{\delta}_i$. Therefore, $\{\check{a}_{\alpha_i}\}_{i \in [k]}$ and $\{\check{b}_{\alpha_i}\}_{i \in [k]}$ satisfy a subset permutation pairing problem meaning that with overwhelming

probability $\{(\check{a}_{\alpha_i}, \check{b}_{\alpha_i}\}_{i\in[k])}$ is a permutation of $\{(\check{h}_i, \check{\delta}_i)\}$. We also have that $\psi = 1$ which implies $\hat{o} = \hat{o'} = \hat{0}$. It follows that $\sum_{i\in\mathcal{J}} \hat{E}_i \cdot \check{a}_i = \sum_{i=1}^{k} \hat{e}_i \cdot \check{h}_i$ and $\sum_{i\in\mathcal{J}} \hat{E}_i \cdot \check{b}_i = \sum_{i=1}^{k} \hat{e}_i \cdot \check{\delta}_i$. By rewriting such equations taking into consideration that $\{(\check{a}_{\alpha_i}, \check{b}_{\alpha_i})\}_{i\in[k]}$ is a permutation of $\{(\check{h}_i, \check{\delta}_i)\}$ we have that there exists a permutation π such that $\sum_{i=1}^{k}(\hat{E}_{\alpha_{\pi(i)}} - \hat{e}_i) \cdot \check{h}_i = 0_{\mathbb{T}}$ and $\sum_{i=1}^{k}(\hat{E}_{\alpha_{\pi(i)}} - \hat{e}_i) \cdot \check{\delta}_i = 0_{\mathbb{T}}$. This gives us a subset simultaneous pairing problem that implies that, except with negligible probability, $\hat{E}_{\alpha_{\pi(i)}} = \hat{e}_i$ for all $i \in [k]$.

Lemma 18. SH *is addition private (Def. 14).*

Proof. Let us point out the differences between the experiment executions with $b = 0$ and $b = 1$. In both cases, as argued before, the statement x' that is handled to $\mathcal{A}$ is exactly the same. Let us now look at the distribution of (Π, AUX) in both experiment. When $b = 0$ both Π and AUX are freshly computed using the witness provided by $\mathcal{A}$. When $b = 1$ the GS commitments to variables $f_\alpha, \mathsf{bit}_\alpha, \check{a}_\alpha, \check{a'}_\alpha, \check{h}_{k+1}, \check{b}_\alpha, \check{b'}_\alpha, \hat{c}_\alpha, \hat{E}_\alpha$ are freshly computed, as well as the proof elements related to equations $\mathcal{D}^1_{\cdot,\alpha}, \mathcal{D}^2_{\cdot,\alpha}, \mathcal{F}_\alpha, \mathcal{B}_\alpha, \mathcal{V}^1_\alpha, \mathcal{V}^2_\alpha$ that are re-computed running GS.Prv. The distribution of these GS commitments and proof elements is identical in both cases.

The remaining proof elements and GS commitments are taken form Π^* before being updated/re-randomized. The auxiliary values are taken from AUX^*. The list AUX^* contains group elements satisfying the verification equations checked by AuxVerify[14]. Let us first focus on equations $\mathcal{K}, \mathcal{S}^1, \mathcal{S}^2, \mathcal{E}^1, \mathcal{E}^1$ (i.e, the ones that come with associated auxiliary values). It straightforward to see that after all the GS commitments related to the variables involved in such equations have been re-randomized by GS.RandPr, we get new commitments that are equally distributed by freshly generated ones. Auxiliary values are updated as random group elements, using the same randomness previously used to re-randomize the commitments. As already shown in previous works [37,11,23], re-randomized proof elements are distributed as randomly chosen proof elements from the space of all valid proof elements, given that the GS commitments to the involved variables are fixed. Since the GS commitments are fully re-randomized, the result is a randomly chosen proof given a fixed solution, along with auxiliary values related to the randomly chosen commitments so that the verification check of AuxVerify is successful. It follows that, the joint distribution of GS commitments, proofs elements, and AUX is identically distributed both when $b = 0$ and $b = 1$[15]. Finally, for all the other equations not involving such variables it suffices to reduce to the re-randomization of GS proofs.

