Threshold Signatures with Private Accountability

Dan Boneh[1] and Chelsea Komlo[2]

[1] Stanford University
[2] University of Waterloo

Abstract. Existing threshold signature schemes come in two flavors: (i) *fully private*, where the signature reveals nothing about the set of signers that generated the signature, and (ii) *accountable*, where the signature completely identifies the set of signers. In this paper we propose a new type of threshold signature, called TAPS, that is a hybrid of privacy and accountability. A TAPS signature is fully private from the public's point of view. However, an entity that has a secret tracing key can trace a signature to the threshold of signers that generated it. A TAPS makes it possible for an organization to keep its inner workings private, while ensuring that signers are accountable for their actions. We construct a number of TAPS schemes. First, we present a generic construction that builds a TAPS from any accountable threshold signature. This generic construction is not efficient, and we next focus on efficient schemes based on standard assumptions. We build two efficient TAPS schemes (in the random oracle model) based on the Schnorr signature scheme. We conclude with a number of open problems relating to efficient TAPS.

1 Introduction

A threshold signature scheme [32] enables a group of n parties to sign a message only if t or more of the parties participate in the signing process. There are two types of threshold signature schemes:

- *A private threshold signature (PTS) scheme:* A signature σ on a message m reveals nothing about the threshold t, and reveals nothing about the quorum of t parties that generated the signature. The same holds even if the adversary sees a sequence of signatures on messages of its choice. Examples of PTS schemes include [59,36,28,41,16,60,48] and many others.
- *An accountable threshold signature (ATS) scheme:* A signature σ on a message m reveals the identity of all t parties who participated in generating the signature (and hence also reveals t). Moreover, it is not feasible for a quorum of t parties to frame another quorum. An ATS scheme is closely related to the notion of an *accountable subgroup multisignature* (ASM) [53,47,10,18,16,6,56]. However, we prefer the term ATS to contrast the two flavors of threshold signatures: ATS vs. PTS. An ATS has also been described as Traceable Secret Sharing (TSS) [45].

We will define these concepts more precisely in the next section.

A private threshold signature (PTS) scheme is used when there is a need to hide the inner-workings of an organization. For example, an organization that runs a web server may choose to split the server's secret TLS key among n machines so that at least t are needed to generate a signature and complete a TLS handshake. By using a PTS, the organization can hide the threshold t from the public, to avoid leaking the number of machines that an attacker needs to compromise in order to forge a signature. Similarly, a signature should reveal nothing about the set of t machines that participated in generating the signature so that nothing is revealed about which machines are currently online.

In contrast, an accountable threshold signature (ATS) scheme is often used in financial applications where there is a need for accountability. For example, if three of five bank executives are needed to authorize a banking transfer, then one wants full accountability in case a fraudulent transfer is approved. When using an ATS scheme, the signature on a fraudulent transaction will identify the three bank executives who authorized it.

The trivial t-out-of-n ATS scheme is one where every signing party locally generates a public-private key pair. The complete public key is defined as the concatenation of all n local public keys. When t parties need to sign a message m, they each sign the message using their local secret key, and the final signature is the concatenation of all t signatures. The verifier accepts such an ATS signature if it contains t valid signatures. This trivial ATS is used widely in practice, for example in Bitcoin multisig transactions [1].

While the scheme has many benefits, its downside is that signature size and verification time are at least linear in $t\lambda$, where λ is the security parameter. Several ATS constructions achieve much smaller signature size and verification time [53,10,18,56].

In summary, existing threshold signatures offer either complete privacy or complete accountability for the signing quorum, but cannot do both.

A new type of threshold signature. In this work we introduce a new type of threshold signature scheme, called TAPS, that provides full accountability while maintaining privacy for the signing quorum.

A **Threshold, Accountable, and Private Signature** scheme, or simply a **TAPS**, works as follows: (i) a key generation procedure takes n and t as input, and generates the public key pk and the n private keys $sk_1, \ldots, sk_n$ for the signers, (ii) a signing protocol among some t signers is used to generate a signature σ on a message m, and (iii) a signature verification algorithm takes as input pk, m, and σ and outputs accept or reject. Signatures generated by the signing protocol reveal nothing to the public about the threshold t or the quorum that generated the signature. In addition, the key generation procedure outputs a **tracing key** sk_{t}. Anyone in possession of sk_{t} can reliably trace a signature to the quorum that generated it. For security we require that a set of signers should be unable to frame some other set of signers by fooling the tracing procedure. We define the precise syntax for a TAPS scheme, and the security requirements, in Section 3.

If the tracing key sk_{t} is made public to all, then a TAPS is no different than an ATS scheme. Similarly, if sk_{t} is destroyed, then a TAPS is no different than a PTS scheme. However, if sk_{t} is known to a trusted tracing party (or secret shared among several parties), then the tracing party can provide accountability in case of a fraudulent transaction, while keeping all other information about the inner-workings of the organization private.

Applications. Consider an organization that holds digital assets that are managed on a public ledger (e.g., a blockchain). A digital signature must be recorded on the ledger in order to transfer an asset. The organization can protect the assets by requiring t-out-of-n trustees to sign a transfer request. It can use an ATS scheme, but then the threshold t and the set of signers will be public for the world to see. Or it can use a PTS scheme to secret share a single signing key among the n trustees, but then there is no accountability for the trustees.

A TAPS provides a much better solution: the organization can hold on to the tracing key sk_{t} so that the threshold and the set of signers remain private, but the trustees are accountable in case of a fraudulent transfer. The value of n and t are typically relatively small, say less than twenty.

The same applies in the web server setting. The web server's TLS secret signing key could be shared among t-out-of-n machines so that t machines are needed to complete a TLS handshake. The tracing key would be kept in offline storage. If at some point it is discovered that the web server's secret key has been compromised, and is being used by a rogue web server, then the tracing key could be applied to the rogue server's signatures to identify the set of machines that were compromised by the attacker.

Constructing TAPS. We provide a number of constructions for TAPS schemes. In Section 4 we present a generic construction that shows how to construct a TAPS from any ATS scheme. The construction is quite inefficient since it makes use of general zero knowledge. While there are several important details that are needed to obtain a secure construction, the high level approach for generating a TAPS signature is as follows: (i) the signing parties generate an ATS signature σ on a message m, (ii) they encrypt σ using a public key encryption scheme to obtain a ciphertext ct, and (iii) the final TAPS signature is $\sigma' = (ct, \pi)$, where π is a non-interactive zero knowledge proof that the decryption of ct is a valid ATS signature on m. To verify a signature, one verifies that π is valid. The tracing key sk_{t} is the decryption key that lets one decrypt ct. Then, using sk_{t} one can decrypt ct, and run the ATS tracing algorithm on the resulting ATS signature σ. The description here is only meant as an outline, and is not secure as is. The complete construction is provided in Section 4.

Next, we turn to constructing a practical TAPS scheme. In Section 5 we build two efficient TAPS schemes from Schnorr signatures [58]. To do so, we modify the generic construction so that the statement that needs to be proved in zero knowledge is as simple as possible. We then use either a Sigma protocol [29] or Bulletproofs [22,24] to prove the statement. The resulting public key and signature sizes are

	Public Key Size		Signature Size		Verify Time (group ops)	Trace Time (group ops)
	$\mathbb{G}$	$\mathbb{Z}_q$	$\mathbb{G}$	$\mathbb{Z}_q$		
Sigma	$2n+4$	0	$n+4$	$2n+5$	$O(n)$	$O(n)$
Bulletproofs	$n+\frac{n}{e}+O(1)$	0	$\frac{n}{e}+O(\log n)$	4	$O(n)$	$O(n \cdot 2^{e/2})$

Table 1. An n-party TAPS based on the Schnorr signature scheme in a group $\mathbb{G}$ of order q. The construction uses either a Sigma protocol or Bulletproofs. The Bulletproofs TAPS signature is shorter by a factor of about e, but tracing time is higher. Taking $e := 40$ is a reasonable choice.

summarized in Table 1. For small n, both schemes have reasonable performance. As n grows, signatures produced by the Bulletproofs scheme are about 40 times shorter.

We note that due to the traceability and privacy requirements, a TAPS signature must encode the signing quorum while hiding the threshold t, and therefore must be at least n bits long. In Section 6 we discuss relaxing the *full tracing* requirement with a weaker tracing property we call *quorum confirmation*. Here the tracing algorithm takes as input sk_t and a suspect quorum set $C \subseteq [n]$, and confirms if C is indeed the quorum set that generated a given signature. If this weaker confirmation property is sufficient, then our Bulletproofs approach can lead to a logarithmic size TAPS signature. Note that when n is small, confirmation can lead to full tracing by testing all possible quorum sets until one is confirmed.

A different perspective. A TAPS system can be described as a group signature scheme where t signers are needed to sign on behalf of the group. Recall that in a group signature scheme [27] a group manager provisions every member in the group with a secret signing key. Any group member can sign on behalf of the group without revealing the identity of the signer. In addition, there is a tracing key that lets an entity that holds that key trace a given group signature to the single member that issued that signature. A TAPS can be viewed as a generalization of this mechanism. In a TAPS scheme, at least t members of the group are needed to generate a group signature. The signature reveals nothing to the public about the identity of the signers or t. However, the tracing key enables one to trace the signature back to some t members that participated in generating the signature.

In the literature, the term *threshold group signature* refers to a scheme where the role of the group manager is distributed among a set of authorities with a threshold access structure [15,26]. A TAPS is quite different. Here the threshold refers to the number of parties needed to generate a signature on behalf of the group. See also our discussion of related work below.

1.1 Additional related work

Ring Signatures. Ring signatures [57,55,12,12] allow a signer to sign a message on behalf of an ad-hoc ring of signers. The signature reveals nothing about which ring member generated the signature. As such, anyone can gather a set of public keys, and produce a ring signature over some message without interacting with the owners of those keys. Our notion of TAPS signatures requires a threshold of t signers to generate a signature, where t is hidden from the public. In the basic group or ring setting the threshold t is not secret, it is always set to $t = 1$.

While accountable (traceable) ring signatures with a tracing authority have been defined in the literature [63,38,37,21], these schemes are limited to a *single* signer, as opposed to a threshold of signers within the ring. Dodis et al. [33] defined a multi-party ring signature that builds upon one-way cryptographic accumulators and supports an identity escrow extension. However, the scheme does not enforce a threshold number of signers to anyone other than the designated tracing authority (by recovering the identities of the signers). In contrast, TAPS requires that anyone be able to verify that a threshold number of signers participated in generating a signature.

Threshold ring signatures, called *thring signatures*, were studied in a number of works [23,50,61,54,46]. Here the ring signature represents some t-out-of-n set of signers. However, these schemes provide no tracing, and therefore do not fulfill the notions of accountability required by TAPS. Similarly, linkable threshold ring signatures [5,34] only require that any two ring signatures produced by the same signers can be linked, but not traced.

A ring signature by Bootle et al. [21] combines Camenisch's group signature scheme [25] with a one-out-of-many proof of knowledge. This construction uses similar techniques as our Schnorr TAPS construction, but supports only a single signer, rather than a threshold, so provides quite a different functionality.

Group Signatures. First introduced by Chaum and van Heyst [27], group signatures [17,39,49,51,31,20,13] enable a group member to sign a message such that the verifier can determine that a member generated the signature, but not *which* member. If needed, a tracing authority can trace a signature to its signer. A group manager is trusted to manage the group's membership. The security notions for a group signatures were defined by Bellare et al. [9], but focus on a single signer who is signing on behalf of the group. Traditionally *threshold group signatures* refers to the ability to distribute the roles of the group manager [15,26], as opposed to requiring a threshold number of participants to issue a signature.

2 Preliminaries

Notation: We use $\lambda \in \mathbb{Z}$ to denote the security parameter in unary. We use $x \leftarrow y$ to denote the assignment of the value of y to x. We write $x \xleftarrow{\$} S$ to denote sampling an element from the set S independently and uniformly at random. For a randomized algorithm $\mathcal{A}$ we write $y \xleftarrow{\$} \mathcal{A}(x)$ to denote the random variable that is the output of $\mathcal{A}(x)$. We use $[n]$ for the set $\{1, \ldots, n\}$. Throughout the paper $\mathbb{G}$ is a cyclic group of prime order q, and $\mathbb{Z}_q$ is the ring $\mathbb{Z}/q\mathbb{Z}$. We let g be a generator of $\mathbb{G}$. We denote vectors in bold font: $\mathbf{u} \in \mathbb{Z}_q^m$ is a vector of length m whose elements are each in $\mathbb{Z}_q$. For vectors $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$ we let $\mathbf{g}^{\mathbf{a}}$ denote the product $\prod_{i=1}^n g_i^{a_i} \in \mathbb{G}$.

Our constructions make use of a few standard primitives. We define these briefly here.

Definition 1. *A* **public key encryption scheme** $\mathcal{PKE}$ *for a message space* $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ *is a triple of* PPT *algorithms* $(KeyGen, Encrypt, Decrypt)$ *invoked as*

$$(pk, sk) \xleftarrow{\$} KeyGen(1^\lambda), \quad ct \xleftarrow{\$} Encrypt(pk, m), \quad m \leftarrow Decrypt(sk, ct).$$

The only security requirement is that $\mathcal{PKE}$ *be* **semantically secure**, *namely, for every* PPT *adversary* $\mathcal{A}$ *the following function is negligible*

$$\mathbf{Adv}^{\text{indcpa}}_{\mathcal{A},\mathcal{PKE}}(\lambda) := \Big|\Pr\big[\mathcal{A}^{\text{ENC}(0,\cdot,\cdot)}(pk) = 1\big] - \Pr\big[\mathcal{A}^{\text{ENC}(1,\cdot,\cdot)}(pk) = 1\big]\Big|,$$

where $(pk, sk) \xleftarrow{\$} KeyGen(1^\lambda)$, *and for* $b \in \{0,1\}$ *and* $m_0, m_1 \in \mathcal{M}_\lambda$, *the oracle* $\text{ENC}(b, m_0, m_1)$ *returns* $ct \xleftarrow{\$} Encrypt(pk, m_b)$.

When $\mathcal{M}_\lambda \subseteq \{0,1\}^{\leq \ell_\lambda}$, for some ℓ_λ, our definition of semantic security requires that the encryption scheme be *length hiding*: an adversary cannot distinguish the encryption of $m_0 \in \mathcal{M}_\lambda$ from $m_1 \in \mathcal{M}_\lambda$ even if m_0 and m_1 are different lengths. This can be achieved by having the encryption algorithm pad the plaintext to a fixed maximum length using an injective pad (e.g., $100\ldots00$), and having the decryption algorithm remove the pad.

Definition 2. *Let* $\mathcal{R} := \{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$. *A* **commitment** *scheme* $\mathcal{COM}$ *is a pair of* PPT *algorithms* $(Commit, Verify)$ *invoked with* $r \in \mathcal{R}_\lambda$ *as*

$$\mathsf{com} \leftarrow Commit(x, r) \quad \text{and} \quad Verify(x, r, \mathsf{com}) \in \{0,1\}.$$

The scheme is **secure** *if it is unconditionally hiding and computationally binding. In particular, for all* x, x' *the distributions* $\{\mathcal{COM}(x, r)\}$ *and* $\{\mathcal{COM}(x', r')\}$ *have negligible statistical distance* $\epsilon(\lambda)$ *when* $r, r' \xleftarrow{\$} \mathcal{R}_\lambda$. *In addition, for every* PPT *adversary* $\mathcal{A}$ *the following function is negligible*

$$\mathbf{Adv}^{\text{bind}}_{\mathcal{A},\mathcal{COM}}(\lambda) := \Pr\left[\begin{array}{l|c} x \neq x', \quad r, r' \in \mathcal{R}_\lambda, & \\ Verify(x, r, \mathsf{com}) = 1 & (\mathsf{com}, x, r, x', r') \xleftarrow{\$} \mathcal{A}(\lambda) \\ Verify(x', r', \mathsf{com}) = 1 & \end{array}\right].$$

Definition 3. *A* **signature scheme** $\mathcal{SIG}$ *is a triple of* PPT *algorithms* $(KeyGen, Sign, Verify)$ *invoked as*

$$(pk, sk) \xleftarrow{\$} KeyGen(1^\lambda), \quad \sigma \xleftarrow{\$} Sign(sk, m), \quad Verify(pk, m, \sigma) \in \{0,1\}.$$

The scheme is **strongly unforgeable** *if the following function is negligible*

$$\mathbf{Adv}^{\text{eufcma}}_{\mathcal{A},\mathcal{SIG}}(\lambda) := \Pr\left[\begin{array}{ll} Verify(pk, m, \sigma) = 1 & \\ (m, \sigma) \notin \{(m_i, \sigma_i)\}_{i=1}^{q} & : \end{array} \begin{array}{l} (pk, sk) \xleftarrow{\$} KeyGen(1^\lambda) \\ (m, \sigma) \xleftarrow{\$} \mathcal{A}^{\text{SIGN}(\cdot)}(pk) \end{array}\right]$$

where $\text{SIGN}(m_i)$ *returns* $\sigma_i \xleftarrow{\$} Sign(sk, m_i)$ *for* $i = 1, \ldots, q$.

Definition 4. *A* **proof system** *for a relation* $\mathcal{R} := \{\mathcal{R}_\lambda \subseteq \mathcal{X}_\lambda \times \mathcal{W}_\lambda\}_{\lambda\in\mathbb{N}}$ *is a pair of interactive machines* $(\mathcal{P}, \mathcal{V})$*, where for* $x \in \mathcal{X}_\lambda$ *and* $w \in \mathcal{W}_\lambda$*, the prover is invoked as* $\mathcal{P}(x, w)$ *and the verifier is invoked as* $\mathcal{V}(x)$*. We let* $\langle \mathcal{P}(x, w); \mathcal{V}(x)\rangle$ *be a random variable that is the verifier's output at the end of the interaction. We let* $\mathsf{trans}(\mathcal{P}(x, w); \mathcal{V}(x))$ *denote a random variable that is the transcript of the interaction.*

- *The proof system* $(\mathcal{P}, \mathcal{V})$ *has* **perfect completeness** *if for all* $(x, w) \in \mathcal{R}_\lambda$

$$\Pr[\langle \mathcal{P}(x, w); \mathcal{V}(x)\rangle = 1] = 1.$$

- *The proof system* $(\mathcal{P}, \mathcal{V})$ *is* **honest verifier zero knowledge***, or HVZK, if there is a* PPT *Sim such that for all* $(x, w) \in \mathcal{R}_\lambda$ *the two distributions*

$$\{Sim(x)\} \qquad \text{and} \qquad \{\mathsf{trans}(\mathcal{P}(x, w); \mathcal{V}(x))\}$$

  are computational indistinguishable. In particular, let $\mathbf{Adv}^{\text{hvzk}}_{\mathcal{A},(\mathcal{P},\mathcal{V})}(\lambda)$ *be the distinguishing advantage for an adversary* $\mathcal{A}$*. This function is negligible for all PPT adversaries* $\mathcal{A}$*.*
- *The proof system* $(\mathcal{P}, \mathcal{V})$ *is an* **argument of knowledge** *if it is perfectly complete, and for every* PPT $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2)$ *there is an expected polynomial time extractor Ext so that the functions*

$$\epsilon_1(\lambda) := \Pr[\langle \mathcal{P}_2(\mathtt{state}); \mathcal{V}(x)\rangle = 1 \ : \ (x, \mathtt{state}) \xleftarrow{\$} \mathcal{P}_1(1^\lambda)]$$
$$\epsilon_2(\lambda) := Pr[(x, w) \in \mathcal{R}_\lambda \ : \ (x, \mathtt{state}) \xleftarrow{\$} \mathcal{P}_1(1^\lambda), \ w \xleftarrow{\$} Ext^{\mathcal{P}_2(\mathtt{state})}(x)]$$

  satisfy

$$\epsilon_2(\lambda) \geq (\epsilon_1(\lambda) - \kappa(\lambda))/q(\lambda), \tag{1}$$

  for some negligible function κ *called the* **knowledge error***, and a polynomial function* q *called the* **extraction tightness***. Here* state *is state data output by* $\mathcal{P}_1$*, and* $Ext^{\mathcal{P}_2(\mathtt{state})}$ *denotes that Ext has oracle access to* $\mathcal{P}_2(\mathtt{state})$ *which is modeled as an "interactive function" [8]. We refer to* $\mathcal{P}_1$ *as an* **instance generator***.*
- *We say that a proof system* $(\mathcal{P}, \mathcal{V})$ *is* **non-interactive** *if the only interaction is a single message* π *from the prover* $\mathcal{P}$ *to the verifier* $\mathcal{V}$*.*
- *We say that the proof system* $(\mathcal{P}, \mathcal{V})$ *is a non-interactive HVZK argument of knowledge in the* **random oracle model** *if* $(\mathcal{P}^H, \mathcal{V}^H)$ *is a proof system that is non-interactive, HVZK, and an argument of knowledge, where* H *is a random oracle.*

A public coin proof system can be made non-interactive using the Fiat-Shamir transform [35]. For some proof systems, this transformation retains the argument of knowledge and HVZK properties in the random oracle model [4]. Implementing the Fiat-Shamir transform in practice is error-prone and it is recommended to use an established implementation to do it (e.g., [30]).