Lemma 19. SH *is extension private (Def. 15).*

Proof. The proof basically mirrors the one of addition privacy.

Lemma 20. SH *is extended zero knowledge (Def. 20).*

Proof. We first change the way crs_1 is generated by generating it as a perfectly hiding string of the GS proof system. By the SXDH assumption perfect binding and perfect hiding common reference strings for the GS proof system are computationally indistinguishable, so the adversary's success probability only changes negligibly. The sole distinction between an actual proof and a simulated proof lies in the witness provided to the GS prover. Due to the perfect witness indistinguishability of the GS proof over a perfectly hiding crs, genuine proofs and simulated proofs are indistinguishable. It remains to argue that the auxiliary values $\mathcal{A}$ receives do not give $\mathcal{A}$ any advantage in distinguishing honest and simulated proofs. It is straightforward to notice that also the auxiliary values are identically distributed in both cases. Indeed, the variables $\mathsf{bit}_i, \check{a}_i, \check{b}_i, \hat{E}_i, \hat{c}_i$ for $i \in [n] \setminus Im(\phi)$ are set to neutral elements in both cases, and thus the auxiliary values related to such variables just allow $\mathcal{A}$ to verify that they are set as expected. Moreover, such auxiliary values are completely independent to the values (and GS commitment openings) of the above variables for indexes $j \in Im(\phi)$, which is the only difference between honest and simulated proofs.

[14] We are guaranteed of that since otherwise $\mathcal{A}$ would not be admissible.

[15] In particular, notice that after the invocation of PAdd, the proofs for all equations are accepting.

C The EP used in our $\mathfrak{Tetris}$

In this section, we describe our instantiation of the EP used in our $\mathfrak{Tetris}$. It uses similar techniques to our extendable shuffle argument and the ENIWI of [8]. We set $\mathcal{L}_{\mathsf{trap}}$ to be the language of DH tuples, and let $x_{\mathsf{trap}} = (\hat{a}, \hat{b}, \hat{c})$. The relation of our EP is defined below.

$$\begin{aligned} R_{\mathsf{Dtr}} = \{&(\mathsf{ck}, x = (k, x_1, \ldots, x_n, \check{\boldsymbol{m}}_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}_{\boldsymbol{n}}), w = (\phi, w_1, \ldots, w_k, r_{\check{m}_1}, \ldots, r_{\check{m}_n}))| \\ &\forall i \in [n]\ \exists\ \mathsf{bit}_i : \ \ \check{\boldsymbol{m}}_{\boldsymbol{i}} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; r_{\check{m}_i}) \wedge \mathsf{bit}_i \in \{0,1\} \wedge \sum_{i=1}^{n} \mathsf{bit}_i = k \wedge \\ &\forall_{i=1}^{k} (\mathsf{bit}_{\phi(i)} = 1 \wedge (\mathsf{ck}, x_{\phi(i)}, w_i) \in R_{\mathsf{D}}) \wedge \phi \text{ is injective}\}, \end{aligned}$$

where

$$\begin{aligned} R_{\mathsf{D}} = \{&(\mathsf{ck}, x = (\hat{\boldsymbol{z}}, \mathsf{pk}_{\mathsf{T}}, \hat{\tau}, m, x_{\mathsf{trap}}), w)|(w = (\hat{e}, r_{\hat{z}}, \mathsf{sk}_{\mathsf{T}}) \wedge e \leftarrow \mathsf{Tag}(\mathsf{sk}_{\mathsf{T}}, \hat{\tau}, m) \wedge \\ &\hat{\boldsymbol{z}} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\hat{e}; r_{\hat{z}}) \wedge (\mathsf{pk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}}) \in R_{\mathsf{key}}) \vee (w = w_{\mathsf{trap}} \wedge (x_{\mathsf{trap}}, w_{\mathsf{trap}}) \in R_{\mathcal{L}_{\mathsf{trap}}})\}. \end{aligned}$$

In Fig. 20 we report the prover algorithm of our EP.

As all of the other algorithms are almost identical to the ones presented in SH, we just briefly describe the other algorithms of our EP:

- CRSSetup just runs the setup algorithm of GS.
- The commitment scheme is the same used in SH.
- The transformations (AddT, ExtT) are identical to the ones of SH, except they do not consider any clear-text element.
- PVfy just runs the verifier algorithm of GS.
- PExt works similarly to ShExtend. It creates new variables, GS commitments, and proof elements for all equations but $\mathcal{K}$. It computes the new auxiliary value $\mathsf{aux}_{\mathcal{K}}^{n+1}$ and updates the proof elements related to $\mathcal{K}$ in the same way of ShExtend. It re-randomizes the statement and adjusts the related GS commitments as in step 13 of ShExtend and runs GS.RandPr.
- PAdd works similarly to ShAdd. It replaces all the variables in position α with fresh ones re-computing the related GS commitments and the proof elements of the corresponding equations, except for $\mathcal{K}$. It updates the proof for $\mathcal{K}$ in the same way ShAdd does. It re-randomizes the statement and adjusts the related GS commitments in the same way ShAdd does.
- AuxUpd and AuxVerify work in the same way as the corresponding algorithms of SH. The only difference is that here only equation $\mathcal{K}$ is involved in the process.

Proof Intuition. Given the similarity with SH and with the ENIWI of [8] here we give a proof sketch of why EP is an extendable non-interactive proof system with WAI (Def. 17), FPWI (Def. 18), and EWI (Def. 19), under the SXDH assumption. We focus on WAI, FPWI, and EWI, as the other properties can be proven with a discussion that basically mirrors the one already done for SH.

1. **Witness addition indistinguishability**: It suffices to observe that x' is identical in both cases. The distribution of the re-randomization factors is identical. The only difference is that two different witness are used to perform an addition over the same position α, but with a different witness. The distribution of the auxiliary values is also identical in both executions, as after re-randomization it only depends on the active indexes, that are identical in both cases. It remains to argue that the distribution of Π is indistinguishable in both executions. This basically from from the re-randomization property of GS, as two accepting GS proofs are indistinguishable after their re-randomization.
2. **Fixed position witness indistinguishability**: Since the active indexes are the same in both cases, it follows that the auxiliary values are identically distributed, as they are independent from the used witness. Therefore to argue that the two experiment executions are indistinguishable it suffices to reduce to the witness indistinguishability of GS.

Set variables: Define the following variables $\forall i \in [n] : r_{\hat{z}_i}, \hat{E}_i, r_{\check{m}_i}, \mathsf{bit}_i, f_i, s_i^0, s_i^1, \mathsf{sk}_{\mathsf{T}}^i, a_i, b_i$. Finally, define the variable ψ.
Set equations:

$$(\forall i)\ \mathcal{D}_{i,1}^1 : s_i^0 \hat{\boldsymbol{z}}_i^1 = \hat{\mathsf{ck}} r_{\hat{z}_i}\ \ (\forall i)\ \mathcal{D}_{i,2}^1 : s_i^0 \hat{\boldsymbol{z}}_i^2 = s_i^0 \hat{E}_i + \hat{g} r_{\hat{z}_i}$$

$$(\forall i)\ \mathcal{D}_{i,1}^2 : \psi \check{\boldsymbol{m}}_i^1 = r_{\check{m}_i} \check{\mathsf{ck}}\ \ (\forall i)\ \mathcal{D}_{i,2}^2 : \psi \check{\boldsymbol{m}}_1^2 = \mathsf{bit}_i \check{h} + r_{\check{m}_i} \check{h}$$