3 Threshold, Accountable, and Private Signatures

In this section, we formalize the notion of threshold, accountable, and private signatures (TAPS). We use n for the total number of signers, and t for the threshold number of required signers. We let $\mathcal{M}$ denote the message space.

The Combiner. When t parties wish to generate a signature on some message m, they send their signature shares to a *Combiner* who uses the t shares to generate a complete signature. Notice that the Combiner will learn the threshold t, which is secret information in our settings. Since the Combiner must be trusted with this private information, we also allow the Combiner to hold a secret key denoted sk_{c}. Secrecy of the Combiner's key is only needed for privacy of the signing quorum. It is not needed for security: if sk_{c} becomes public, an adversary cannot use it to defeat the unforgeability or accountability properties of the scheme. Looking ahead, we will model this by giving sk_{c} to the adversary in the unforgeability and accountability security games, but we keep this key hidden in the privacy game.

The Tracer. A tracing entity is trusted to hold a secret tracing key sk_{t} that allows one to trace a valid signature to the quorum of signers who generated it. Without knowledge of sk_{t}, recovering the quorum should be difficult.

With these parties in mind, let us define the syntax for a TAPS.

Definition 5. *A* **private and accountable threshold signature** *scheme, or* **TAPS**, *is a tuple of five polynomial time algorithms*

$$\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$$

where:

- $\mathit{KeyGen}(1^\lambda, n, t) \to \big(pk, (sk_1, \ldots, sk_n), sk_{\mathrm{c}}, sk_{\mathrm{t}}\big)$*: a probabilistic algorithm that takes as input a security parameter λ, the number of parties n and threshold t. It outputs a public key pk, signer keys $(sk_1, \ldots, sk_n)$, a combiner secret key sk_{c}, and a tracing secret key sk_{t}.*
- $\mathit{Sign}(sk_i, m, C) \to \delta_i$*: a probabilistic algorithm performed by* one *signer who uses its secret key sk_i to generate a signature "share" δ_i on a message m in $\mathcal{M}$. In some constructions it is convenient to allow the signer to know the identity of the members of the signing quorum $C \subseteq [n]$. We provide it as an optional input to Sign.*
- $\mathit{Combine}(sk_{\mathrm{c}}, m, C, \{\delta_i\}_{i\in C}) \to \sigma$*: a probabilistic algorithm that takes as input the Combiner's secret key, a message m, a description of the signing quorum $C \subseteq [n]$ where $|C| = t$, and t valid signature shares by members of C. If the input is valid, the algorithm outputs a* TAPS *signature σ.*
- $\mathit{Verify}(pk, m, \sigma) \to 0/1$*: a deterministic algorithm that verifies the signature σ on a message m with respect to the public key pk.*
- $\mathit{Trace}(sk_{\mathrm{t}}, m, \sigma) \to C/\mathsf{fail}$*: a deterministic algorithm that takes as input the tracer's secret key sk_{t}, along with a message and a signature. The algorithm outputs a set $C \subseteq [n]$, where $|C| \geq t$, or a special message* fail*. If the algorithm outputs a set C, then the set is intended to be a set of signers whose keys must have been used to generate σ. We refer to the entity performing Trace as the* Tracer*.*
- *For correctness we require that for all allowable $1 \leq t \leq n$, for all t-size sets $C \subseteq [n]$, all $m \in \mathcal{M}$, and for $\big(pk, (sk_1, \ldots, sk_n), sk_{\mathrm{c}}, sk_{\mathrm{t}}\big) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t)$ the following two conditions hold:*

$$\Pr\big[\mathit{Verify}\big(pk, m, \mathit{Combine}(sk_{\mathrm{c}}, m, C, \{\mathit{Sign}(sk_i, m, C)\}_{i\in C})\big) = 1\big] = 1$$
$$\Pr\big[\mathit{Trace}\big(sk_{\mathrm{t}}, m, \mathit{Combine}(sk_{\mathrm{c}}, m, C, \{\mathit{Sign}(sk_i, m, C)\}_{i\in C})\big) = C\big] = 1. \tag{2}$$

Remark 1 (signing algorithm vs. signing protocol). In this paper we treat *Sign*() as an algorithm that is run locally by each of the signing parties. However, in some schemes, *Sign* is an interactive protocol between each signing party and the Combiner. Either way, the end result is that the Combiner obtains a list of signature shares $\{\delta_i\}_{i\in C}$, one share from each signer in C. The distinction between a local non-interactive signing algorithm vs. an interactive signing protocol is not relevant to the constructions in this paper.

Remark 2 (distributed key generation). Our syntax assumes a centralized setup algorithm *KeyGen* to generate the signing key shares. However, all our schemes can be adapted to use a decentralized key generation protocol among the signers, the Combiner, and the Tracer. At the end of the protocol every signer knows its secret key, the Combiner knows sk_{c}, the Tracer knows sk_{t}, and pk is public. No other information is known to any party.

Remark 3 (Why use a Tracer?). The Combiner knows which parties contributed signature shares to create a particular signature. A badly designed tracing system could operate as follows: whenever the Combiner constructs a signature, it records the quorum that was used to generate that signature in its database. Later, when a signature needs to be traced, the Combiner could look up the signature in its database and reveal the quorum that generated that signature. If the signature scheme is strongly unforgeable, then one could hope that the only valid signatures in existence are ones generated by an honest Combiner, so that every valid signature can be easily traced with the help of the Combiner. The problem, of course, is that a malicious quorum of signers could collude with the Combiner to generate a valid signature that cannot be traced because the data is not recorded in the database. Or a malicious quorum might delete the relevant entry from the Combiner's database and prevent tracing.

Instead, we require that every valid signature can be traced to the quorum that generated it using the secret tracing key sk_{t}. The tracing key sk_{t} can be kept in a "safety deposit box" and only accessed when tracing is required. The Combiner in a TAPS is *stateless*.

In the next two subsections we define security, privacy, and accountability for a TAPS. The scheme has to satisfy the standard notion of existential unforgeability under a chosen messages attack (EUF-CMA) [44]. In addition, the scheme has to be private and accountable. It is convenient to define unforgeability and accountability in a single game. We define privacy as an additional requirement.

3.1 Unforgeability and Accountability

Like any signature scheme, a TAPS must satisfy the standard notion of unforgeability against a chosen message attack (EUF-CMA). Further, a TAPS scheme should be *accountable*. Informally, this means that a tracer that has the tracing key sk_{t} should output the correct quorum set $C \subseteq [n]$ of signers for a given message-signature pair.

We refer to these simultaneous notions of unforgeability and accountability as *Existential Unforgeability under a Chosen Message Attack with Traceability*. Informally, this notion captures the following unforgeability and accountability properties, subject to restrictions of the chosen message attack:

- Unforgeability: an adversary that controls fewer than t participants cannot construct a valid message-signature pair; and
- Accountability: an adversary that controls t or more corrupt participants cannot construct a valid message-signature pair that traces to at least one honest participant.

We formalize this in the attack game in Figure 1. Let $\mathbf{Adv}^{\mathrm{forg}}_{\mathcal{A},\mathcal{S}}(\lambda)$ be the probability that adversary $\mathcal{A}$ wins the game of Figure 1 against the TAPS scheme $\mathcal{S}$.

Definition 6 (accountable TAPS). *A* TAPS *scheme $\mathcal{S}$ is **unforgeable and accountable** if for all probabilistic polynomial time adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, the function $\mathbf{Adv}^{\mathrm{forg}}_{\mathcal{A},\mathcal{S}}(\lambda)$ is a negligible function of λ.*

Our game in Figure 1 captures both unforgeability (EUF-CMA) for a threshold signature scheme as well as accountability. During the game the adversary obtains the secret keys of parties in C and obtains signature shares for m' from parties in C'. The adversary should be unable to produce a valid signature σ' that causes the tracing algorithm to fail, or causes the tracing algorithm to blame a signing party outside of $C \cup C'$. This captures the accountability property. To see why this implies unforgeability, suppose the adversary $\mathcal{A}$ obtains fewer than threshold t signature shares for m', meaning that $|C \cup C'| < t$. Yet, the adversary is able to produce a valid signature σ' that causes the tracing algorithm to blame some quorum C_{t}. By definition of *Trace* we know that $|C_{\mathrm{t}}| \geq t$ and therefore C_{t} cannot be contained in $C \cup C'$. Therefore the adversary succeeds in blaming an honest party, and consequently $\mathcal{A}$ wins the game. Hence, if the adversary cannot win the game in Figure 1, the scheme must be unforgeable.

Remark 4. Definition 6 captures unforgeability, but not strong unforgeability, where the adversary should be unable to generate a new signature on a previously signed message. However, our definition of privacy (Definition 7) in combination with Definition 6, imply strong unforgeability.

Unforgeability and accountability attack game:

$(n, t, C, \texttt{state}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$; where $t \in [n]$ and $C \subseteq [n]$ // $\mathcal{A}_0$ outputs n, t and C (no size bound on C)

$(pk, \{sk_1, \ldots, sk_n\}, sk_c, sk_t) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t)$ // generate keys using n and t

$(m', \sigma') \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot)}(pk, \{sk_i\}_{i \in C}, sk_c, sk_t, \texttt{state})$ // $\mathcal{A}_1$ receives secret keys for all of C, // as well as the tracing and combiner's secret keys

where $\mathcal{O}(C_j, m_j)$ returns the sig. shares $\{\mathit{Sign}(sk_i, m_j, C_j)\}_{i \in C_j}$ // $\mathcal{A}_1$ can request signature shares for m_j

winning condition:

let (C_1, m_1), (C_2, m_2), ... be $\mathcal{A}_1$'s queries to $\mathcal{O}$

let $C' \leftarrow \bigcup C_j$, union over all queries to $\mathcal{O}(C_j, m')$, // collect all signers that signed m'

if no such queries, set $C' \leftarrow \emptyset$ // if no $\mathcal{O}$-queries for m', then $C' = \emptyset$

let $C_t \leftarrow \mathit{Trace}(sk_t, m', \sigma')$ // trace the forgery (m', σ')

output 1 if $\mathit{Verify}(pk, m', \sigma') = 1$ and either // $\mathcal{A}$ wins if someone outside of $(C \cup C')$ is blamed,

$C_t \not\subseteq (C \cup C')$ or $C_t = \mathsf{fail}$ // or if tracing fails

Fig. 1. Game defining the advantage of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ to produce a valid forgery against a TAPS scheme $\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$ with respect to a security parameter λ.

3.2 Privacy

Next, we define privacy for a TAPS. Privacy for a threshold signature scheme is often defined by requiring that a threshold signature on a message m be indistinguishable from a signature on m generated by some standard (non-threshold) signature scheme [42]. This property ensures that a threshold signature reveals nothing about the threshold and the quorum that produced the signature.

A TAPS may not be derived from a non-threshold signature scheme, so this definitional approach does not work well in our setting. Instead, we define privacy as an intrinsic property of the TAPS. Our definition of privacy applies equally well to a private threshold signature (PTS) scheme.

We impose two privacy requirements:

- **Privacy against the public:** A party who only has pk and sees a sequence of message-signature pairs, learns nothing about the threshold t or the set of signers that contributed to the creation of those signatures.
- **Privacy against signers:** The set of all signers working together, who also have pk (but not sk_c or sk_t), and see a sequence of message-signature pairs, cannot determine which signers contributed to the creation of those signatures. Note that t is not hidden in this case since the set of all signers knows the threshold.

These properties are captured by the games in Figure 2 and Figure 3 respectively.

Let W be the event that the game in Figure 2 outputs 1. Similarly, let W' be the event that the game in Figure 3 outputs 1. We define the two advantage functions for an adversary $\mathcal{A}$ against the scheme $\mathcal{S}$, as a function of the security parameter λ:

$$\mathbf{Adv}^{\mathrm{priv1}}_{\mathcal{A},\mathcal{S}}(\lambda) := \left|2\Pr[W] - 1\right| \quad \text{and} \quad \mathbf{Adv}^{\mathrm{priv2}}_{\mathcal{A},\mathcal{S}}(\lambda) := \left|2\Pr[W'] - 1\right|.$$

Definition 7 (Privacy for a TAPS scheme). *A TAPS scheme is **private** if for all probabilistic polynomial time public adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, the functions $\mathbf{Adv}^{\mathrm{priv1}}_{\mathcal{A},\mathcal{S}}(\lambda)$ and $\mathbf{Adv}^{\mathrm{priv2}}_{\mathcal{A},\mathcal{S}}(\lambda)$ are negligible functions of λ.*

Let us examine the definition in more detail. Privacy against the public for a TAPS is defined using the game in Figure 2. The adversary chooses two thresholds t_0 and t_1 in $[n]$ and is given a public key pk for one of these thresholds. The adversary then issues a sequence of signature queries to a signing oracle $\mathcal{O}_1$, where each signature query includes a message m and two quorums C_0 and C_1. The adversary gets

The game defining privacy against the public:

$b \xleftarrow{\$} \{0,1\}$

$(n, t_0, t_1, \texttt{state}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$ where $t_0, t_1 \in [n]$ // $\mathcal{A}_0$ outputs n and two thresholds t_0, t_1

$(pk, \{sk_1, \ldots, sk_n\}, sk_c, sk_t) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t_b)$ // generate keys using n and t_b

$b' \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\cdot,\cdot,\cdot),\ \mathcal{O}_2(\cdot,\cdot)}(pk, \texttt{state})$

output $(b = b')$

where $\mathcal{O}_1(C_0, C_1, m)$ returns $\sigma \xleftarrow{\$} \mathit{Combine}(sk_c, m, C_b, \{\mathit{Sign}(sk_i, m, C_b)\}_{i \in C_b})$ // sign using C_b
for $C_0, C_1 \subseteq [n]$ with $|C_0| = t_0$ and $|C_1| = t_1$,

and where $\mathcal{O}_2(m, \sigma)$ returns $\mathit{Trace}(sk_t, m, \sigma)$. // trace (m, σ)

Restriction: if σ is obtained from a query $\mathcal{O}_1(\cdot, \cdot, m)$, then $\mathcal{O}_2$ is never queried at (m, σ).

Fig. 2. The game used to define privacy against the public for an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against a TAPS scheme $\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$ with respect to a security parameter λ.

The game defining privacy against signers:

$b \xleftarrow{\$} \{0,1\}$

$(n, t, \texttt{state}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$ where $t \in [n]$ // $\mathcal{A}_0$ outputs n and t

$(pk, \{sk_1, \ldots, sk_n\}, sk_c, sk_t) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t)$ // generate keys using n and t

$b' \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\cdot,\cdot,\cdot),\ \mathcal{O}_2(\cdot,\cdot)}(pk, \{sk_1, \ldots, sk_n\}, \texttt{state})$ // $\mathcal{A}_1$ issues signature and trace queries

output $(b = b')$

where $\mathcal{O}_1(C_0, C_1, m)$ returns $\sigma \xleftarrow{\$} \mathit{Combine}(sk_c, m, C_b, \{\mathit{Sign}(sk_i, m, C_b)\}_{i \in C_b})$ // sign using C_b
for $C_0, C_1 \subseteq [n]$ with $|C_0| = |C_1| = t$,

and where $\mathcal{O}_2(m, \sigma)$ returns $\mathit{Trace}(sk_t, m, \sigma)$. // trace (m, σ)

Restriction: if σ is obtained from a query $\mathcal{O}_1(\cdot, \cdot, m)$, then $\mathcal{O}_2$ is never queried at (m, σ).

Fig. 3. The game used to define privacy against signers for an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against a TAPS scheme $\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$ with respect to a security parameter λ. Here, $\mathcal{A}_1$ is granted knowledge of all signing keys $sk_1, \ldots, sk_n$.

back a signature generated using either the left or the right quorum. We also give the adversary access to a restricted tracing oracle $\mathcal{O}_2$ that will trace a valid message-signature pair. The adversary should be unable to determine whether the sequence of signatures it saw were with respect to the left or the right sequence of quorums.

Our definition of privacy ensures that the threshold t is hidden, but we do not try to hide the number of signers n because there is no need to: one can covertly inflate n to some upper bound by generating superfluous signing keys.

Privacy against signers is defined using the game in Figure 3. This game is the same as in Figure 2, however here the adversary chooses the threshold t, and is given *all* the signing keys. Again, the adversary should be unable to determine if a signing oracle $\mathcal{O}_1$ that takes two quorums C_0 and C_1, responds using the left or the right quorum. As before, the adversary has access to a restricted tracing oracle $\mathcal{O}_2$. Note that we do not aim to prevent signers from recognizing a signature that was generated with their help, as discussed in Section 6.

Remark 5 (Randomized signing). The privacy games in Figures 2 and 3 require that signature generation be a randomized process: calling $\mathit{Combine}(sk_c, m, C, \{\mathit{Sign}(sk_i, m, C)\}_{i \in C})$ with the same arguments m and C twice must result in different signatures, with high probability. Otherwise, the adversary could

trivially win these games: it would query $\mathcal{O}_1$ twice, once as $\mathcal{O}_1(C_0, C_1, m)$ and again as $\mathcal{O}_1(C_0, C_1', m)$, for suitable quorums C_0, C_1, C_1' where $C_1 \neq C_1'$. It would then check if the resulting signatures are the same. If so, it learns that $b = 0$, and if not it learns that $b = 1$. For this reason, if a scheme satisfies Definition 7, then the output of $\mathit{Combine}(sk_c, m, C, \{\mathit{Sign}(sk_i, m, C)\}_{i \in C})$ must be sampled from some high entropy distribution.

3.3 Accountable Threshold Schemes (ATS)

For completeness, we note that the standard notions of private threshold signatures (PTS) and accountable threshold signatures (ATS) are special cases of a TAPS. We review these concepts in the next two definitions.
To obtain an ATS we impose two syntactic requirements on a TAPS scheme:

- In an ATS, the tracing key is publicly known, meaning that anyone can trace a valid message-signature pair to the quorum that participated in generating it. We capture this by requiring that the TAPS tracing key sk_t is equal to the public key pk.
- In an ATS, the Combiner is not a trusted party and cannot hold secrets. We capture this by requiring that the Combiner's secret key sk_c is also equal to the public key pk.

For clarity, whenever we make use of an ATS, we will drop sk_t and sk_c as explicit outputs of the TAPS key generation algorithm.

Definition 8. *An* **accountable threshold signature scheme**, *or an* **ATS**, *is a special case of a* TAPS, *where the tracing key sk_t and the Combiner key sk_c are both equal to the public key pk. The scheme is said to be* **secure** *if it is accountable and unforgeable as in Definition 6.*

Notice that there is no privacy requirement in Definition 8.

Remark 6. As mentioned in the introduction, an ATS scheme is closely related to the concept of an *accountable multi-signature scheme* (ASM) [53,10,18,16,6,56]. One can construct an ATS from an ASM by including a threshold t in the ASM public key. The ASM verification algorithm is modified to ensure that at least t signers represented in pk signed the message.

Next, we define a private threshold signature scheme, or a PTS. In the literature, a private threshold signature scheme is simply called a *threshold signature scheme*. However, ATS and PTS are equally important concepts, and we therefore add an explicit adjective to clarify which threshold signature concept we are using.

Definition 9. *A* **private threshold signature scheme**, *or a* **PTS**, *is a special case of a* TAPS, *where the Trace algorithm always returns* fail, *and the correctness requirement for a* TAPS *in Definition 5 is modified to remove the requirement on Trace in Eq. (2). The scheme is said to be* **secure** *if it is private as in Definition 7, and unforgeable as in Definition 6 with one modification: the adversary wins if the forgery is valid and $|C \cup C'| < t$.*

The modification of Definition 6 reduces the accountability and unforgeability game in Definition 6 to a pure unforgeability game under a chosen message attack, ignoring accountability. Interestingly, this game captures a security notion related to *dual-parameter* threshold security [59]. If one puts a further bound requiring $|C| < t' < t$ in Figure 1, for some parameter t', then one obtains the usual definition of dual-parameter threshold security from [59]. Other notions of unforgeability for threshold signatures were considered in [11].

4 A Generic Construction via an Encrypted ATS

We next turn to constructing a TAPS scheme. In this section we present a generic construction from a secure ATS scheme. The generic TAPS construction makes use of five building blocks:

- a secure accountable threshold signature (ATS) scheme as in Definition 8, namely $\mathcal{ATS} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$;

- a semantically secure public-key encryption scheme $\mathcal{PKE} = (\mathit{KeyGen}, \mathit{Encrypt}, \mathit{Decrypt})$ as in Definition 1 whose message space is the space of signatures output by the ATS signing algorithm;
- a binding and hiding commitment scheme $\mathcal{COM} = (\mathit{Commit}, \mathit{Verify})$, where algorithm $\mathit{Commit}(m, r)$ outputs a commitment to a message m using a random nonce $r \xleftarrow{\$} \mathcal{R}$, as in Definition 2;
- a strongly unforgeable signature scheme $\mathcal{SIG} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Verify})$ as in Definition 3;
- a non-interactive zero knowledge argument of knowledge (P, V), possibly constructed in the random oracle model using the Fiat-Shamir transform.

Recall that our definition of semantic security in Section 2 ensures that the encryption scheme $\mathcal{PKE}$ is length-hiding: the encryption of messages m_0 and m_1 of different lengths are indistinguishable.

The generic TAPS **scheme.** The generic TAPS scheme $\mathcal{S}$ is shown in Figure 4. In our construction, a TAPS signature on a message m is a triple $\sigma = (ct, \pi, tg)$, where (i) ct is a public key encryption of an ATS signature σ_{m} on m, encrypted using the tracing public key pk_{t}, (ii) π is a zero-knowledge proof that the decryption of ct is a valid ATS signature on m, and (iii) tg is the Combiner's signature on (m, ct, π). The reason for the Combiner's signature is explained in Remark 7.

Recall that an ATS public key can reveal the threshold t in the clear, which would violate the TAPS privacy requirements. As such, the TAPS public key cannot include the ATS public key in the clear. Instead, the TAPS public key only contains a hiding *commitment* to the ATS public key.

Correctness. The scheme is correct if the underlying ATS scheme, commitment scheme, encryption scheme, signature scheme, and proof system are correct.

Efficiency. When using a succinct commitment scheme, the public key is quite short; its length depends only on the security parameter. When using a zk-SNARK [14] for the proof system, the signature overhead over the underlying ATS signature is quite short; its length depends only on the security parameter. Moreover, signature verification time is dominated by the SNARK proof verification, which is at most logarithmic in the total number of signing parties n.

However, the Combiner's work in this scheme is substantial because it needs to generate a zk-SNARK proof for a fairly complex statement. In addition, zk-SNARK proof systems rely on strong complexity assumptions for security [43]. To address these issues, we construct in the next section more efficient TAPS schemes whose security relies on DDH in the random oracle model, a much simpler assumption.

Security, privacy, and accountability. We next turn to proving that the generic scheme is secure, private, and accountable.

Theorem 1. *The generic* TAPS *scheme $\mathcal{S}$ in Figure 4 is unforgeable, accountable, and private, assuming that the underlying accountable threshold scheme $\mathcal{ATS}$ is secure, the encryption scheme $\mathcal{PKE}$ is semantically secure, the non-interactive proof system (P, V) is an argument of knowledge and HVZK, the commitment scheme $\mathcal{COM}$ is hiding and binding, and the signature scheme $\mathcal{SIG}$ is strongly unforgeable.*

We provide concrete security bounds in the lemmas below. First, let us explain the need for the Combiner's signature in Step 4 of $\mathcal{S}.\mathit{Combine}$.

Remark 7. Observe that the privacy games in Figures 2 and 3 give the adversary a tracing oracle for any message-signature pair of its choice. In the context of our construction this enables the adversary to mount a chosen ciphertext attack on the encryption scheme $\mathcal{PKE}$. Yet, Theorem 1 only requires that $\mathcal{PKE}$ be semantically secure, not chosen ciphertext secure. The need for a weak security requirement on $\mathcal{PKE}$ will become important in the next section where we construct more efficient TAPS schemes. To secure against the chosen ciphertext attack, we rely on the Combiner's signature included in every TAPS signature. It ensures that the adversary cannot call the tracing oracle with anything other than a TAPS signature output by the Combiner.

We now prove Theorem 1. The proof is captured in the following three lemmas.

- $\underline{\mathcal{S}.KeyGen}(1^\lambda, n, t)$:

  1: $\big(pk',\ (sk_1, \ldots, sk_n)\big) \xleftarrow{\$} \mathcal{ATS}.KeyGen(1^\lambda, n, t)$

  2: $r_{pk} \xleftarrow{\$} \mathcal{R}_\lambda$ and $\mathsf{com}_{pk} \leftarrow \mathcal{COM}.Commit(pk', r_{pk})$

  3: $(pk_{\mathrm{t}}, sk'_{\mathrm{t}}) \xleftarrow{\$} \mathcal{PKE}.KeyGen(1^\lambda)$

  4: $(pk_{\mathrm{cs}}, sk_{\mathrm{cs}}) \xleftarrow{\$} \mathcal{SIG}.KeyGen(1^\lambda)$ // Combiner's signing key

  5: $sk_{\mathrm{t}} \leftarrow (pk', sk'_{\mathrm{t}}, pk_{\mathrm{cs}})$ // the secret tracing key

  6: $sk_{\mathrm{c}} \leftarrow (pk', pk_{\mathrm{t}}, sk_{\mathrm{cs}}, t, \mathsf{com}_{pk}, r_{pk})$ // Combiner's secret key

  7: $pk \leftarrow (\mathsf{com}_{pk},\ pk_{\mathrm{t}},\ pk_{\mathrm{cs}})$

  8: output $\big(pk,\ (sk_1, \ldots, sk_n),\ sk_{\mathrm{c}},\ sk_{\mathrm{t}}\big)$

- $\underline{\mathcal{S}.Sign}(sk_i, m, C) \to \delta_i$: output $\delta_i \xleftarrow{\$} \mathcal{ATS}.Sign(sk_i, m, C)$.
  Here $C \subseteq [n]$ is a set of size t of participating signers. Recall that in some schemes $\mathcal{ATS}.Sign$ is an algorithm run by the signing parties, while in other schemes $\mathcal{ATS}.Sign$ is an interactive protocol between the Combiner and the signing parties. Either way, the end result is that the Combiner obtains signature shares $\{\delta_i\}_{i\in C}$.

- $\underline{\mathcal{S}.Combine}(sk_{\mathrm{c}}, m, C, \{\delta_i\}_{i\in C}) \to \sigma$: with $sk_{\mathrm{c}} = (pk', pk_{\mathrm{t}}, sk_{\mathrm{cs}}, t, \mathsf{com}_{pk}, r_{pk})$, the Combiner does

  1: $\sigma_{\mathrm{m}} \xleftarrow{\$} \mathcal{ATS}.Combine(pk', m, C, \{\delta_i\}_{i\in C})$, if fail then output fail and stop

  2: $ct \leftarrow \mathcal{PKE}.Encrypt(pk_{\mathrm{t}}, \sigma_{\mathrm{m}}; r)$, where r is a fresh nonce

  3: use the prover P to generate a proof π for the relation:

$$\mathcal{R}\Big((\mathsf{com}_{pk}, pk_{\mathrm{t}}, m, ct)\ ;\ (\sigma_{\mathrm{m}}, r, r_{pk}, pk')\Big) = \text{true} \quad \text{iff} \quad \left\{ \begin{array}{l} ct = \mathcal{PKE}.Encrypt(pk_{\mathrm{t}}, \sigma_{\mathrm{m}}; r), \\ \mathcal{ATS}.Verify(pk', m, \sigma_{\mathrm{m}}) = 1, \\ \mathcal{COM}.Verify(pk', r_{pk}, \mathsf{com}_{pk}) = 1 \end{array} \right\} \tag{3}$$

  4: $tg \xleftarrow{\$} \mathcal{SIG}.Sign\big(sk_{\mathrm{cs}}, (m, ct, \pi)\big)$ // sign with Combiner's signing key

  5: output the TAPS signature $\sigma \leftarrow (ct, \pi, tg)$

- $\underline{\mathcal{S}.Verify}\Big(pk = (\mathsf{com}_{pk}, pk_{\mathrm{t}}, pk_{\mathrm{cs}}),\ m,\ \sigma = (ct, \pi, tg)\Big) \to \{0, 1\}$: accept if
  - $\mathcal{SIG}.Verify\big(pk_{\mathrm{cs}},\ (m, ct, \pi),\ tg\big) = 1$, and
  - π is a valid proof for the relation $\mathcal{R}$ in (3) with respect to the statement $(\mathsf{com}_{pk}, pk_{\mathrm{t}}, m, ct)$.

- $\underline{\mathcal{S}.Trace}\Big(sk_{\mathrm{t}} = (pk', sk'_{\mathrm{t}}, pk_{\mathrm{cs}}),\ m,\ \sigma = (ct, \pi, tg)\Big) \to C$:

  1: if $\mathcal{SIG}.Verify\big(pk_{\mathrm{cs}}, (m, ct, \pi), tg\big) \neq 1$, output fail and stop

  2: set $\sigma_{\mathrm{m}} \leftarrow \mathcal{PKE}.Decrypt(sk'_{\mathrm{t}}, ct)$, if fail then output fail and stop

  3: otherwise, output $\mathcal{ATS}.Trace(pk', m, \sigma_{\mathrm{m}})$

Fig. 4. The generic TAPS scheme $\mathcal{S}$

Lemma 1. *The generic* TAPS *scheme $\mathcal{S}$ is unforgeable and accountable, as in Definition 6, assuming the accountable threshold scheme $\mathcal{ATS}$ is secure, the non-interactive proof system (P, V) is an argument of knowledge, and the commitment scheme is binding. Concretely, for every adversary $\mathcal{A}$ that attacks $\mathcal{S}$ there exists adversaries $\mathcal{B}_1, \mathcal{B}_2$, that run in about the same time as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\text{forg}}_{\mathcal{A},\mathcal{S}}(\lambda) \leq \left(\mathbf{Adv}^{\text{forg}}_{\mathcal{B}_1,\mathcal{ATS}}(\lambda) + \mathbf{Adv}^{\text{bind}}_{\mathcal{B}_2,\mathcal{COM}}(\lambda)\right) \cdot q(\lambda) + \kappa(\lambda) \tag{4}$$

where κ and q are the knowledge error and tightness of the proof system from Definition 4.

We provide the proof of Lemma 1 in Appendix A.

Lemma 2. *The generic* TAPS *scheme $\mathcal{S}$ is private against the public assuming the non-interactive proof system (P, V) is HVZK, the public-key encryption scheme $\mathcal{PKE}$ is semantically secure, the commitment scheme $\mathcal{COM}$ is hiding, and the signature scheme $\mathcal{SIG}$ is strongly unforgeable. Concretely, for every adversary $\mathcal{A}$ that attacks $\mathcal{S}$ there exist adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, that run in about the same time as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\text{priv1}}_{\mathcal{A},\mathcal{S}}(\lambda) \leq 2\Big(\mathbf{Adv}^{\text{eufcma}}_{\mathcal{B}_1,\mathcal{SIG}}(\lambda) + \mathbf{Adv}^{\text{indcpa}}_{\mathcal{B}_2,\mathcal{PKE}}(\lambda) + Q \cdot \mathbf{Adv}^{\text{hvzk}}_{\mathcal{B}_3,(P,V)}(\lambda) + \epsilon(\lambda)\Big) \tag{5}$$

where $\epsilon(\lambda)$ is the hiding statistical distance of the commitment scheme $\mathcal{COM}$ and Q is the number of signature queries from $\mathcal{A}$.

We provide the proof of Lemma 2 in Appendix B.

Lemma 3. *The generic* TAPS *scheme $\mathcal{S}$ is private against signers assuming the non-interactive proof system (P, V) is HVZK, the public-key encryption scheme $\mathcal{PKE}$ is semantically secure, and the signature scheme $\mathcal{SIG}$ is strongly unforgeable. Concretely, for every adversary $\mathcal{A}$ that attacks $\mathcal{S}$ there exist adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, that run in about the same time as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\text{priv2}}_{\mathcal{A},\mathcal{S}}(\lambda) \leq 2\Big(\mathbf{Adv}^{\text{eufcma}}_{\mathcal{B}_1,\mathcal{SIG}}(\lambda) + \mathbf{Adv}^{\text{indcpa}}_{\mathcal{B}_2,\mathcal{PKE}}(\lambda) + Q \cdot \mathbf{Adv}^{\text{hvzk}}_{\mathcal{B}_3,(P,V)}(\lambda)\Big). \tag{6}$$

The proof of Lemma 3 is almost identical to the proof of Lemma 2 and is omitted.

5 An Efficient TAPS from Schnorr Signatures

In this section we construct a secure TAPS in the random oracle model, based on the Schnorr signature scheme. The construction is far more efficient than applying the generic construction from the previous section to a Schnorr ATS. We obtain this improvement by taking advantage of the algebraic properties of the Schnorr signature scheme to vastly simplify the zero knowledge statement that the Combiner needs to prove when making a signature.

The construction makes use of a group $\mathbb{G}$ of prime order q in which the Decision Diffie-Hellman problem is hard. Let g and h be independent generators of $\mathbb{G}$, so that the discrete log of h base g is unknown. We also require a hash function $H : \mathcal{PK} \times \mathbb{G} \times \mathcal{M} \to \mathbb{Z}_q$ that will be modeled as a random oracle, where $\mathcal{PK}$ is a space of public keys.

5.1 A review of the Schnorr ATS schemes

Let us first review the (uncompressed) Schnorr signature scheme [58]:

- *KeyGen*(λ): $sk \xleftarrow{\$} \mathbb{Z}_q$, $pk \leftarrow g^{sk}$, output (sk, pk).
- *Sign*(sk, m): $r \xleftarrow{\$} \mathbb{Z}_q$, $R \leftarrow g^r$, $c \leftarrow H(pk, R, m) \in \mathbb{Z}_q$, $z \leftarrow r + sk \cdot c \in \mathbb{Z}_q$, output $\sigma \leftarrow (R, z)$.
- *Verify*(pk, m, σ): compute $c \leftarrow H(pk, R, m) \in \mathbb{Z}_q$ and accept if $g^z = pk^c \cdot R$.

Our Schnorr TAPS builds upon an existing Schnorr accountable threshold signature (ATS), such as [53,52,56][3]. Using our terminology, these ATS schemes operate as follows:

[3] Technically, these are multisignature schemes, but as noted in Remark 6, they can easily be made into an ATS.

- *KeyGen*(λ, n, t): Choose $sk_1, \ldots, sk_n \xleftarrow{\$} \mathbb{Z}_q$ and set $pk_i \leftarrow g^{sk_i}$ for $i \in [n]$.
  Set $pk \leftarrow (t, pk_1, \ldots, pk_n)$ and $sk \leftarrow (sk_1, \ldots, sk_n)$. Output (pk, sk).
  In an ATS, the Combiner key sk_c and the tracing key sk_t are equal to pk.
- *Sign*(sk_i, m, C): An interactive protocol between the Combiner and signer i. At the end of the protocol the Combiner has $\delta_i = (R_i, z_i) \in \mathbb{G} \times \mathbb{Z}_q$, where (R_i, z_i) satisfies $g^{z_i} = pk_i^c \cdot R_i$ for $c \leftarrow H(pk, R, m) \in \mathbb{Z}_q$. Here $R \in \mathbb{G}$ is defined[4] as $R := \prod_{i \in C} R_i$. This R is obtained from the Combiner's interaction with all the signers participating in the current signature process.
- *Combine*$(pk, m, C, \{\delta_i\}_{i \in C})$: Abort if $|C| \neq t$. Parse δ_i as $\delta_i = (R_i, z_i)$,
  set $z \leftarrow \sum_{i \in C} z_i \in \mathbb{Z}_q$ and $R \leftarrow \prod_{i \in C} R_i \in \mathbb{G}$. Output $\sigma \leftarrow (R, z, C)$.
  Then (R, z) is a valid Schnorr signature on m with respect to the public key $pk_C := \prod_{i \in C} pk_i$.
- *Verify*(pk, m, σ): parse $pk = (t, pk_1, \ldots, pk_n)$ and $\sigma = (R, z, C)$. Accept if $|C| = t$ and the Schnorr verification algorithm accepts the triple (pk_C, m, σ') where $\sigma' \leftarrow (R, z)$ and $pk_C \leftarrow \prod_{i \in C} pk_i$. Here the challenge c is computed as $c \leftarrow H(pk, R, m) \in \mathbb{Z}_q$ and the algorithm accepts if $|C| = t$ and $g^z = pk_C^c \cdot R$.
- *Trace*(pk, m, σ): parse $\sigma = (R, z, C)$, run *Verify*(pk, m, σ), the verification algorithm from the previous bullet, and if valid, output C; else output fail.

The Schnorr ATS papers [53,52,56] describe different ways to instantiate the *Sign* protocol. They prove security of the resulting Schnorr ATS scheme using differing security models. Here we treat the *Sign* protocol as a black box, and rely on the following assumption.

Assumption 1 *The Schnorr ATS outlined above is a secure ATS scheme, as in Definition 8.*

5.2 An efficient Schnorr TAPS

We next construct our Schnorr-based TAPS scheme. If we were to follow the generic construction from Section 4, the combiner would encrypt the entire Schnorr signature (R, z), and would need to produce a zero knowledge proof for a complicated relation. In particular, it would need to prove that an encrypted Schnorr signature is valid, which is difficult to prove in zero knowledge efficiently. However, observe that in the public's view, R is a product of random elements in $\mathbb{G}$, and as such, is independent of the quorum set C. Therefore, R can be revealed in the TAPS signature in the clear without compromising the privacy of C in the public's view. Even an adversary who has all the signing keys learns nothing about C from R. We only need to encrypt the quantity $z \in \mathbb{Z}_q$. The challenge then is to develop an efficient zero knowledge proof that the cleartext R and an encrypted z are a valid Schnorr signature with respect to an encrypted quorum set C.

The scheme. Our Schnorr TAPS is built from any Schnorr ATS that operates as described in Section 5.1 and satisfies Assumption 1. In addition, we use a single-party (non-threshold) signature scheme $\mathcal{SIG} = (KeyGen, Sign, Verify)$. The complete TAPS scheme is presented in Figure 5. The combine algorithm in Step 4 generates a zero-knowledge proof for the relation $\mathcal{R}_S$ in Figure 6. We present two efficient proof systems for this relation in Sections 5.3 and 5.4.

In Step 4 of the tracing algorithm there is a need to find a set $C \subseteq [n]$ of size t that satisfies a certain property. If n is logarithmic in the security parameter, then this set C can be found by exhaustive search over all t-size subsets of $[n]$. For larger n, we explain how to find C efficiently in Sections 5.3 and 5.4.

Correctness. The scheme is correct assuming the Schnorr ATS scheme, the signature scheme $\mathcal{SIG}$, and proof system for $\mathcal{R}_S$ are correct.

Security. We next prove security, privacy, and accountability.

Theorem 2. *The Schnorr* TAPS *scheme is unforgeable, accountable, and private, assuming that the underlying Schnorr ATS is secure (Assumption 1), the signature scheme $\mathcal{SIG}$ is strongly unforgeable, DDH holds in $\mathbb{G}$, and the non-interactive proof system (P, V) for $\mathcal{R}_S$ is an argument of knowledge and HVZK.*

[4] In some Schnorr ATS schemes (e.g., [56]) this R is defined as $R := \prod_{i \in C} R_i^{\gamma_i}$, for public scalars $\{\gamma_i \in \mathbb{Z}_q\}_{i \in C}$. We assume that all these scalars are set to 1, but our constructions can easily accommodate any scalars.