$$(\forall i)\ \mathcal{F}_i : f_i(1 - f_i) = 0\ \ (\forall i)\ \mathcal{B}_i : \mathsf{bit}_i(1 - \mathsf{bit}_i) = 0\ \ (\forall i)\ \mathcal{U}_i^1 : s_i^0 = \mathsf{bit}_i(1 - f_i)$$

$$(\forall i)\ \mathcal{U}_i^2 : s_i^1 = \mathsf{bit}_i f_i\ \ (\forall i)\ \mathcal{T}_i : s_i^0 \hat{E}_i = \hat{\tau} \mathsf{sk}_{\mathsf{T}}^i + \hat{\mathsf{pk}_2^i} m \mathsf{sk}_{\mathsf{T}}^i\ (\forall i)\ \mathcal{Y}_i : s_i^0 \hat{\mathsf{pk}_1^i} = \hat{g} \mathsf{sk}_{\mathsf{T}}^i$$

$$(\forall i)\ \mathcal{M}_i^1 : s_i^1 \hat{a} = \hat{g} a_i\ (\forall i)\ \mathcal{M}_i^2 : s_i^1 \hat{b} = \hat{g} b_i\ (\forall i)\ \mathcal{M}_i^3 : b_i \hat{a} = \hat{c} s_i^1$$

$$\mathcal{K} : \textstyle\sum_{i=1}^n \mathsf{bit}_i = k\psi\ \ \mathcal{P} : \psi = 1.$$

Map witness to satisfying assignment: For all $i \in [n]$ set the variables as follows:

- For all $i \in [n]$ set $r_{\check{m}_i}$ as the value given in input.
- Set $\mathcal{I} \leftarrow \emptyset$.
- For all $j \in [k]$ let $\phi(j) = i$, set $r_{\hat{z}_i} = r_j, \hat{E}_i = \hat{e}_j, \mathsf{bit}_i = 1, f_i = 0, s_i^0 = 1, s_i^1 = 0, \mathsf{sk}_{\mathsf{T}}^i = \mathsf{sk}_{\mathsf{T}}^j, a_i = 0, b_i = 0$ $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$.
- For all $i \in [n] \setminus \mathcal{I}$, set $r_{\hat{z}_i} = 0, \hat{E}_i = \hat{0}, \mathsf{bit}_i = 0, f_i = 0, s_i^0 = 0, s_i^1 = 0, \mathsf{sk}_{\mathsf{T}}^i = 0, a_i = 0, b_i = 0$.

Set $\psi = 1$. Notice that variable f_i can be set to 1 whenever the prover wants to use a witness for x_{trap}.
Proof and auxiliary values generation:

1. Generate GS proof elements and GS commitments for all the above equations and put them in Π.
2. Define $\mathsf{AUX} = (\mathsf{aux}_i)_{i \in [n]}$, with $\mathsf{aux}_i = \mathsf{aux}_{\mathcal{K}}^i$, where $\mathsf{aux}_{\mathcal{K}}^i = (\mathsf{aux}_{\vec{\boldsymbol{\pi}}_{\mathfrak{d}}})$ with $\mathsf{aux}_{\vec{\boldsymbol{\pi}}_{\mathfrak{d}}} = (r_{\mathsf{bit}_i} \check{\boldsymbol{u}}^1, r_{\mathsf{bit}_i} \check{\boldsymbol{u}}^2)$.

Fig. 20. Proving algorithm of our EP.

3. **Extended witness indistinguishability**: We sketch a sequence of indistinguishable hybrids that go from the experiment execution with $b = 0$ to the experiment execution with $b = 1$. We first switch the crs of the GS proof system to a hiding crs generated with the trapdoor (computational indistinguishability of the crs). We then use the trapdoor to prove all the equations except $\mathcal{P}$ by setting $\psi = 0$ as well as all the other variables (witness indistinguishability of GS). Then, we compute the commitments $\check{\boldsymbol{m}}_i$ with $i \in [n]$ with the bits induced from ϕ^1 (hiding of $\mathsf{Com}_{\mathsf{ck}}$). Finally, we compute the proof using w_1 (witness indistinguishability of GS).