- $\mathcal{S}.KeyGen(\lambda, n, t)$: using the independent generators g and h of $\mathbb{G}$ do:

1: Run the Schnorr ATS *KeyGen* procedure from Section 5.1.
That is, choose $sk_1, \ldots, sk_n \xleftarrow{\$} \mathbb{Z}_q$ and set $pk_i \leftarrow g^{sk_i}$ for $i \in [n]$. Set $pk' \leftarrow (pk_1, \ldots, pk_n)$.

2: Encrypt t with ElGamal: $\psi \xleftarrow{\$} \mathbb{Z}_q$ and $(T_0, T_1) \leftarrow (g^{\psi},\ g^t h^{\psi})$

3: Generate $(sk_{\text{cs}}, pk_{\text{cs}}) \xleftarrow{\$} \mathcal{SIG}.KeyGen(\lambda)$

4: Sample $sk_{\text{e}} \xleftarrow{\$} \mathbb{Z}_q$ and set $pk_{\text{t}} \leftarrow g^{sk_{\text{e}}} \in \mathbb{G}$

5: $pk \leftarrow (pk', pk_{\text{t}}, pk_{\text{cs}}, T_0, T_1)$ // the verifier's public key

6: $sk_{\text{c}} \leftarrow (pk, sk_{\text{cs}}, t, \psi)$ // the combiner's secret key

7: $sk_{\text{t}} \leftarrow (pk, sk_{\text{e}}, t)$ // the tracing secret key

8: Output $\big(pk,\ (sk_1, \ldots, sk_n),\ sk_{\text{c}},\ sk_{\text{t}}\big)$

- $\mathcal{S}.Sign(sk_i, m, C)$: Run the Schnorr ATS *Sign* procedure from Section 5.1 so that the Combiner obtains a signature share $\delta_i \xleftarrow{\$} (R_i, z_i) \in \mathbb{G} \times \mathbb{Z}_q$.

- $\mathcal{S}.Combine(sk_{\text{c}}, m, C, \{\delta_i\}_{i \in C})$: With $|C| = t$ and $\delta_i = (R_i, z_i)$, the coordinator does:

1: $R \leftarrow \prod_{i \in C} R_i$, $z \leftarrow \sum_{i \in C} z_i \in \mathbb{Z}_q$, $c \leftarrow H(pk, R, m) \in \mathbb{Z}_q$ // then $g^z = \big[\prod_{i \in C} pk_i\big]^c \cdot R$

2: Encrypt z with ElGamal: $\rho \xleftarrow{\$} \mathbb{Z}_q$, $ct := (c_0, c_1) \leftarrow (g^{\rho},\ g^z\, pk_{\text{t}}^{\rho})$.

3: Set $(b_1, \ldots, b_n) \in \{0,1\}^n$, such that $b_i = 1$ iff $i \in C$ // then $g^z = \Big[\prod_{i=1}^{n} (pk_i)^{b_i}\Big]^c \cdot R$

4: Generate a zero knowledge proof π for the relation $\mathcal{R}_S$ listed in Figure 6. We present two efficient non-interactive proof systems for this relation in Sections 5.3 and 5.4.

5: $tg \xleftarrow{\$} \mathcal{SIG}.Sign\big(sk_{\text{cs}}, (m, R, ct, \pi)\big)$ // sign with Combiner's key

6: Output the TAPS signature $\sigma \leftarrow (R, ct, \pi, tg)$.

- $\mathcal{S}.Verify(pk,\ m,\ \sigma)$: Let $\sigma = (R, ct, \pi, tg)$ where $ct = (c_0, c_1)$.
Parse $pk = (pk', pk_{\text{t}}, pk_{\text{cs}}, T_0, T_1)$ and set $c \leftarrow H(pk, R, m)$. Accept if:
  - $\mathcal{SIG}.Verify\big(pk_{\text{cs}}, (m, R, ct, \pi), tg\big) = 1$, and
  - π is a valid proof for the relation $\mathcal{R}_S$ in Figure 6
    with respect to the statement $(g, h, pk', pk_{\text{t}}, T_0, T_1, R, c, ct = (c_0, c_1))$.

- $\mathcal{S}.Trace(sk_{\text{t}}, m, \sigma)$: Parse $sk_{\text{t}} = \big(pk, sk_{\text{e}}, t\big)$ where $pk = (pk', pk_{\text{t}}, pk_{\text{cs}}, T_0, T_1)$ and $pk' = (pk_1, \ldots, pk_n)$.

1: Abort if $\mathcal{S}.Verify(pk, m, \sigma)$ rejects. // ensure that (m, σ) is a valid signature

2: Parse σ as (R, ct, π, tg) and $ct = (c_0, c_1)$. Set $c \leftarrow H(pk, R, m)$.

3: ElGamal decrypt $ct = (c_0, c_1)$ as $g^{(z')} \leftarrow c_1 / {c_0}^{sk_{\text{e}}} \in \mathbb{G}$.

4: Find a set $C \subseteq [n]$, where $|C| = t$ and $g^{(z')} = R \cdot (\prod_{i \in C} pk_i)^c$. This equality implies that (R, z') is a valid Schnorr signature on m with respect to the public key $pk_C \leftarrow \prod_{i \in C} pk_i$.

5: If such a set $C \subseteq [n]$ is found, output C. Otherwise, output fail.

Fig. 5. The Schnorr TAPS scheme

$$\mathcal{R}_S = \Big\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \ ; \ (z, \rho, \psi, b_1, \ldots, b_n) \Big\} \quad \text{iff}$$

$$(1) \quad g^z = \Big[\prod_{i=1}^{n} (pk_i)^{b_i} \Big]^c \cdot R,$$

$$(2) \quad c_0 = g^\rho \quad \text{and} \quad c_1 = g^z \cdot pk_t^\rho,$$

$$(3) \quad T_0 = g^\psi \quad \text{and} \quad T_1 = g^{\sum_{i=1}^n b_i} \cdot h^\psi,$$

$$(4) \quad b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n \qquad (\text{i.e. } b_i \in \{0, 1\}).$$

Fig. 6. The relation $\mathcal{R}_S$ used in the *Combine* algorithm of the Schnorr TAPS. Condition (1) verifies that (R, z) is a valid signature for m assuming $c = H(pk, R, m)$; (2) verifies that (c_0, c_1) is an ElGamal encryption of z using the tracing public key pk_t; (3) verifies that the quorum C contains t signers; and (4) verifies that each b_i is in $\{0, 1\}$. Here g and h are public random generators of $\mathbb{G}$.

The proof of Theorem 2 is presented in the following three lemmas, where we also provide concrete security bounds.

Lemma 4. *The Schnorr* TAPS *scheme is unforgeable and accountable, as in Definition 6, assuming the underlying Schnorr ATS is secure, as in Definition 8, and the non-interactive proof system (P, V) for $\mathcal{R}_S$ is an argument of knowledge. Concretely, for every adversary $\mathcal{A}$ that attacks* TAPS*, there exists an adversary $\mathcal{B}$ that runs in about the same time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\text{forg}}_{\mathcal{A},\mathcal{S}}(\lambda) \leq \big(\mathbf{Adv}^{\text{forg}}_{\mathcal{B},\mathcal{ATS}}(\lambda)\big) \cdot q(\lambda) + \kappa(\lambda) \tag{7}$$

where κ and q are the knowledge error and tightness of the proof system.

We provide the proof of Lemma 4 in Appendix C.

Lemma 5. *The Schnorr* TAPS *scheme is private against the public, as in Definition 7, assuming DDH holds in $\mathbb{G}$, the non-interactive proof system (P, V) for $\mathcal{R}_S$ is HVZK, and the signature scheme $\mathcal{SIG}$ is strongly unforgeable. Concretely, for every adversary $\mathcal{A}$ that attacks $\mathcal{S}$ there exist adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ that run in about the same time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\text{priv1}}_{\mathcal{A},\mathcal{S}}(\lambda) \leq 2\Big(\mathbf{Adv}^{\text{eufcma}}_{\mathcal{B}_1,\mathcal{SIG}}(\lambda) + Q \cdot \mathbf{Adv}^{\text{hvzk}}_{\mathcal{B}_2,(P,V)}(\lambda) \ + (Q+1) \cdot \mathbf{Adv}^{\text{ddh}}_{\mathcal{B}_3,\mathbb{G}}(\lambda)\Big) \tag{8}$$

where Q is the number of signature queries from $\mathcal{A}$.

We provide the proof of Lemma 5 in Appendix D.

Lemma 6. *The Schnorr scheme is private against signers, as in Definition 7, assuming DDH holds in $\mathbb{G}$, the non-interactive proof system (P, V) for $\mathcal{R}_S$ is HVZK, and the signature scheme $\mathcal{SIG}$ is strongly unforgeable.*

The proof of Lemma 6 is mostly the same as the proof of Lemma 5 and is omitted.

5.3 A sigma protocol proof for $\mathcal{R}_S$

It remains to construct an efficient non-interactive zero knowledge argument of knowledge for the relation $\mathcal{R}_S$ from Figure 6. In this section we construct a Sigma protocol, and in the next section we construct a protocol using Bulletproofs. We describe these as interactive protocols, but they can be made non-interactive using the Fiat-Shamir transform [35,4].

Let $g, h, h_1, \ldots, h_n \in \mathbb{G}$ be independent random generators of $\mathbb{G}$. To prove knowledge of a witness for the relation $\mathcal{R}_S$ from Figure 6 we use the following approach:

$$\mathcal{R}_{S1} := \Big\{ (g, h, h_1, \ldots, h_n, pk_1, \ldots, pk_n, pk_t, T_0, T_1, R, c, ct = (c_0, c_1), v_0, v_1, \ldots, v_n, \alpha) \; ;$$
$$(z, \rho, \psi, \gamma, b_1, \ldots, b_n, \phi_1, \ldots, \phi_n) \Big\} \text{ where}$$

(1) $g^z = R \cdot \prod_{i=1}^{n} (pk_i)^{c \cdot b_i}$

(2) $c_0 = g^\rho$ and $c_1 = pk_t^\rho \cdot g^z$

(3) $T_0 = g^\psi$ and $T_1 = g^{\sum_{i=1}^n b_i} \cdot h^\psi$

(4) $v_0 = g^\gamma$ and $v_i = g^{b_i} h_i^\gamma$ for $i \in [n]$ and $\prod_{i=1}^{n} v_i^{\alpha^i (1-b_i)} = \prod_{i=1}^{n} h_i^{\phi_i}$

Fig. 7. The relation $\mathcal{R}_{S1}$. Equations (1), (2), and (3) are the same as in the relation $\mathcal{R}_S$ in Figure 6. Equation (4) proves that $b_i(1-b_i) = 0$ for $i \in [n]$. As usual, both the prover and verifier have $c \leftarrow H(pk, R, m)$. The prover computes the witness element $\phi_1, \ldots, \phi_n \in \mathbb{Z}_q$ on its own as $\phi_i \leftarrow \alpha^i \gamma (1 - b_i)$.

Protocol $S1$:

1: The prover chooses $\gamma \xleftarrow{\$} \mathbb{Z}_q$ and commits to its bits $(b_1, \ldots, b_n) \in \{0,1\}^n$ as

$$\big(\; v_0 \leftarrow g^\gamma, \quad v_1 \leftarrow g^{b_1} h_1^\gamma, \quad \ldots, \quad v_n \leftarrow g^{b_n} h_n^\gamma \; \big) \in \mathbb{G}^{n+1}$$

It sends $(v_0, v_1, \ldots, v_n)$ to the verifier. Observe that for $i \in [n]$ the pair (v_0, v_i) is an ElGamal encryption of b_i with respect to the public key h_i. The term v_0 will be used for efficient tracing.

2: The verifier samples a challenge $\alpha \xleftarrow{\$} \mathbb{Z}_q$ and sends α to the prover.

3: The prover computes $\phi_i \leftarrow \alpha^i \gamma (1 - b_i) \in \mathbb{Z}_q$ for $i \in [n]$.

4: Finally, the prover uses a Sigma protocol to prove knowledge of a witness $(z, \rho, \psi, \gamma, b_1, \ldots, b_n, \phi_1, \ldots, \phi_n)$ for the relation $\mathcal{R}_{S1}$ in Figure 7.

We present the concrete steps for the 3-round Sigma protocol for the relation $\mathcal{R}_{S1}$ used in Step 4 in Appendix E, where we also show the TAPS signature obtained from this protocol. After applying the Fiat-Shamir transform to Protocol $S1$, the resulting proof π for the relation $\mathcal{R}_S$ from Figure 6 contains $n+1$ group elements and $2n+5$ elements in $\mathbb{Z}_q$.

Theorem 3. *Let $\mathbb{G}$ be a group of prime order q. If the Decision Diffie-Hellman (DDH) assumption holds in $\mathbb{G}$, and n/q is negligible, then Protocol $S1$ is an HVZK argument of knowledge for the relation $\mathcal{R}_S$ from Figure 6.*

We provide the proof for Theorem 3 in Appendix F.

*Remark 8 (**Efficient tracing**).* Recall that the tracing algorithm in Figure 5 requires the tracer to find a set $C \subseteq [n]$ of size t such that $g^{(z')} = (\prod_{i \in C} pk_i)^c \cdot R$. When using Protocol $S1$, the tracing algorithm can efficiently find this set $C \subseteq [n]$ by decrypting the Combiner's ElGamal commitment $(v_0, v_1, \ldots, v_n) \in \mathbb{G}^{n+1}$ to the bits $b_1, \ldots, b_n \in \{0,1\}$ that define C. To see how, let us extend algorithm *KeyGen* in Figure 5 by adding the following steps:

- choose $\tau_i \xleftarrow{\$} \mathbb{Z}_q$ and set $h_i \leftarrow g^{\tau_i}$ for $i \in [n]$
- $aug\text{-}sk_t \leftarrow (sk_t, \; \tau_1, \ldots, \tau_n)$ // augmented tracing key
- $aug\text{-}sk_c \leftarrow (sk_c, \; h_1, \ldots, h_n)$ // augmented Combiner's key
- $aug\text{-}pk \leftarrow (pk, \; h_1, \ldots, h_n)$ // augmented public key

The Combiner and verifier use $h_1, \ldots, h_n$ in their augmented keys to produce and verify the proof for the relation $\mathcal{R}_S$ using Protocol $S1$. The proof contains an ElGamal commitment $(v_0, v_1, \ldots, v_n)$ to the bits $b_1, \ldots, b_n$. The tracing algorithm can obtain $b_1, \ldots, b_n \in \{0,1\}$ by decrypting the ElGamal ciphertexts (v_0, v_i) for $i \in [n]$ using the secret keys $\tau_1, \ldots, \tau_n \in \mathbb{Z}_q$. Soundness of Protocol $S1$ ensures that the resulting bits define the correct quorum set C. Note that *aug-pk* contains a total of $2n+4$ group elements.

5.4 A bulletproofs protocol proof for $\mathcal{R}_S$

The Sigma protocol for the relation $\mathcal{R}_S$ from Figure 6 may be adequate for many real-world settings where the number of allowed signers is small. However, if a large number of parties n is used, then the resulting proof size may be too large. We can shrink the proof using an argument system that produces shorter proofs (e.g., using a zk-SNARK). This approach raises two difficulties. First, computing the proof will be slow because the exponentiations in Figure 6 would need to be implemented explicitly in the zk-SNARK relation. Second, we would lose the efficient tracing algorithm from Remark 8.

We can avoid both issues using the Bulletproofs proof system [22,24] or its treatment as a compressed Sigma protocol in [3]. First, the exponentiations in Figure 6 are handled efficiently. Second, we can retain efficient tracing with a much shorter TAPS signature compared to the Sigma protocol in Section 5.3.

Let $\mathbb{G}$ be a group of prime order q, let $a_1, \ldots, a_n$ be generators of $\mathbb{G}$, and $\mathbf{a} := (a_1, \ldots, a_n) \in \mathbb{G}^n$. For $\mathbf{w} \in \mathbb{Z}_q^n$ we write $\mathbf{a}^{\mathbf{w}} := \prod_{i=1}^n a_i^{w_i} \in \mathbb{G}$. Recall that bulletproofs is an HVZK proof system that can prove knowledge of a satisfying witness $\mathbf{w} \in \mathbb{Z}_q^n$ for the relation

$$\mathcal{R}_{\text{BP}} := \left\{ (P,\ \mathbf{a} \in \mathbb{G}^n,\ u \in \mathbb{G})\ ;\ \mathbf{w} \in \mathbb{Z}_q^n \right\} \qquad \text{iff } P(\mathbf{w}) = 1 \text{ and } \mathbf{a}^{\mathbf{w}} = u,$$

where P is a *rank one constraint system* (R1CS), meaning that P is a triple of matrices $A, B, C \in \mathbb{Z}_q^{\ell \times n}$ and $P(\mathbf{w}) = 1$ iff $(A\mathbf{w}) \circ (B\mathbf{w}) = C\mathbf{w}$. The $\circ$ operator denotes the Hadamard product (component-wise product) of two vectors in $\mathbb{Z}_q^n$. The program P is said to have ℓ constraints over n variables. We represent the program P in $\mathcal{R}_{\text{BP}}$ using R1CS instead of an arithmetic circuit because R1CS is more convenient in our settings: it more directly captures the relations we need to prove.

The Bulletproofs proof is succinct, containing only $2\lceil \log_2(n + \ell) \rceil$ group elements and two elements in $\mathbb{Z}_q$. For a convincing prover P^*, the Bulletproofs extractor outputs some $\mathbf{w} \in \mathbb{Z}_q^n$ such that either (i) $\mathbf{w}$ is a valid witness for $\mathcal{R}_{\text{BP}}$, or (ii) $\mathbf{w}$ is a non-trivial relation among the generators[5] $\mathbf{a} \in \mathbb{G}^n$, namely $\mathbf{a}^{\mathbf{w}} = 1$. If the discrete log problem in $\mathbb{G}$ is difficult, and $\mathbf{a}$ are random generators of $\mathbb{G}$, then an efficient prover cannot cause (ii) to happen. Then bulletproofs is an argument of knowledge for $\mathcal{R}_{\text{BP}}$.

To capture this notion formally, it is convenient to define a *restricted argument of knowledge* where the extractor is only guaranteed to work against a certain class of instance generators.

Definition 10. *A proof system $(\mathcal{P}, \mathcal{V})$ is a* **restricted argument of knowledge** *for a relation $\mathcal{R}$ with respect to a class of instance generators IG if it is perfectly complete and there is an extractor Ext that satisfies the requirements of Definition 4 against all PPT provers $(\mathcal{P}_1, \mathcal{P}_2)$ where the instance generator $\mathcal{P}_1$ is in IG.*

Theorem 4 ([22,24]). *Suppose that the discrete log assumption holds in the group $\mathbb{G}$. Then the bulletproofs proof system is a* **restricted argument of knowledge** *for $\mathcal{R}_{BP}$ with respect to the class of instance generators $\mathcal{P}_1 \in \mathsf{IG}_{BP}$ that operate in two steps: first a vector $\mathbf{a} \xleftarrow{\$} \mathbb{G}^n$ is sampled, then $\mathcal{P}_1(\mathbf{a})$ outputs a program P and $u \in \mathbb{G}$ to form the $\mathcal{R}_{BP}$ statement $(P, \mathbf{a}, u)$.*

An expanded bulletproofs relation. For our application we need to generalize the bulletproofs relation. Let $\mathbf{V} = (v_{i,j}) \in \mathbb{G}^{\ell' \times n}$ be a matrix of arbitrary group elements in $\mathbb{G}$ (not necessarily random generators of $\mathbb{G}$). Let $\mathbf{w} \in \mathbb{Z}_q^n$ and $\mathbf{u}' := \mathbf{V}^{\mathbf{w}} \in \mathbb{G}^{\ell'}$. We claim that bulletproofs can be used to efficiently prove knowledge of a valid witness $\mathbf{w} \in \mathbb{Z}_q^n$ for the following expanded relation:

$$\mathcal{R}_{\text{eBP}} := \left\{ (P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')\ ;\ \mathbf{w} \in \mathbb{Z}_q^n \right\} \quad \text{iff } P(\mathbf{w}) = 1,\ \ \mathbf{a}^{\mathbf{w}} = u,\ \text{ and } \mathbf{V}^{\mathbf{w}} = \mathbf{u}'.$$

In other words, a valid witness $\mathbf{w}$ for an $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$ must be a valid witness for the $\mathcal{R}_{\text{BP}}$ instance $(P, \mathbf{a}, u)$, and furthermore it must satisfy $\mathbf{V}^{\mathbf{w}} = \mathbf{u}'$ where $\mathbf{V}$ is a matrix of *arbitrary* group elements. In particular, the prover may know a non-trivial relation among the group elements in $\mathbf{V}$.

We need this expanded BP relation because the relation $\mathcal{R}_S$ from Figure 6 involves group elements for which the adversary knows their discrete log base g. In particular, the adversary knows the discrete log of public keys that belong to corrupt parties. The basic bulletproofs system for $\mathcal{R}_{\text{BP}}$ is not an argument of knowledge in this setting, because the bulletproofs extractor may fail to extract a valid witness.

The following theorem shows that a proof system for $\mathcal{R}_{\text{eBP}}$ is no harder than bulletproofs for $\mathcal{R}_{\text{BP}}$, as long as $\mathbf{V} \in \mathbb{G}^{\ell' \times n}$ is fixed before the random generators $\mathbf{a} \in \mathbb{G}^n$ are sampled. The proof can be found in Appendix G.

[5] This relation might include additional random generators of $\mathbb{G}$, as explained in Appendix G.

$$\mathcal{R}_{S2} := \left\{ (P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}') \;;\; \mathbf{w} := (z, \rho, \psi, b_1, \ldots, b_n, \alpha) \in \mathbb{Z}_q^{n+4} \right\} \text{ iff}$$
$$P(\mathbf{w}) = 1, \quad \mathbf{a}^{\mathbf{w}} = u, \quad \mathbf{V}^{\mathbf{w}} = \mathbf{u}', \quad \text{where}$$
$$\mathbf{V} := \begin{pmatrix} g, & 1, & 1, & pk_1^{-c}, \ldots, & pk_n^{-c}, & 1 \\ 1, & g, & 1, & & & 1 \\ g, & pk_{\mathrm{t}}, & 1, & 1 & & 1 \\ 1, & 1, & g, & & & 1 \\ 1, & 1, & h, & g, \ldots, & g, & 1 \end{pmatrix} \in \mathbb{G}^{5\times(n+4)} \quad \text{and} \quad \mathbf{u}' := \begin{pmatrix} R \\ c_0 \\ c_1 \\ T_0 \\ T_1 \end{pmatrix} \in \mathbb{G}^5$$
$$\text{and} \quad P(\mathbf{w}) = 1 \iff b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n.$$

Fig. 8. Recasting the relation $\mathcal{R}_S$ from Figure 6 as an instance of $\mathcal{R}_{\text{eBP}}$. Here $\mathbf{a} \in \mathbb{G}^{n+4}$ is a vector of random generators and $u \in \mathbb{G}$. The vector equation $\mathbf{V}^{\mathbf{w}} = \mathbf{u}'$ captures lines (1), (2), (3) in Figure 6, one row for each of the five equalities in those lines. The program P captures line (4). The new random α appended to the witness $\mathbf{w} \in \mathbb{Z}_q^{n+4}$ is there to ensure that $u := \mathbf{a}^{\mathbf{w}}$ leaks nothing about the first $n+3$ elements in the witness.

Theorem 5. *Suppose that the discrete log assumption holds in the group $\mathbb{G}$. Then there is an HVZK restricted argument of knowledge for $\mathcal{R}_{eBP}$ where the proof size is the same as in bulletproofs for $\mathcal{R}_{BP}$, and proof generation and verification times differ by at most the time to compute $(n+1)\ell'$ exponentiations in $\mathbb{G}$. The argument of knowledge property holds with respect to instance generators $\mathcal{P}_1 \in \mathsf{IG}_{eBP}$ that operate as follows: $\mathcal{P}_1$ outputs $\mathbf{V} = (v_{i,j}) \in \mathbb{G}^{\ell' \times n}$ and $\mathbf{u}' \in \mathbb{G}^{\ell'}$, then $\mathbf{a} \xleftarrow{\$} \mathbb{G}^n$ is sampled, and finally $\mathcal{P}_1(\mathbf{a})$ outputs P, u to form the $\mathcal{R}_{eBP}$ statement $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$.*

Figure 8 shows how to recast the relation $\mathcal{R}_S$ from Figure 6 as an instance of $\mathcal{R}_{\text{eBP}}$. We call this relation $\mathcal{R}_{S2}$. It should be clear that if $\mathbf{w} := (\mathbf{w}', \alpha) \in \mathbb{Z}_q^{n+4}$ is a valid witness for $\mathcal{R}_{S2}$ then $\mathbf{w}' \in \mathbb{Z}_q^{n+3}$ is a valid witness for $\mathcal{R}_S$.

Now, the HVZK argument of knowledge for the relation $\mathcal{R}_S$ from Figure 6 follows directly from Theorem 5. To see how, consider an $\mathcal{R}_S$ statement, and let $\mathbf{w}' := (z, \rho, \psi, b_1, \ldots, b_n) \in \mathbb{Z}_q^{n+3}$ be a valid witness.

Protocol $S2$:

1: The verifier sends a vector of random generators $\mathbf{a} \in \mathbb{G}^{n+4}$ to the prover.

2: The prover chooses a random $\alpha \xleftarrow{\$} \mathbb{Z}_q$, sets $\mathbf{w} := (\mathbf{w}', \alpha) \in \mathbb{Z}_q^{n+4}$, and computes $u \leftarrow \mathbf{a}^{\mathbf{w}} \in \mathbb{G}$. It sends u to the verifier.

3: The prover and verifier run the proof system from Theorem 5 to prove that $\mathbf{w}$ is valid witness for $\mathcal{R}_{S2}$ in Figure 8. As required, $\mathbf{V}$ is chosen before $\mathbf{a}$ is sampled.

Note that the vector $\mathbf{a} \in \mathbb{G}^{n+4}$ is chosen after the matrix $\mathbf{V}$ is fixed. Hence, if the discrete log problem in $\mathbb{G}$ is difficult, then by Theorem 5, this protocol is an HVZK argument of knowledge for $\mathcal{R}_{S2}$ and hence also for $\mathcal{R}_S$.

The program P in $\mathcal{R}_{S2}$ is an R1CS program over $n+4$ variables (the length of the witness $\mathbf{w}$) and has n constraints. Therefore, the Fiat-Shamir transform applied to Protocol $S2$ results in a proof for $\mathcal{R}_S$ from Figure 6 of size

$$2\lceil \log_2(2n+4) \rceil + 1 \text{ group elements and two elements in } \mathbb{Z}_q.$$

The analysis of the Fiat-Shamir transform in such settings can be found in [2,4,40,62].

Efficient tracing. The proof for $\mathcal{R}_S$ using Protocol $S2$ and the resulting TAPS signature now have size logarithmic in n. However, we lost the ability to efficiently trace a signature using the tracing key. The tracing algorithm in Figure 5 needs to find a set $C \subseteq [n]$ of size t such that $g^{(z')} = (\prod_{i \in C} pk_i)^c \cdot R$. This

can be done, in principal, by trying all sets $C \subseteq [n]$ of size t, assuming $\binom{n}{t}$ is polynomial in the security parameter λ. However, we want a more efficient tracing algorithm.

We can restore efficient tracing for larger n and t in a way similar to Remark 8. Let $(b_1, \ldots, b_n) \in \{0,1\}^n$ be the characteristic vector of the quorum of signers $C \subseteq [n]$. In Section 5.3 we encrypted every bit b_i on its own, and added the $n+1$ group elements $(v_0, \ldots, v_n)$ to the signature. The tracing algorithm could then decrypt each of the n ElGamal ciphertexts (v_0, v_i), for $i \in [n]$, and efficiently recover the quorum set C.

Using Bulletproofs we can compress the commitment to the bits $(b_1, \ldots, b_n)$ by committing to a *batch* of bits at a time using a single ElGamal ciphertext. We will then need to extend the relation $\mathcal{R}_{S2}$ to verify that every batch commitment is well formed.

To see how, let us fix a batch size e, say $e := 40$. For simplicity suppose that e divides n. We extend algorithm *KeyGen* in Figure 5 by adding the following steps:

- for $i \in [n/e]$: choose $\tau_i \xleftarrow{\$} \mathbb{Z}_q$ and set $h_i \leftarrow g^{\tau_i} \in \mathbb{G}$
- $\textit{aug-pk} \leftarrow (pk,\ h_1, \ldots, h_{n/e})$ // augmented public key
- $\textit{aug-sk}_{\mathrm{c}} \leftarrow (sk_{\mathrm{c}},\ h_1, \ldots, h_{n/e})$ // augmented Combiner's key
- $\textit{aug-sk}_{\mathrm{t}} \leftarrow (sk_{\mathrm{t}},\ \tau_1, \ldots, \tau_{n/e})$ // augmented tracing key

Next, we augment the prover in Protocol $S2$ by adding a step 0 where the prover does:

- step (i): Divide the n bits into (n/e) buckets $0 \le B_1, \ldots, B_{n/e} < 2^e$ as:

$$\begin{cases} B_1 \leftarrow b_1 + 2b_2 + 4b_3 + \ldots + 2^e b_e \in \mathbb{Z}_q, \\ B_2 \leftarrow b_{e+1} + 2b_{e+2} + \ldots + 2^e b_{2e} \in \mathbb{Z}_q, \\ \quad \vdots \\ B_{n/e} \leftarrow b_{n-e+1} + 2b_{n-e+2} + \ldots + 2^e b_n \in \mathbb{Z}_q. \end{cases}$$

- step (ii): Choose a random $\gamma \xleftarrow{\$} \mathbb{Z}_q$ and compute

$$\left(v_0 \leftarrow g^{\gamma}, \quad v_1 \leftarrow g^{B_1} h_1^{\gamma}, \quad \ldots, \quad v_{n/e} \leftarrow g^{B_{n/e}} h_{n/e}^{\gamma} \right) \in \mathbb{G}^{(n/e)+1}.$$

  Send $(v_0, v_1, \ldots, v_{n/e})$ to the verifier. Observe that for $i \in [n/e]$ the pair (v_0, v_i) is an ElGamal encryption of g^{B_i} with respect to the public key h_i.

Finally, we augment the relation $\mathcal{R}_{S2}$ to verify that $(v_0, v_1, \ldots, v_{n/e})$ were constructed correctly. This adds one column to the matrix $\mathbf{V}$ in Figure 8 to account for the new witness element γ, and adds $(n/e)+1$ rows to the matrix $\mathbf{V}$ and to the vector $\mathbf{u}'$ to ensure that $v_i = g^{B_i} h_i^{\gamma}$ for $i \in [n/e]$ and that $v_0 = g^{\gamma}$. Now, after applying Fiat-Shamir, the resulting proof for $\mathcal{R}_S$ (generated by the Combiner) has size

$$(n/e) + 2\lceil \log_2(2n + (n/e) + 6) \rceil + 2 \text{ group elements and two elements in } \mathbb{Z}_q.$$

The dominant term is the (n/e) group elements. The final TAPS signature is expanded by $(n/e) + 1$ group elements $(v_0, v_1, \ldots, v_{n/e})$.

When the tracing algorithm is given a signature to trace, it can obtain $g^{B_1}, \ldots, g^{B_{n/e}} \in \mathbb{G}$ by decrypting the ElGamal ciphertexts (v_0, v_i) for $i \in [n/e]$ using the secret keys $\tau_1, \ldots, \tau_{n/e} \in \mathbb{Z}_q$ in the tracing key $\textit{aug-sk}_{\mathrm{t}}$. Next, the tracing algorithm computes the discrete log base g of these group elements to obtain $B_1, \ldots, B_{n/e} \in \mathbb{Z}_q$. Since each B_i is in $\{0, 1, \ldots, 2^e - 1\}$, each discrete log computation can be done with about $2^{e/2}$ group operations.

Taking $e := 40$ gives a reasonable amount of time for computing all of $B_1, \ldots, B_{n/e} \in \mathbb{Z}_q$ from $g^{B_1}, \ldots, g^{B_{n/e}}$. The tracing algorithm then computes $b_1, \ldots, b_n \in \{0,1\}$ from $B_1, \ldots, B_{n/e}$, and this reveals the required quorum set C. Soundness of the augmented Protocol $S2$ ensures that the resulting bits $b_1, \ldots, b_n$ define the correct quorum set $C \subseteq [n]$.

6 Extensions

Shorter public keys. While the size of the public key in our Schnorr construction grows linearly in n, there are several ways to shrink the public key. First, the public key can be replaced by a short binding commitment to the linear-size public key, and the full public key could be included in every signature. This shrinks the public key at the cost of expanding the signature. Alternatively, both the public key and signature can be kept short by making the public key a witness in the zero-knowledge proof statement, as is done in the generic construction (Figure 4). However, doing so comes at the cost of increased complexity of the statement that the Combiner needs to prove.

Shorter signatures using tracing confirmation. The need to trace a TAPS signature to the signing quorum implies that a TAPS signature must encode the signing set, and therefore must be at least $\log_2 \binom{n}{t}$ bits long. We can design shorter TAPS signatures by relaxing this requirement: replace the tracing algorithm by a *quorum confirmation* algorithm. The confirmation algorithm takes the signing quorum set C as input, along with the secret tracing key sk_t, and a pair (m, σ). It outputs 1 if the set C is the set that generated σ. The security definitions in Section 3 can be adapted to support quorum confirmation instead of tracing. Since a signature no longer needs to encode the quorum set, this lets us construct TAPS where signature size in independent of the number of parties, for example by using a constant-size zk-SNARK for the relation $\mathcal{R}_S$ in Figure 6. Our bulletproofs construction can be made to directly achieve a TAPS with quorum confirmation and logarithmic size signatures.

Stronger privacy against signers. Our privacy against signers game in Figure 3 ensures that the signer's private keys cannot be used to link a TAPS signature to the quorum that created it. However, it is possible that the quorum of signers that helped create a TAPS signature σ, can later recognize σ, using its knowledge of the random bits used during the signing process. The same is true for many Schnorr private threshold signature (PTS) schemes: the quorum that creates a signature can recognize that signature. If needed, our Schnorr TAPS construction can be strengthened so that the Combiner can ensure that a TAPS signature cannot be recognized by the quorum of signers that helped create it. The Combiner need only blind the quantity $R \in \mathbb{G}$ in the signature by a random group element, and adjust the relation in Figure 6 accordingly. We leave this variation for future work.

A construction from the BLS signature scheme. In this paper we focused on a TAPS from the Schnorr signature scheme. A TAPS can also be constructed from the BLS signature scheme [18] as the underlying ATS. We leave this for future work.

Beyond threshold: supporting monotone access structures. While threshold access structures are widely used in practice, our constructions generalize to support more general monotone access structures. For example, one can require that a quorum of signers contain t_1 parties from one set of signers and t_2 from another set of signers. More generally, standard techniques [7] can be used to generalize our construction to support any access structure derived from a polynomial size monotone formula.

7 Conclusion and Future Work

In this work, we present TAPS, a new threshold signature primitive that ensures both accountability and privacy. While notions of accountable threshold schemes and private threshold schemes exist in the literature, our work takes a step towards defining a primitive with both properties simultaneously.

We hope that future work can lead to TAPS schemes with shorter signatures and public keys. Our generic construction has a short public key: the public key is simply a commitment to an ATS public key, and so its size is independent of the number of parties n. However, our Schnorr-based systems with efficient tracing require a linear size public key. An important research direction is to design an efficient TAPS that relies on standard assumptions where the size of the public key is independent of n. One possible avenue for a more efficient TAPS is for pk to be the root of a Merkle tree whose leaves are the n signers' public keys. The zero-knowledge proof output by the Combiner will then be a succinct non-interactive zero-knowledge argument of knowledge (a zk-SNARK) demonstrating that t of the n signers participated in signing. A related direction is to employ the approach of Dodis et al. [33], by

defining the public key via an accumulator scheme. The signature is then a proof that the t signers know the corresponding secret keys to t public keys in the accumulator. However, it remains an open problem to design such a scheme that fulfills our notion of accountability.

Another direction for future work is to improve the efficiency of verification in our Schnorr TAPS. In settings where n is small, such as financial transactions, the linear-time cost of verification of the Schnorr construction is acceptable. For large n the cost may be prohibitive. Future work could consider other constructions that support full tracing, but with a faster verifier.

Acknowledgments. This work was funded by NSF, DARPA, a grant from ONR, the Simons Foundation, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

1. G. Andresen. Bitcoin m-of-n standard transactions, 2011. BIP-0011.
2. T. Attema and R. Cramer. Compressed σ-protocol theory and practical application to plug & play secure algorithmics. In *CRYPTO '20*, volume 12172 of *LNCS*, pages 513–543. Springer, 2020.
3. T. Attema and R. Cramer. Compressed ς-protocol theory and practical application to plug & play secure algorithmics. In *CRYPTO 2020*, volume 12172 of *LNCS*, pages 513–543. Springer, 2020.
4. T. Attema, S. Fehr, and M. Klooß. Fiat-shamir transformation of multi-round interactive proofs. Cryptology ePrint Archive, Report 2021/1377, 2021. https://ia.cr/2021/1377.
5. M. H. Au, S. S. M. Chow, W. Susilo, and P. P. Tsang. Short linkable ring signatures revisited. In *Public Key Infrastructure PKI Workshop*, 2006.
6. A. Bagherzandi, J. H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *Conference on Computer and Communications Security*, 2008.
7. A. Beimel. Secret-sharing schemes: A survey. In *IWCC*, 2011.
8. M. Bellare and O. Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, volume 740 of *LNCS*, pages 390–420. Springer, 1992.
9. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT '03*, volume 2656 of *LNCS*, pages 614–629. Springer, 2003.
10. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *CCS'06*, pages 390–399. ACM, 2006.
11. M. Bellare, S. Tessaro, and C. Zhu. Stronger security for non-interactive threshold signatures: Bls and frost. Cryptology ePrint Archive, Paper 2022/833, 2022. https://eprint.iacr.org/2022/833.
12. A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *J. of Cryptology*, 22(1):114–138, 2009.
13. P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi. Get shorty via group signatures without encryption. In *SCN'10*, volume 6280 of *LNCS*, pages 381–398. Springer, 2010.
14. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS'12*, pages 326–349. ACM, 2012.
15. J. Blömer, J. Juhnke, and N. Löken. Short group signatures with distributed traceability. In *Mathematical Aspects of Computer and Information Sciences MACIS*, 2015.
16. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Y. Desmedt, editor, *PKC*, volume 2567, pages 31–46, 2003.
17. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.
18. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, 2001.
19. D. Boneh and V. Shoup. *A graduate course in applied cryptography (version 0.5)*. 2020. cryptobook.us.
20. J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth. Foundations of fully dynamic group signatures. *J. of Cryptology*, 33(4):1822–1870, 2020.
21. J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit. Short accountable ring signatures based on DDH. In *European Symposium on Research in Computer Security*, volume 9326 of *LNCS*, pages 243–265. Springer, 2015.
22. J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 327–357. Springer, 2016.
23. E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures and applications to ad-hoc groups. In *Crypto'02*, volume 2442 of *LNCS*, pages 465–480. Springer, 2002.

24. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *IEEE S&P*, pages 315–334, 2018.
25. J. Camenisch. Efficient and generalized group signatures. In *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 465–479. Springer, 1997.
26. J. Camenisch, M. Drijvers, A. Lehmann, G. Neven, and P. Towa. Short threshold dynamic group signatures. In *SCN'20*, volume 12238 of *LNCS*, pages 401–423. Springer, 2020.
27. D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT '91*, volume 547 of *LNCS*, pages 257–265. Springer, 1991.
28. I. Damgård and M. Koprowski. Practical threshold RSA signatures without a trusted dealer. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 152–165. Springer, 2001.
29. I. Damgård. On Σ Protocols, 2010.
30. H. de Valence. Merlin transcripts. https://merlin.cool.
31. D. Derler and D. Slamanig. Highly-efficient fully-anonymous dynamic group signatures. In *AsiaCCS'18*, 2018.
32. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO '89*, 1989.
33. Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In *EUROCRYPT '04*, volume 3027 of *LNCS*, pages 609–626. Springer, 2004.
34. H. Feng, J. Liu, D. Li, Y. Li, and Q. Wu. Traceable ring signatures: general framework and post-quantum security. *Designs, Codes and Cryptography*, 89(6):1111–1145, 2021.
35. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.
36. P. Fouque and J. Stern. Fully distributed threshold RSA under standard assumptions. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 310–330. Springer, 2001.
37. E. Fujisaki. Sub-linear size traceable ring signatures without random oracles. In *CT-RSA*, 2011.
38. E. Fujisaki and K. Suzuki. Traceable ring signature. In *Public Key Cryptography*, 2007.
39. J. Furukawa and H. Imai. An efficient group signature scheme from bilinear maps. *IEICE Tr. on Fundamentals of Electronics Comm. and Comp. Sci.*, 89-A(5):1328–1338, 2006.
40. C. Ganesh, C. Orlandi, M. Pancholi, A. Takahashi, and D. Tschudi. Fiat–shamir bulletproofs are non-malleable (in the algebraic group model). Cryptology ePrint Archive, Report 2021/1393, 2021. https://ia.cr/2021/1393.
41. R. Gennaro and S. Goldfeder. One round threshold ecdsa with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020.
42. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inf. Comput.*, 164(1):54–84, 2001.
43. C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC'11*, pages 99–108. ACM, 2011.
44. S. Goldwasser, S. Micali, and A. C. Yao. Strong signature schemes. In *FOCS'83*, pages 431–439. ACM, 1983.
45. V. Goyal, Y. Song, and A. Srinivasan. Traceable secret sharing and applications. In *CRYPTO '21*, volume 12827 of *LNCS*, pages 718–747. Springer, 2021.
46. A. Haque and A. Scafuro. Threshold ring signatures: New definitions and post-quantum security. In *Public-Key Cryptography*, volume 12111 of *LNCS*, pages 423–452. Springer, 2020.
47. K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC research and development*, 1983.
48. C. Komlo and I. Goldberg. FROST: flexible round-optimized schnorr threshold signatures. In *SAC'20*, 2020.
49. B. Libert and M. Yung. Dynamic fully forward-secure group signatures. In *AsiaCCS'10*, 2010.
50. J. K. Liu, V. K. Wei, and D. S. Wong. A separable threshold ring signature scheme. In *Information Security and Cryptology - ICISC*, 2003.
51. M. Manulis, N. Fleischhacker, F. Günther, F. Kiefer, and B. Poettrering. Group signatures: Authentication with privacy. *Bundesamt fur Sicherheit in der Informationstechnik*, 2012.
52. G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.
53. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *CCS'01*, pages 245–254. ACM, 2001.
54. A. Munch-Hansen, C. Orlandi, and S. Yakoubov. Stronger notions and a more efficient construction of threshold ring signatures. In *Latincrypt*, 2021.
55. M. Naor. Deniable ring authentication. In *CRYPTO 2002*, 2002.
56. J. Nick, T. Ruffing, and Y. Seurin. MuSig2: Simple Two-Round Schnorr Multi-Signatures. In *CRYPTO 2021*, volume 12825 of *LNCS*, pages 189–221. Springer, 2021.
57. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, 2001.
58. C. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO '89*, volume 435 of *LNCS*, pages 239–252. Springer, 1989.

59. V. Shoup. Practical threshold signatures. In *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, 2000.
60. D. R. Stinson and R. Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In *Information Security and Privacy*, 2001.
61. P. P. Tsang, V. K. Wei, T. K. Chan, M. H. Au, J. K. Liu, and D. S. Wong. Separable linkable threshold ring signatures. In *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 384–398. Springer, 2004.
62. D. Wikström. Special soundness in the random oracle model. Cryptology ePrint Archive, Report 2021/1265, 2021. https://ia.cr/2021/1265.
63. S. Xu and M. Yung. Accountable ring signatures: A smart card approach. In *IFIP'04*, 2004.

A Proof of Lemma 1

Proof. We prove Lemma 1 using a sequence of three games.

Game 0. This is the unforgeability and accountability game in Figure 1 applied to the generic scheme $\mathcal{S}$. If W_0 is the event that $\mathcal{A}$ wins in Game 0, then

$$\mathbf{Adv}^{\text{forg}}_{\mathcal{A},\mathcal{S}}(\lambda) = \Pr[W_0]. \tag{9}$$

Game 1. Let $pk = (\mathsf{com}_{pk}, pk_{\text{t}}, pk_{\text{cs}})$ be the public key given to the adversary, and let $sk_{\text{t}} = (pk', sk'_{\text{t}}, sk_{\text{cs}})$ be the tracing key given to the adversary.

Game 1 is identical to Game 0 except that we strengthen the winning condition over Game 0: to win the game, the adversary must also output a valid forgery (m', σ') where $\sigma' = (ct', \pi', tg')$, along with a witness $(\sigma''_{\text{m}}, r'', r''_{pk}, pk'')$ such that

$$\mathcal{R}\Big((\mathsf{com}_{pk}, pk_{\text{t}}, m', ct')\ ;\ (\sigma''_{\text{m}}, r'', r''_{pk}, pk'')\Big) = \text{true}, \tag{10}$$

where $\mathcal{R}$ is defined in (3).

We build a Game 1 adversary $\mathcal{A}'$ from a Game 0 adversary $\mathcal{A}$. This $\mathcal{A}'$ is given $(\mathsf{com}_{pk}, pk_{\text{t}})$ along with other quantities listed in Figure 1. It then runs $\mathcal{A}$, answering all its queries, until it obtains from $\mathcal{A}$ the quantities m' and ct' to form an $\mathcal{R}$ statement $(\mathsf{com}_{pk}, pk_{\text{t}}, m', ct')$. Next, $\mathcal{A}'$ runs the extractor Ext for the proof system (P, V) on the remaining execution of $\mathcal{A}$. The extractor outputs a witness $w := (\sigma''_{\text{m}}, r'', r''_{pk}, pk'')$ for $\mathcal{R}$. Finally, $\mathcal{A}'$ uses w and sk_{cs} to generate π' and tg' so that $\sigma' := (ct', \pi', tg')$ is a valid signature on m'. It outputs (m', σ') and w.

By definition of the extractor, if W_1 is the event that $\mathcal{A}'$ wins in Game 1, then

$$\Pr[W_1] \geq \big(\Pr[W_0] - \kappa(\lambda)\big)/q(\lambda). \tag{11}$$

Game 2. Let $sk_{\text{t}} = (pk', sk'_{\text{t}}, sk_{\text{cs}})$ and $pk = (\mathsf{com}_{pk}, pk_{\text{t}}, pk_{\text{cs}})$ be the tracing key and public key given to the adversary, where $\mathsf{com}_{pk} = \mathcal{COM}.Commit(pk', r_{pk})$. Hence, the adversary has pk', com_{pk}, and it also has r_{pk} from the combiner's secret key sk_{c}. The Game 1 adversary $\mathcal{A}'$ outputs a forgery (m', σ') and a witness $(\sigma''_{\text{m}}, r'', r''_{pk}, pk'')$. In Game 2 we further strengthen the winning condition by requiring that $pk' = pk''$.

Let us show that because the commitment scheme is binding, the difference in advantage between Games 1 and 2 is at most negligible. We know that

$$\mathcal{COM}.Verify(pk', r_{pk}, \mathsf{com}_{pk}) = \mathcal{COM}.Verify(pk'', r''_{pk}, \mathsf{com}_{pk}) = 1$$

where the second equality follows from (10). Therefore, if $pk' \neq pk''$ we obtain an attack on the binding property of $\mathcal{COM}$. In particular, let W_2 be the event that $\mathcal{A}'$ wins in Game 2, and let $\mathcal{E}$ be the event that $pk' \neq pk''$. Then we know that $\Pr[W_2] = \Pr[W_1 \wedge \neg\mathcal{E}] \geq \Pr[W_1] - \Pr[\mathcal{E}]$. Moreover, we just argued that there is an adversary $\mathcal{B}_2$ such that $\Pr[\mathcal{E}] = \mathbf{Adv}^{\text{bind}}_{\mathcal{B}_2,\mathcal{COM}}(\lambda)$. It follows that

$$\Pr[W_2] \geq \Pr[W_1] - \mathbf{Adv}^{\text{bind}}_{\mathcal{B}_2,\mathcal{COM}}(\lambda). \tag{12}$$

Finishing the proof. We show that for every Game 2 adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there is an ATS adversary $\mathcal{B}_1$ that wins the security game in Figure 1 against the underlying ATS scheme, with about the same advantage as $\mathcal{A}$'s advantage in Game 2. In particular, we show that

$$\mathbf{Adv}^{\text{forg}}_{\mathcal{B}_1,\mathcal{ATS}}(\lambda) \geq \Pr[W_2]. \tag{13}$$

Adversary $\mathcal{B}_1$ works as follows:

- run $\mathcal{A}_0$ and receive from $\mathcal{A}_0$ the tuple $(n, t, C, \mathtt{state}_0)$ where $C \subseteq [n]$.
- send (n, t, C) to its own ATS challenger and receives back $\big(pk',\ \{sk_i\}_{i\in C}\big)$.
- commit to pk' as: $r_{pk} \xleftarrow{\$} \mathcal{R}_\lambda$ and $\mathsf{com}_{pk} \leftarrow \mathcal{COM}.\mathit{Commit}(pk', r_{pk})$.
- generate the tracing key pair $(pk_{\text{t}}, sk'_{\text{t}}) \xleftarrow{\$} \mathcal{PKE}.\mathit{KeyGen}(1^\lambda)$.
- generate the combiner's signing key pair $(pk_{\text{cs}}, sk_{\text{cs}}) \xleftarrow{\$} \mathcal{SIG}.\mathit{KeyGen}(1^\lambda)$.
- set $sk_{\text{t}} \leftarrow (pk', sk'_{\text{t}}, pk_{\text{cs}})$, $pk \leftarrow (\mathsf{com}_{pk}, pk_{\text{t}}, pk_{\text{cs}})$, and $sk_{\text{c}} \leftarrow (pk', pk_{\text{t}}, sk_{\text{cs}}, t, \mathsf{com}_{pk}, r_{pk})$.
- run $\mathcal{A}_1(pk, \{sk_i\}_{i\in C}, sk_{\text{c}}, sk_{\text{t}}, \mathtt{state}_0)$.
- $\mathcal{A}_1$ issues signing queries: to respond to a signing query for m_j, our $\mathcal{B}$ issues a signing query for m_j to its own ATS challenger and gets back σ_j. It then runs steps (2)-(4) of the $\mathcal{S}.\mathit{Combine}$ algorithm to obtain a TAPS signature (ct_j, π_j, tg_j), and forwards it to $\mathcal{A}_1$.
- Eventually $\mathcal{A}_1$ outputs a forgery $(m', \sigma') = \big(m', (ct', \pi', tg')\big)$ and a witness $(\sigma''_{\text{m}}, r'', r''_{pk}, pk'')$ that satisfy the winning condition of Games 1 and 2.
- $\mathcal{B}$ outputs $(m', \sigma''_{\text{m}})$.

Suppose $\mathcal{A}$ wins in Game 2, so that in particular, $\mathcal{S}.\mathit{Verify}(pk, m', \sigma') = 1$. Then

$$\mathcal{SIG}.\mathit{Verify}\big(pk_{\text{cs}}, (m', ct', \pi'), tg'\big) = 1 \quad \text{and} \quad \mathcal{PKE}.\mathit{Decrypt}(sk'_{\text{t}}, ct') = \sigma''_{\text{m}}$$

where the equality on the right follows from the fact that the relation (10) holds. Therefore, if $C_t \leftarrow \mathcal{S}.\mathit{Trace}(sk_{\text{t}}, m', \sigma')$, then $C_t = \mathcal{ATS}.\mathit{Trace}(m', \sigma''_{\text{m}})$. Since $\mathcal{A}$ won in Game 2 we know that either $C_t = \mathsf{fail}$ or $C_t \not\subseteq (C \cup C')$, as defined in Figure 1.

It remains to argue that σ''_{m} is a valid ATS signature on m' with respect to pk'. Indeed, by (10) we know that $\mathcal{ATS}.\mathit{Verify}(pk'', m', \sigma''_{\text{m}}) = 1$. Moreover, by the condition in Game 2 we know that $pk' = pk''$. Therefore, σ''_{m} is a valid ATS signature on m' with respect to pk'. We conclude that if $\mathcal{A}$ wins in Game 2, then $(m', \sigma''_{\text{m}})$ output by $\mathcal{B}_1$ is a valid forgery for the underlying ATS.

In conclusion, since the ATS is secure, the adversary's advantage in winning Game 2 is at most negligible. It follows that the adversary has at most negligible advantage in Game 0, which completes the proof of the lemma. Concretely, combining (9), (11), (12), (13) proves (4), as required. □

B Proof of Lemma 2

Proof. We prove the lemma using a sequence of four games.

Game 0. This is the privacy against the public game in Figure 2 and applied to the generic scheme $\mathcal{S}$. If W_0 is the event that the output of the game is 1 then

$$\mathbf{Adv}^{\text{priv1}}_{\mathcal{A},\mathcal{S}}(\lambda) = \big|2\Pr[W_0] - 1\big|. \tag{14}$$

Game 1. Game 1 is the same as Game 0 except that the response to $\mathcal{O}_2(\cdot,\cdot)$ tracing queries is always fail. Recall that the game in Figure 2 allows the adversary to query $\mathcal{O}_2$ with any (m_i, σ_i) pair, but requires that (m_i, σ_i) have *not* been obtained by querying the signing oracle $\mathcal{O}_1$. As such, if the signature scheme $\mathcal{SIG}$ is strongly unforgeable, then the adversary's advantage in Game 1 is at most negligibly different from its advantage in Game 0. In particular, if W_1 is the event that the output of the game is 1 then there is an adversary $\mathcal{B}_1$ such that

$$\big|\Pr[W_1] - \Pr[W_0]\big| \leq \mathbf{Adv}^{\text{eufcma}}_{\mathcal{B}_1,\mathcal{SIG}}(\lambda) \tag{15}$$

Game 2. Game 2 is the same as Game 1 except that the signing oracle $\mathcal{O}_1(C_0, C_1, m)$ is modified to behave as follows: when outputting the response $\sigma = (ct, \pi, tg)$, the proof π in Step 3 of *Combine* is

generated using the simulator *Sim* which is given the statement $(\mathsf{com}_{pk}, pk_{\mathrm{t}}, m, ct)$ as input. Because the simulator generates proofs that are computationally indistinguishable from real proofs, the adversary's advantage in Game 2 is at most negligibly different from its advantage in Game 1. In particular, if W_2 is the event that the output of the game is 1 then there is an adversary $\mathcal{B}_3$ such that

$$\big|\Pr[W_2] - \Pr[W_1]\big| \leq Q \cdot \mathbf{Adv}^{\mathrm{hvzk}}_{\mathcal{B}_3,(P,V)}(\lambda) \tag{16}$$

where the factor of Q is due to a hybrid argument across $\mathcal{A}$'s signing queries.

Game 3. Game 3 is the same as Game 2 except that Step 2 of $\mathcal{S}.KeyGen$ is modified so that $\mathcal{B}$ computes $\mathsf{com}_{pk} = \mathcal{COM}.Commit(0, r_{pk})$. That is, com is a commitment to 0 instead of a commitment to pk'. Note that this step of *KeyGen* is the only place in Game 2 where pk' is used. Therefore, because the commitment is hiding, the adversary's advantage in Game 3 is at most negligibly different from its advantage in Game 2. In particular, if W_3 is the event that the output of the game is 1 then

$$\big|\Pr[W_3] - \Pr[W_2]\big| \leq \epsilon(\lambda) \tag{17}$$

where $\epsilon(\lambda)$ is the hiding statistical distance of the commitment scheme.

Game 4. Game 4 is the same as Game 3 except that the signing oracle $\mathcal{O}_1(C_0, C_1, m)$ is modified so that Step 2 of *Combine* now computes $ct \leftarrow \mathcal{PKE}.Encrypt(pk_{\mathrm{t}}, 0; r)$. That is, the encrypted plaintext is 0 instead of an ATS signature. Note that this step is the only place in Game 3 where the plaintext ATS signature is used. Therefore, because $\mathcal{PKE}$ is semantically secure, the adversary's advantage in Game 4 is at most negligibly different from its advantage in Game 3. In particular, if W_4 is the event that the output of the game is 1 then there is an adversary $\mathcal{B}_2$ such that

$$\big|\Pr[W_4] - \Pr[W_3]\big| \leq \mathbf{Adv}^{\mathrm{indcpa}}_{\mathcal{B}_2,\mathcal{PKE}}(\lambda) \tag{18}$$

Finishing the proof. Finally, in Game 4, the adversary's view is independent of the bit b. Therefore, the adversary has advantage zero in Game 4. In particular,

$$\Pr[W_4] = 1/2. \tag{19}$$

It follows that the adversary has at most negligible advantage in Game 0. Concretely, combining (14)–(19) proves (5). This completes the proof of the lemma. □

C Proof of Lemma 4

Proof. We prove the lemma using a sequence of games.

Game 0. This is the unforgeability and accountability game in Figure 1, applied to our Schnorr TAPS. If W_0 is the event that $\mathcal{A}$ wins in Game 0, then

$$\mathbf{Adv}^{\mathrm{forg}}_{\mathcal{A},\mathcal{S}}(\lambda) = \Pr[W_0] \tag{20}$$

Game 1. Let $pk = (pk_1, \ldots, pk_n, pk_{\mathrm{t}}, pk_{\mathrm{cs}}, T_0, T_1)$ be the public key given to the adversary. In addition, the adversary is given the combiner's secret key $sk_{\mathrm{c}} = (pk', pk_{\mathrm{t}}, sk_{\mathrm{cs}}, t, \psi)$. Game 1 is identical to Game 0 except that we strengthen the winning condition in that the adversary must output a valid forgery (m', σ') where $\sigma' = (R', ct', \pi', tg')$, along with a witness $(z, \rho, \psi, b_1, \ldots, b_n)$ such that

$$\mathcal{R}_{\mathcal{S}}\Big((pk_1, \ldots, pk_n, pk_{\mathrm{t}}, T_0, T_1, R', c', ct')\ ;\ (z, \rho, \psi, b_1, \ldots, b_n)\Big) = \text{true}. \tag{21}$$

where $\mathcal{R}_S$ is defined in (6) and $c' \leftarrow H(R', m')$.

We build a Game 1 adversary $\mathcal{A}'$ from a Game 0 adversary $\mathcal{A}$. $\mathcal{A}'$ runs $\mathcal{A}$, answering all of its queries, until it obtains from $\mathcal{A}$ a message m' and (R', ct') to form a valid $\mathcal{R}_S$ statement $(pk_1, \ldots, pk_n, pk_{\mathrm{t}}, T_0, T_1, R', c', ct')$. Next, $\mathcal{A}'$ runs the extractor *Ext* for the proof system (P, V) on the remaining execution of $\mathcal{A}$. *Ext* outputs a witness $w := (z'', \rho'', \psi'', b_1'', \ldots, b_n'')$. If w is a valid witness, then using w and sk_{c}, $\mathcal{A}'$ generates π' and

tg' so that $\sigma' := (R', ct', \pi', tg')$ is a valid signature for m' with respect to the public key pk. It outputs (m', σ') and w.

By definition of the extractor, if W_1 is the event that $\mathcal{A}'$ wins in Game 1, then

$$\Pr[W_1] \geq (\Pr[W_0] - \kappa(\lambda))/q(\lambda) \tag{22}$$

Finishing the Proof. Next, we show that an adversary $\mathcal{A}' = (\mathcal{A}_0, \mathcal{A}_1)$ that can win in Game 1 can be used to construct an ATS adversary $\mathcal{B}$ to break security of the underlying Schnorr ATS with the same advantage as $\mathcal{A}'$. In particular, we show that

$$\mathbf{Adv}^{\text{forg}}_{\mathcal{B},\mathcal{ATS}}(\lambda) \geq \Pr[W_1] \tag{23}$$

Adversary $\mathcal{B}$ works as follows:

- run $\mathcal{A}_0$ and receive from $\mathcal{A}_0$ the tuple $(n, t, C, \mathtt{state}_0)$ where $C \subseteq [n]$.
- send (n, t, C) to its own Schnorr ATS challenger and receives back $pk' = (pk_1, \ldots, pk_n)$ and $(sk_1, \ldots, sk_n)$.
- Generate $(pk_{\text{cs}}, sk_{\text{cs}}) \xleftarrow{\$} \mathcal{SIG}.KeyGen(\lambda)$ and $sk_{\text{e}} \xleftarrow{\$} \mathbb{Z}_q$.
- Encrypt t with ElGamal: $\psi \xleftarrow{\$} \mathbb{Z}_q$ and $(T_0, T_1) \leftarrow (g^{\psi},\ g^t h^{\psi})$.
- Construct $pk, sk_{\text{c}}, sk_{\text{t}}$ as in the *KeyGen* procedure.
- run $\mathcal{A}_1(pk, \{sk_i\}_{i \in C}, sk_{\text{c}}, sk_{\text{t}}, \mathtt{state}_0)$.
- $\mathcal{A}_1$ issues signing queries, which $\mathcal{B}$ answers by querying its own signing oracle. $\mathcal{B}$ receives $\sigma_{mi} = (R_i, z_i)$ in response, constructs a TAPS signature, and sends the result to $\mathcal{A}_1$.
- Eventually $\mathcal{A}_1$ outputs a forgery $(m', \sigma') = (m', (R', ct', \pi', tg'))$ along with a witness $(z, \rho, \psi, b_1, \ldots, b_n)$ such that (21) holds. In particular, thanks to condition (4) in Figure 6 we know that $b_i \in \{0, 1\}$ for $i \in [n]$.
- Let $C \subseteq [n]$ be the set whose characteristic vector is $(b_1, \ldots, b_n) \in \{0, 1\}^n$.
- Output the Schnorr ATS forgery $\big(m',\ \sigma = (R', z, C)\big)$.