Lemma 21. *Our* EP *achieves perfect Δ-extraction (Def. 16).*

Proof. Let $\mathsf{ck} \leftarrow \mathsf{ComSetup}(\mathsf{gk})$, and $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CRSSetup}(\mathsf{gk})$ and let $\mathsf{GS.Ext}$ be the extractor of GS [28]. We define the following extractor Ext that takes in input a statement $(\mathsf{crs}, \mathsf{td}, \mathsf{ck}, x = (k, x_1, \ldots, x_n, \check{\boldsymbol{m}}_{\boldsymbol{1}}, \ldots, \check{\boldsymbol{m}}_{\boldsymbol{n}}), \Pi_{\mathsf{EP}}))$, with $x_i = (\hat{\boldsymbol{z}}_{\boldsymbol{i}}, \mathsf{pk}^i_{\mathsf{T}}, \hat{\tau}, m, x_{\mathsf{trap}})$. Recall that $x_{\mathsf{trap}} \notin \mathcal{L}_{\mathsf{trap}}$, thus Ext runs $\mathsf{GS.Ext}$ on input the proof, extracting the group elements and the group-element representation (i.e., exponentiation in either $\hat{\mathbb{G}}$ or $\check{\mathbb{H}}$) of the scalars that are in the witness $w = (\phi, w_1, \ldots, w_k, r_{\check{m}_1}, \ldots, r_{\check{m}_n})$ for x. That is, for each $i \in [n]$, $\mathsf{GS.Ext}$ extracts $\check{h} r_{\check{m}_i}$, and $\check{h}\mathsf{bit}_i$, while for each $j \in [k]$ and $w_j = (\hat{e}_j, r_{\hat{z}_j}, \mathsf{sk}^j_{\mathsf{T}})$ it extracts $\hat{e}_j$, $\hat{g} r_{\hat{z}_j}$, $\hat{g}\mathsf{sk}^j_{\mathsf{T}}$. Ext defines the function $(\hat{e}_1, \ldots, \hat{e}_k) = \Delta((\hat{e}_j, r_{\hat{z}_j}, \mathsf{sk}^j_{\mathsf{T}})_{j \in [k]})$. Ext computes ϕ' as follows:

$c \leftarrow 1$
for $j \in [k]$:
  $flag = 0$
  while $c < n \wedge flag = 0$:
    if $\check{h}\mathsf{bit}_c = \check{h}$:
      $phi'(j) \leftarrow c$
      $flag = 1$
    $c \leftarrow c + 1$
return ϕ'

Ext returns $(\phi', w' = (\hat{e_1}, \ldots, \hat{e_k}))$. Notice that the CRS of our EP is generated as in the binding mode of GS, so for perfect F-knowledge of GS, it holds that each $\mathsf{bit}_i \in \{0, 1\}$, $\sum_{i=1}^n \mathsf{bit}_i = k$, the values $r_{\check{m}_1}, \ldots, r_{\check{m}_n}$ in w are such that $\check{\boldsymbol{m}}_{\boldsymbol{1}} \leftarrow \mathsf{Com}_{\mathsf{ck}}(\mathsf{bit}_i; r_{\check{m}_i})$, $\forall_{j=1}^k (\mathsf{bit}_{\phi(j)} = 1 \wedge F(\mathsf{ck}, x_{\phi(j)}, e_j, w_j) = 1)$. The only thing that remains to show is that $w' = \Delta(w_1, \ldots, w_k)$, which holds by construction.

Remark 2. The relation R_{key} of our DAPT (Sec. 7) contains a part that it is publicly verifiable that simply consists in checking that a pairing equation holds between the two group elements in the public key. Therefore, we do not include this check in the equation proved by the prover and assume it is verified by the verifier for every public key included in the statement.