We argue that if $\mathcal{A}' = (\mathcal{A}_0, \mathcal{A}_1)$ wins in Game 1, then $\big(m',\ \sigma = (R', z, C)\big)$ output by $\mathcal{B}$ is a forgery for the underlying Schnorr ATS. First, condition (1) in Figure 6 implies that $\big(m',\ \sigma = (R', z, C)\big)$ is a valid Schnorr ATS signature with respect to pk'. Second, condition (3) implies that $|C| = t$. Third, condition (2) implies that the Schnorr TAPS tracing algorithm will compute g^z in its step 3 and then output the set C. Since $\mathcal{A}'$ won in Game 1, this C is such that $\big(m',\ (R', z, C)\big)$ is a valid forgery for the Schnorr ATS.

Now, combining (20), (22) and (23) proves (7), as required. □

D Proof of Lemma 5

Proof. We prove the lemma using a sequence of games.

Game 0. This is the privacy against the public game in Figure 2 and applied to the Schnorr scheme $\mathcal{S}$. If W_0 is the event that the output of the game is 1 then

$$\mathbf{Adv}^{\text{priv1}}_{\mathcal{A},\mathcal{S}}(\lambda) = \big|2\Pr[W_0] - 1\big|. \tag{24}$$

Game 1. Game 1 is the same as Game 0 except that the response to $\mathcal{O}_2(\cdot,\cdot)$ tracing queries is always fail. Recall that any query the adversary makes to $\mathcal{O}_2$ cannot be the output from the signing oracle $\mathcal{O}_1$. Therefore, if the signature scheme $\mathcal{SIG}$ is strongly unforgeable, as in Definition 3, then the adversary's advantage in Game 1 is at most negligibly different from its advantage in Game 0. As such, if W_1 is the event that the output of the game is 1 then there is an adversary $\mathcal{B}_1$ such that

$$\big|\Pr[W_1] - \Pr[W_0]\big| \leq \mathbf{Adv}^{\text{eufcma}}_{\mathcal{B}_1,\mathcal{SIG}}(\lambda) \tag{25}$$

Game 2. Game 2 is the same as Game 1 except that the signing oracle $\mathcal{O}_1(C_0, C_1, m)$ is modified to behave as follows: when outputting the response $\sigma = (R, ct, \pi, tg)$, the proof π in Step 4 of *Combine* is generated using the zero knowledge simulator *Sim* which is only given the statement $(pk_1, \ldots, pk_n, pk_{\text{t}}, T_0, T_1, R, c, ct)$,

where $c \leftarrow H(pk, R, m)$. Because the simulator generates proofs that are computationally indistinguishable from real proofs, the adversary's advantage in Game 2 is at most negligibly different from its advantage in Game 1. In particular, if W_2 is the event that the output of the game is 1 then there is an adversary $\mathcal{B}_2$ such that

$$\left|\Pr[W_2] - \Pr[W_1]\right| \leq Q \cdot \mathbf{Adv}^{\text{hvzk}}_{\mathcal{B}_2,(P,V)}(\lambda) \tag{26}$$

where the factor of Q is due to a hybrid argument across $\mathcal{A}$'s signing queries.

Game 3. Game 3 is the same as Game 2 except that Step 2 of *KeyGen* is modified so that $\mathcal{B}$ computes the ElGamal ciphertext (T_0, T_1) as $(T_0, T_1) \xleftarrow{\$} \mathbb{G}^2$, instead of computing an ElGamal encryption of g^t. Note that in Game 2, the threshold t is not used anywhere other than in Step 2 of *KeyGen*. Hence, since DDH holds in $\mathbb{G}$, the adversary's advantage in Game 3 is at most negligibly different from its advantage in Game 2. Concretely, if W_3 is the event that the output of the game is 1 then there is a DDH adversary $\mathcal{B}_3$ such that

$$\left|\Pr[W_3] - \Pr[W_2]\right| \leq \mathbf{Adv}^{\text{ddh}}_{\mathcal{B}_3,\mathbb{G}}(\lambda) \tag{27}$$

Game 4. Game 4 is the same as Game 3 except that the signing oracle $\mathcal{O}_1(C_0, C_1, m)$ is modified so that Step 2 of *Combine* now computes the ElGamal ciphertext $ct := (c_0, c_1)$ as $ct \xleftarrow{\$} \mathbb{G}^2$. That is, the ciphertext is chosen at random rather than as an ElGamal encryption of z. Note that z is never used in Game 3 other than in this Step 2 of *Combine*. Therefore, because DDH holds in $\mathbb{G}$, the adversary's advantage in Game 4 is at most negligibly different from its advantage in Game 3. Concretely, if W_4 is the event that the output of the game is 1 then there is a DDH adversary $\mathcal{B}'_3$ such that

$$\left|\Pr[W_4] - \Pr[W_3]\right| \leq Q \cdot \mathbf{Adv}^{\text{ddh}}_{\mathcal{B}'_3,\mathbb{G}}(\lambda) \tag{28}$$

where the factor of Q is due to a hybrid argument across $\mathcal{A}$'s signing queries.

Finishing the Proof. The adversary's view in Game 4 is independent of the bit b, and therefore the adversary has advantage zero in Game 4. It follows that the adversary has at most negligible advantage in Game 0. Concretely, combining (25)–(28) proves (8). This completes the proof of the lemma. □

E Sigma Protocol Proof for Schnorr-based TAPS

The last step of Protocol S1 from Section 5.3 uses a Sigma protocol for proving the relation $\mathcal{R}_{S1}$ from Figure 7. We now describe how the prover performs this protocol, interacting with the verifier. We then describe the result of making this protocol non-interactive via the Fiat-Shamir transform.

Recall that both the prover and verifier have the statement

$$\Big(g, h, h_1, \ldots, h_n, pk_1, \ldots, pk_n, pk_{\text{t}}, T_0, T_1, R, c, ct = (c_0, c_1), v_0, v_1, \ldots, v_n, \alpha\Big)$$

and the prover has the witness

$$\Big(z, \rho, \psi, \gamma, b_1, \ldots, b_n, \phi_1, \ldots, \phi_n\Big).$$

The prover and verifier then jointly perform the protocol $S1'$.

Protocol $S1'$:

1: The prover chooses blinding factors:

$$k_z, k_\rho, k_\gamma, k_\psi,\ (k_{b1}, \ldots, k_{bn}),\ (k_{\phi 1}, \ldots, k_{\phi n}) \xleftarrow{\$} \mathbb{Z}_q$$

and computes the following commitments in $\mathbb{G}$:

$$
\begin{aligned}
S_1 &\leftarrow g^{k_z} \cdot \prod_{i=1}^{n} pk_i^{-c \cdot k_{bi}} \\
S_{2a} &\leftarrow g^{k_\rho}; \qquad S_{2b} \leftarrow pk_{\mathrm{t}}^{k_\rho} \cdot g^{k_z} \\
S_{3a} &\leftarrow g^{k_\psi}; \qquad S_{3b} \leftarrow g^{\sum_{i=1}^{n} k_{bi}} \cdot h^{k_\psi} \\
S_{4a} &\leftarrow g^{k_\gamma}; \qquad \forall\ i \in \{1, \ldots, n\}\ :\ S_{4bi} \leftarrow g^{k_{bi}} h_i^{k_\gamma} \\
S_{4c} &\leftarrow \prod_{i=1}^{n} {v_i}^{\alpha^i k_{bi}} {h_i}^{k_{\phi i}}
\end{aligned}
\tag{29}
$$

2: The prover sends $S_1, S_{2a}, S_{2b}, S_{3a}, S_{3b}, S_{4a}, S_{4b1}, \ldots, S_{4bn}, S_{4c} \in \mathbb{G}$ to the verifier.

3: The verifier chooses a random challenge $\beta \xleftarrow{\$} \mathbb{Z}_q$ and sends β to the prover.

4: The prover computes:

$$
\begin{aligned}
\hat{z} &\leftarrow z \cdot \beta + k_z \\
\hat{\rho} &\leftarrow \rho \cdot \beta + k_\rho \\
\hat{\gamma} &\leftarrow \gamma \cdot \beta + k_\gamma \\
\hat{\psi} &\leftarrow \psi \cdot \beta + k_\psi \\
\forall\ i \in \{1, \ldots, n\}\ &: \hat{b_i} \leftarrow b_i \cdot \beta + k_{bi} \\
\forall\ i \in \{1, \ldots, n\}\ &: \hat{\phi_i} \leftarrow \phi_i \cdot \beta + k_{\phi i}
\end{aligned}
\tag{30}
$$

and sends $\hat{z}, \hat{\rho}, \hat{\gamma}, \hat{\psi}, \hat{b_i}, \hat{\phi_i} \in \mathbb{Z}_q$ for $i = 1, \ldots, n$ to the verifier.

5: The verifier outputs "accept" if

$$
S_1 \cdot R^\beta \cdot \Big[\prod_{i=1}^{n} pk_i^{\hat{b_i}}\Big]^c \stackrel{?}{=} g^{\hat{z}} \qquad \text{Verify (1)}
$$

$$
S_{2a} \cdot {c_0}^\beta \stackrel{?}{=} g^{\hat{\rho}} \quad \text{and} \quad S_{2b} \cdot {c_1}^\beta \stackrel{?}{=} pk_{\mathrm{t}}^{\hat{\rho}} \cdot g^{\hat{z}} \qquad \text{Verify (2)}
$$

$$
S_{3a} \cdot {T_0}^\beta \stackrel{?}{=} g^{\hat{\psi}} \quad \text{and} \quad S_{3b} \cdot T_1^\beta \stackrel{?}{=} \Big(\prod_{i=1}^{n} g^{\hat{b_i}}\Big) \cdot h^{\hat{\psi}} \qquad \text{Verify (3)}
$$

$$
S_{4a} \cdot v_0^\beta \stackrel{?}{=} g^{\hat{\gamma}} \quad \text{and} \quad \forall\ i \in \{1, \ldots, n\}\ :\ S_{4bi} \cdot {v_i}^\beta \stackrel{?}{=} g^{\hat{b_i}} {h_i}^{\hat{\gamma}}
$$

$$
\text{and} \quad S_{4c} \cdot \prod_{i=1}^{n} v_i^{\beta \cdot \alpha^i} \stackrel{?}{=} \prod_{i=1}^{n} v_i^{\alpha^i \hat{b_i}} h_i^{\hat{\phi_i}} \qquad \text{Verify (4)}
$$

This completes the description of the protocol. We next state its security theorem.

Theorem 6. *The protocol is a zero-knowledge proof of knowledge for the relation $\mathcal{R}_{S1}$ from Figure 7.*

The proof of Theorem 6 follows from the general theory of Sigma protocols (see, e.g. [19, example 19.5.3]).

Applying the Fiat-Shamir transform. The Fiat-Shamir transform [35] can be used to make this protocol non-interactive. The resulting proof π for the relation $\mathcal{R}_{S1}$ contains the challenge β and the prover's responses, as follows:

$$
\pi = \Big\{ \quad \beta, \hat{z}, \hat{\rho}, \hat{\gamma}, \hat{\psi}, \hat{b_1}, \ldots, \hat{b_n}, \hat{\phi_1}, \ldots, \hat{\phi_n} \in \mathbb{Z}_q \quad \Big\} \tag{31}
$$

The complete proof. The complete proof π generated by $\mathcal{S}.Combine$ from Figure 5 is a proof for the relation $\mathcal{R}_S$ from Figure 6. This proof is generated using Protocol $S1$ after it is made non-interactive using Fiat-Shamir. This proof π includes all the elements in (31) above along with the elements $(v_0, v_1, \ldots, v_n) \in \mathbb{G}^{n+1}$ generated in step 1 of Protocol $S1$. The quantity $\alpha \in \mathbb{Z}_q$ used in Step 2 of Protocol $S1$ is generated via the Fiat-Shamir transform and is not included in the proof.

F Proof of Theorem 3

Proof. Completeness is immediate. The HVZK property follows directly from the DDH assumption and from the HVZK property of the sigma protocol for the relation $\mathcal{R}_{S1}$ in Figure 7.

It remains to prove that the protocol is an argument of knowledge for $\mathcal{R}_S$. The sigma protocol extractor for the relation $\mathcal{R}_{S1}$ in Figure 7 will extract from a malicious prover a valid witness $(z, \rho, \psi, \gamma, b_1, \ldots, b_n, \phi_1, \ldots, \phi_n)$ for $\mathcal{R}_{S1}$. As Equations (1), (2), (3) are the same in both $\mathcal{R}_S$ and $\mathcal{R}_{S1}$, we need only prove that equation (4) in $\mathcal{R}_{S1}$ implies that $b_i(1-b_i) = 0$ for all $i \in [n]$ (and consequently, $b_i \in \{0, 1\}$).

Suppose $\mathcal{A}$ is able to produce a vector $(v_0, v_1, \ldots, v_n) \in \mathbb{G}^{n+1}$ such that the extracted witness satisfies Equation (4) in $\mathcal{R}_{S1}$, but $b_i(1-b_i) \neq 0$ for some $i \in [n]$. We show that this $\mathcal{A}$ can be used to solve discrete log in $\mathbb{G}$.

By Equation (4), the extracted witness must satisfy

$$\prod_{i=1}^{n} (g^{b_i} h_i^{\gamma})^{\alpha^i (1-b_i)} = \prod_{i=1}^{n} h_i^{\phi_i}.$$

This leads to

$$g^{\kappa} = \prod_{i=1}^{n} h_i^{\phi_i - \alpha^i \gamma (1-b_i)} \quad \text{where} \quad \kappa := \sum_{i=1}^{n} \alpha^i b_i (1-b_i).$$

Since $b_i(1-b_i) \neq 0$ for some $i \in [n]$, and α is chosen at random after the prover commits to $b_1, \ldots, b_n$, it follows that $\kappa \neq 0$ with probability at least $1 - n/q$. Hence, with probability close to 1 we obtain a non-trivial relation between the random independent generators $g, h_1, \ldots, h_n$. If $\mathcal{A}$ can produce such a relation, then $\mathcal{A}$ can be used to solve discrete log in $\mathbb{G}$ by a standard reduction (see, e.g., [19, ex. 10.19]). Therefore, if discrete log in $\mathbb{G}$ is hard and the extracted witness satisfies Equation (4), then it must be that $b_i(1-b_i) = 0$ for all $i \in [n]$. □

G Proof of Theorem 5

We say that $\mathcal{P}_1$ is an **eBP instance generator** if $\mathcal{P}_1$ generates an $\mathcal{R}_{\text{eBP}}$ statement as follows: $\mathcal{P}_1$ first outputs some $\mathbf{V} = (v_{i,j}) \in \mathbb{G}^{\ell \times n}$ and $\mathbf{u}' \in \mathbb{G}^{\ell}$, then an honest party samples a random $\mathbf{a} \xleftarrow{\$} \mathbb{G}^n$, and finally $\mathcal{P}_1(\mathbf{a})$ outputs an R1CS program P and $u \in \mathbb{G}$. We say that $\mathcal{P}_1$ generated the $\mathcal{R}_{\text{eBP}}$ statement $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$. We write $\mathcal{P}_1 \in \mathsf{IG}_{\text{eBP}}$ to indicate that $\mathcal{P}_1$ is an eBP instance generator.

Let us restate Theorem 5 and then prove it.

Theorem 5. *Suppose that the discrete log assumption holds in the group $\mathbb{G}$. Then there is an HVZK restricted argument of knowledge as in Definition 10 for $\mathcal{R}_{eBP}$ where the proof size is the same as in the bulletproofs system for $\mathcal{R}_{BP}$, and proof generation and verification times differ by at most the time to compute $(n+1)\ell$ exponentiations in $\mathbb{G}$. The restricted argument of knowledge property holds with respect to an eBP instance generator $\mathcal{P}_1 \in \mathsf{IG}_{eBP}$.*

To prove the theorem we note that bulletproofs [22,24] operates in two steps. In the first step, a compiler is used to compile the relation

$$\mathcal{R}_{\text{BP}} := \left\{ (P,\ \mathbf{a} \in \mathbb{G}^n,\ u \in \mathbb{G})\ ;\ \mathbf{w} \in \mathbb{Z}_q^n \right\} \qquad \text{iff } P(\mathbf{w}) = 1 \text{ and } \mathbf{a}^{\mathbf{w}} = u,$$

into an inner-product relation

$$\mathcal{R}_{\text{IP}} := \left\{ \left(\mathbf{b}, \mathbf{c} \in \mathbb{G}^{n'},\ g, h \in \mathbb{G}\right)\ ;\ \left(\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{Z}_q^{n'},\ \delta \in \mathbb{Z}_q\right) \right\} \\ \text{where } h = \mathbf{b}^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^{\delta} \text{ and } \langle \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle = \delta \tag{32}$$

for some $n' \geq n$. Here $\langle \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle$ is the inner product of the vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, and $\mathbf{b}, \mathbf{c}, g$ are random generators of $\mathbb{G}$. The compiler ensures that an HVZK argument of knowledge for this $\mathcal{R}_{\text{IP}}$ instance implies an HVZK argument of knowledge for the original $\mathcal{R}_{\text{BP}}$ instance. We note that this compiler is itself a public coin interactive protocol.

The second step, is an HVZK argument of knowledge for $\mathcal{R}_{\text{IP}}$. To do so, bulletproofs uses a $\log_2(n')$-round protocol between the prover and the verifier called an *inner-produce argument* (IPA). There is an extractor, called the IPA extractor Ext_{IPA} with the following property. For every prover $(\mathcal{P}_1, \mathcal{P}_2)$, if $(\mathbf{b}, \mathbf{c}, g, h) \xleftarrow{\$} \mathcal{P}_1(\lambda)$ is an $\mathcal{R}_{\text{IP}}$ statement for which $\mathcal{P}_2$ successfully convinces the verifier, then the IPA extractor extracts $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ from $\mathcal{P}_2$ such that either

- $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ is a valid $\mathcal{R}_{\text{IP}}$ witness for $(\mathbf{b}, \mathbf{c}, g, h)$, or
- $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ is a relation among the generators $\mathbf{b}, \mathbf{c}, g$, namely $\mathbf{b}^{\boldsymbol{\beta}}\mathbf{c}^{\boldsymbol{\gamma}}g^{\delta} = 1$.

Hence, if discrete log in $\mathbb{G}$ is hard and $\mathbf{b}, \mathbf{c}, g$ are random independent generators in $\mathbb{G}$, then the latter case cannot happen, and the IPA extractor Ext_{IPA} must extract a valid witness for the $\mathcal{R}_{\text{IP}}$ statement $(\mathbf{b}, \mathbf{c}, g, h)$. Although the IPA protocol from [22,24] is not designed to be HVZK, it can be made HVZK by adding a round of blinding.

The protocol for $\mathcal{R}_{\text{eBP}}$. We next describe the required HVZK argument of knowledge for the relation $\mathcal{R}_{\text{eBP}}$ defined in Section 5.4. Recall that

$$\mathcal{R}_{\text{eBP}} := \left\{ (P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}') \;;\; \mathbf{w} \in \mathbb{Z}_q^n \right\} \quad \text{iff } P(\mathbf{w}) = 1, \;\; \mathbf{a}^{\mathbf{w}} = u, \;\; \text{and } \mathbf{V}^{\mathbf{w}} = \mathbf{u}',$$

where $\mathbf{V} \in \mathbb{G}^{\ell \times n}$ and $\mathbf{u}' \in \mathbb{G}^{\ell}$. We can treat $(P, \mathbf{a}, u)$ as a $\mathcal{R}_{\text{BP}}$ instance and compile it to a $\mathcal{R}_{\text{IP}}$ instance $(\mathbf{b}, \mathbf{c}, g, h)$ with witness $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$. Recall from Eq. (32) that

$$\mathbf{b}^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^{\delta} = h \qquad \text{and} \qquad \langle \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle = \delta.$$

The compiler has an important property: it compiles a witness $\mathbf{w} \in \mathbb{Z}_q^n$ for $\mathcal{R}_{\text{BP}}$ into a witness $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ for $\mathcal{R}_{\text{IP}}$ where $\mathbf{w}$ is a prefix of $\boldsymbol{\beta}$. In particular, we can write

$$\begin{gathered} \boldsymbol{\beta} = (\mathbf{w}, \boldsymbol{\beta}') \quad \text{and} \quad \mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2) \\ \text{where } \boldsymbol{\beta}' \in \mathbb{Z}_q^{n'-n},\ \mathbf{b}_1 \in \mathbb{G}^n, \text{ and } \mathbf{b}_2 \in \mathbb{G}^{n'-n}. \end{gathered} \tag{33}$$

Now, the protocol for an $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$ is shown in Figure 9. The prover's and verifier's running time beyond the inner product argument is dominated by step 2 which is about the time to compute $(n+1)\ell$ exponentiations in $\mathbb{G}$, as required.

Security. It remains to show that Protocol eBP is an HVZK argument of knowledge for $\mathcal{R}_{\text{eBP}}$. Completeness and HVZK follow immediately from these properties of the inner product argument in step 3. It remains to construct an extractor for a convincing prover P^*. To construct the extractor, it is convenient to first define the following game between an adversary $\mathcal{A}$ and a challenger:

The adaptive vector discrete log game in a group $\mathbb{G}$ of order q:

1: $\mathcal{A}$ begins by choosing parameters $m, k \in \mathbb{N}$ and a vector $\mathbf{v} \in \mathbb{G}^m$. $\mathcal{A}$ then sends $(m, k, \mathbf{v})$ to the challenger.

2: The challenger samples $\mathbf{a} \xleftarrow{\$} \mathbb{G}^k$ and sends $\mathbf{a}$ to $\mathcal{A}$,

3: $\mathcal{A}$ outputs $\boldsymbol{\alpha} = (\boldsymbol{\nu}, \boldsymbol{\omega}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^k$.

Let $\mathbf{h} := (\mathbf{v} \| \mathbf{a}) \in \mathbb{G}^{m+k}$. The adversary wins the game if $\boldsymbol{\omega} \neq \mathbf{0}$ and $\mathbf{h}^{\boldsymbol{\alpha}} = 1$, where 1 is the identity element of $\mathbb{G}$. The condition $\boldsymbol{\omega} \neq \mathbf{0}$ is needed to ensure that $\mathcal{A}$ cannot trivially satisfy the relation $\mathbf{h}^{\boldsymbol{\alpha}} = 1$.

For a group family $\mathbb{G} = \{\mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$, let $\mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda)$ be the probability that $\mathcal{A}$ wins the game in the group $\mathbb{G}_\lambda$.

Definition 11. *We say that the* **adaptive vector discrete log assumption** *(AVDL) holds in the group family $\mathbb{G} = \{\mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$ if for every PPT adversary $\mathcal{A}$ the function $\mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda)$ is negligible.*

Lemma 7. *Let $\mathbb{G} = \{\mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of prime order cyclic groups where $q(\lambda)$ is the order of $\mathbb{G}_\lambda$. If the discrete log assumption holds in $\mathbb{G}$, then so does the adaptive vector discrete log assumption. In particular, for every AVDL adversary $\mathcal{A}$ that outputs m in step 1 of the adaptive vector discrete log game, there exists a discrete log adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda) \leq \mathbf{Adv}^{\text{dlog}}_{\mathcal{B},\mathbb{G}}(\lambda) + 1/q(\lambda) \tag{35}$$

where $E[time(\mathcal{B})] \leq (m+1) \cdot time(\mathcal{A})/\epsilon$ for $\epsilon := \mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda)$.

Protocol eBP for an $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$:

1: The verifier choose a random $\xi \xleftarrow{\$} \mathbb{Z}_q$ and sends ξ to the prover.

2: For $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$, let $\mathbf{v}_1, \ldots, \mathbf{v}_\ell \in \mathbb{G}^n$ be the rows of the matrix $\mathbf{V}$, and let $\mathbf{u}' = (u_1, \ldots, u_\ell) \in \mathbb{G}^\ell$. Suppose that $(P, \mathbf{a}, u)$ compiles to an $\mathcal{R}_{\text{IP}}$ instance $(\mathbf{b}, \mathbf{c}, g, h)$, where $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2) \in \mathbb{G}^{n'}$ as in (33). The witness $\mathbf{w} \in \mathbb{Z}_q^n$ compiles to an $\mathcal{R}_{\text{IP}}$ witness $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ where $\boldsymbol{\beta} = (\mathbf{w}, \boldsymbol{\beta}') \in \mathbb{Z}_q^{n'}$.

The prover and verifier compute

$$\begin{aligned} \hat{\mathbf{b}}_1 &\leftarrow \mathbf{b}_1 \circ \mathbf{v}_1^{\xi} \circ \mathbf{v}_2^{(\xi^2)} \circ \cdots \circ \mathbf{v}_\ell^{(\xi^\ell)} \in \mathbb{G}^n \\ \hat{h} &\leftarrow h \cdot u_1^{\xi} \cdot u_2^{(\xi^2)} \cdots u_\ell^{(\xi^\ell)} \in \mathbb{G} \\ \hat{\mathbf{b}} &\leftarrow (\hat{\mathbf{b}}_1, \mathbf{b}_2) \in \mathbb{G}^{n'} \end{aligned} \tag{34}$$

where $\mathbf{b} \circ \mathbf{v}$ denotes the Hadamard product of the vectors $\mathbf{b}$ and $\mathbf{v}$. Since $\mathbf{b}^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^\delta = h$ and $\boldsymbol{\beta} = (\mathbf{w}, \boldsymbol{\beta}')$ and $\mathbf{v}_i^{\mathbf{w}} = u_i$ for $i = 1, \ldots, \ell$, it follows that

$$(\hat{\mathbf{b}})^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^\delta = \hat{h}.$$

3: The prover and verifier now execute the HVZK inner-product argument to convince the verifier that the prover has a valid witness for the $\mathcal{R}_{\text{IP}}$ instance $(\hat{\mathbf{b}}, \mathbf{c}, g, \hat{h})$. The prover's witness is $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$.

Fig. 9. A protocol for an $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$.

Proof. Let $\mathbb{G}$ be a group of of prime order q in the given group family. We use adversary $\mathcal{A}$ to build an adversary $\mathcal{B}$ that can solve the discrete log problem in $\mathbb{G}$. Algorithm $\mathcal{B}$ is given random $g, h \in \mathbb{G}$ and needs to output $\kappa \in \mathbb{Z}_q$ such that $g^\kappa = h$.

Algorithm $\mathcal{B}$ is shown in Figure 10. The main part of the proof is to show that step (9) in the algorithm is valid, namely that $\gamma \neq 0$ with high probability. Recall that (37) in Figure 10 defines $\gamma \in \mathbb{Z}_q$ as

$$\gamma \leftarrow \sum_{j=1}^{m+1} \sum_{i=1}^{k} \lambda_j \gamma_{j,i} \cdot \omega_{j,i}.$$

Then $\gamma \in \mathbb{Z}_q$ is the direct product of two vectors $\vec{\gamma}$ and $\vec{\omega}$ in $\mathbb{Z}_q^{k(m+1)}$, where

$$\vec{\gamma} := \big(\gamma_{1,1}, \ldots, \gamma_{m+1,k}\big) \quad ; \qquad \vec{\omega} := \big(\lambda_1 \omega_{1,1},\ \ldots,\ \lambda_{m+1} \omega_{m+1,k}\big).$$

We make two observations about these vectors. First, the vector $\vec{\gamma}$, whose elements are sampled as in step (3) in Figure 10, is sampled uniformly in $\mathbb{Z}_q^{k(m+1)}$ and is independent of the adversary's view. Second, the vector $\vec{\omega}$ must be non-zero. To see why, choose some $j \in [m+1]$ such that $\lambda_j \neq 0$. Then if $\vec{\omega} = \mathbf{0}$ it must be that $\boldsymbol{\omega}_j = \mathbf{0}$, which violates the requirements on $\mathcal{A}$.

The first observation lets us treat $\vec{\gamma}$ as if it were sampled uniformly *after* the vector $\vec{\omega}$ is chosen. The second observation implies that the set of vectors orthogonal to $\vec{\omega}$ is a linear space of co-dimension one. Therefore the event that $\gamma = 0$ is the same as the event that a random vector in $\mathbb{Z}_q^{k(m+1)}$ falls in a linear space of co-dimension one, which happens with probability $1/q$. Hence,

$$\Pr\big[\gamma = 0\big] = 1/q.$$

Next, let $\mathcal{E}$ be the event that $\mathcal{B}$ does not abort in step (4). We know that $\Pr[\mathcal{E}] = \mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda)$ and therefore

$$\mathbf{Adv}^{\text{dlog}}_{\mathcal{B},\mathbb{G}}(\lambda) = \Pr\big[\mathcal{E} \text{ and } \gamma \neq 0\big] \geq \Pr\big[\mathcal{E}\big] - \Pr\big[\gamma = 0\big] = \mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda) - (1/q)$$

from which (35) follows.

Finally, the running time of $\mathcal{B}$ is dominated by its time in steps (4) and (6). Since each iteration in step (6) produces an output with probability $\epsilon = \mathbf{Adv}^{\text{avdl}}_{\mathcal{A},\mathbb{G}}(\lambda)$, the expected number of iterations in

Input: $g, h \in \mathbb{G}$, Output: $\kappa \in \mathbb{Z}_q$ such that $g^\kappa = h$.

1: Run $\mathcal{A}$ to get $m, k \in \mathbb{N}$ and a vector $\mathbf{v} \in \mathbb{G}^m$.

2: Sample $\beta_i, \gamma_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \in [k]$.

3: $a_i \leftarrow g^{\beta_i} h^{\gamma_i} \in \mathbb{G}$ and $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{G}^k$.

4: Send $\mathbf{a} \in \mathbb{G}^k$ to $\mathcal{A}$. If $\mathcal{A}$ outputs fail then abort.

5: Otherwise, $\mathcal{A}$ outputs $\boldsymbol{\alpha} = (\boldsymbol{\nu}, \boldsymbol{\omega}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^k$ where $\boldsymbol{\omega} \neq \mathbf{0}$ and $\mathbf{h}^{\boldsymbol{\alpha}} = 1$ for $\mathbf{h} := (\mathbf{v} \| \mathbf{a}) \in \mathbb{G}^{m+k}$.

6: Repeat:

rerun steps (1)–(5) while giving $\mathcal{A}$ the same randomness as in the first run, but choosing fresh $\beta_i, \gamma_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \in [k]$ in step (2).

until we obtain m successful outputs from $\mathcal{A}$.

7: From steps (4) and (6) we now have $m+1$ vectors: for $j = 1, \ldots, m+1$

$$\boldsymbol{\alpha}_j = (\boldsymbol{\nu}_j, \boldsymbol{\omega}_j) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^k \quad \text{where } \boldsymbol{\omega}_j \neq \mathbf{0} \text{ and } (\mathbf{v} \| \mathbf{a}_j)^{\alpha_j} = \mathbf{v}^{\boldsymbol{\nu}_j} \mathbf{a}_j^{\boldsymbol{\omega}_j} = 1 \tag{36}$$

Here $\mathbf{a}_j = \big(a_{j,1}, \ldots, a_{j,k}\big) \in \mathbb{G}^k$ where $a_{j,i} = g^{\beta_{j,i}} \cdot h^{\gamma_{j,i}} \in \mathbb{G}$ for $i \in [k]$.
We write $\boldsymbol{\omega}_j = \big(\omega_{j,1}, \ldots, \omega_{j,k}\big)$ for $j \in [m+1]$.

8: Find a non-zero vector $\boldsymbol{\lambda} := (\lambda_1, \ldots, \lambda_{m+1}) \in \mathbb{Z}_q^{m+1}$ such that

$$\sum_{j=1}^{m+1} \lambda_j \cdot \boldsymbol{\nu}_j = \mathbf{0}.$$

Then by taking the weighted product of the $(m+1)$ relations in (36) we obtain

$$(\mathbf{a}_1)^{\lambda_1 \boldsymbol{\omega}_1} \cdot (\mathbf{a}_2)^{\lambda_2 \boldsymbol{\omega}_2} \cdots (\mathbf{a}_{m+1})^{\lambda_{m+1} \boldsymbol{\omega}_{m+1}} = 1.$$

By definition of $\mathbf{a}_1, \ldots, \mathbf{a}_{m+1} \in \mathbb{G}^k$, this implies that $g^\beta h^\gamma = 1$ where $\beta, \gamma \in \mathbb{Z}_q$ are defined as

$$\beta \leftarrow \sum_{j=1}^{m+1} \sum_{i=1}^{k} \lambda_j \beta_{j,i} \cdot \omega_{j,i} \qquad \text{and} \qquad \gamma \leftarrow \sum_{j=1}^{m+1} \sum_{i=1}^{k} \lambda_j \gamma_{j,i} \cdot \omega_{j,i} \tag{37}$$

9: Output $\kappa \leftarrow -\beta/\gamma \in \mathbb{Z}_q$. We prove that $\gamma \neq 0$ w.h.p in the body of the proof.

Fig. 10. The discrete log algorithm $\mathcal{B}$ from an AVDL adversary $\mathcal{A}$.

On input $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$ do:

1: For $i = 1, \ldots, \ell + 1$ do:
   a: Choose $\xi_i \xleftarrow{\$} \mathbb{Z}_q$.
   b: Restart $\mathcal{P}_2(\mathtt{state})$ from the beginning sending it ξ_i in step 1 of Protocol eBP (Figure 9). Step 2 of the protocol constructs an $\mathcal{R}_{\text{IP}}$ instance $(\hat{\mathbf{b}}_i, \mathbf{c}, g, \hat{h}_i)$ which is known to both $\mathcal{P}_2$ and *Ext*.
   c: Run the IPA extractor Ext_{IPA} on $\mathcal{P}_2$ for the $\mathcal{R}_{\text{IP}}$ instance $(\hat{\mathbf{b}}_i, \mathbf{c}, g, \hat{h}_i)$. Ext_{IPA} outputs the extracted witness $\mathbf{0} \neq (\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i) \in \mathbb{Z}_q^{2n'+1}$.
2: Now *Ext* has $(\ell + 1)$ extracted witnesses $\mathbf{0} \neq (\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i) \in \mathbb{Z}_q^{2n'+1}$ for $i \in [\ell + 1]$.
3: If there is an $i \in [\ell+1]$ such that $(\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i)$ is not a valid witness for the $\mathcal{R}_{\text{IP}}$ instance $(\hat{\mathbf{b}}_i, \mathbf{c}, g, \hat{h}_i)$, then *Ext* outputs fail and stops.
4: If there are $i, j \in [\ell + 1]$ such that $(\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i) \neq (\boldsymbol{\beta}_j, \boldsymbol{\gamma}_j, \delta_j)$ then *Ext* outputs fail and stops.
5: Otherwise, $(\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i) = (\boldsymbol{\beta}_1, \boldsymbol{\gamma}_1, \delta_1)$ for all $i \in [\ell + 1]$. Write $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta) := (\boldsymbol{\beta}_1, \boldsymbol{\gamma}_1, \delta_1)$.
6: Write $\boldsymbol{\beta}$ as $\boldsymbol{\beta} = (\mathbf{w}, \boldsymbol{\beta}') \in \mathbb{Z}_q^n \times \mathbb{Z}_q^{n'-n}$; *Ext* outputs $\mathbf{w} \in \mathbb{Z}_q^n$ and stops.

Fig. 11. The extractor *Ext* for Protocol eBP in Figure 9.

step (6) is m/ϵ. Therefore

$$E\big[\text{time}(\mathcal{B})\big] \leq \underbrace{\text{time}(\mathcal{A})}_{\text{step (4)}} + \underbrace{(m/\epsilon) \cdot \text{time}(\mathcal{A})}_{\text{step (6)}} \leq \big((m+1)/\epsilon\big) \cdot \text{time}(\mathcal{A})$$

as claimed. □

The extractor for Protocol eBP. We now build the extractor for Protocol in Figure 9. This is captured in the following lemma that, in combination with Lemma 7, concludes the proof of Theorem 5.

Lemma 8. *Suppose that the adaptive vector discrete log assumption (AVDL) holds in* $\mathbb{G}$. *Let* $\mathcal{P}^* = (\mathcal{P}_1, \mathcal{P}_2)$ *be a prover in Protocol eBP, where the instance generator* $\mathcal{P}_1$ *is in* IG_{eBP} *as defined in Theorem 5. Then there is an extractor Ext such that the functions*

$$\epsilon_1(\lambda) := \Pr\big[\langle \mathcal{P}_2(\mathtt{state}); \mathcal{V}(x)\rangle = 1 \ : \ (x, \mathtt{state}) \xleftarrow{\$} \mathcal{P}_1(1^\lambda)\big]$$
$$\epsilon_2(\lambda) := Pr\big[(x, \mathbf{w}) \in \mathcal{R}_{eBP} \ : \ (x, \mathtt{state}) \xleftarrow{\$} \mathcal{P}_1(1^\lambda), \ \mathbf{w} \xleftarrow{\$} Ext^{\mathcal{P}_2(\mathtt{state})}(x)\big]$$

satisfy

$$\epsilon_2(\lambda) \geq \big(\epsilon_1(\lambda) - \kappa(\lambda)\big)/2q(\lambda), \tag{38}$$

where κ *and* q *are the knowledge error and tightness for the inner product argument (IPA) extractor* Ext_{IPA} *for* $\mathcal{R}_{IP}$. *Moreover, whenever* $\mathcal{P}_1$ *outputs a* $\mathcal{R}_{eBP}$ *instance* $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$ *where* $\mathbf{V} \in \mathbb{G}^{\ell \times n}$, *the expected running time of Ext is time*$(Ext) \leq O\big(2(\ell+1)time(Ext_{\text{IPA}})\big)$.

Proof. Let us describe how *Ext* works. Prover $\mathcal{P}_1(1^\lambda)$ runs and outputs a $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$. This instance is given as input to the extractor *Ext*, which then interacts with the oracle $\mathcal{P}_2(\mathtt{state})$, and needs to produce a witness $\mathbf{w} \in \mathbb{Z}_q^n$ for the $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$.

The extractor *Ext* uses the IPA extractor Ext_{IPA} and works as shown in Figure 11. For now let us assume that the IPA extractor Ext_{IPA} is perfect: whenever it is invoked on an $\mathcal{R}_{\text{IP}}$ instance $(\hat{\mathbf{b}}, \mathbf{c}, g, \hat{h})$ it either returns a valid $\mathcal{R}_{\text{IP}}$ witness or a valid relation among the generators $\hat{\mathbf{b}}, \mathbf{c}, g$. We will address an imperfect IPA extractor at the end of the proof.

We prove that the extractor *Ext* in Figure 11 satisfies the requirements of the lemma. There are three cases.

Case 1. *Ext* could fail in step 3. This means that for some $i \in [\ell+1]$, the extracted witness $(\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i) \in \mathbb{Z}_q^{2n'+1}$ in step 3 is not a valid witness for $(\hat{\mathbf{b}}_i, \mathbf{c}, g, \hat{h}_i)$. Instead, it must be a relation among the generators $\hat{\mathbf{b}}_i, \mathbf{c}, g$, namely

$$(\hat{\mathbf{b}}_i)^{\boldsymbol{\beta}_i} \cdot \mathbf{c}^{\boldsymbol{\gamma}_i} \cdot g^{\delta_i} = 1$$

and $\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i$ are not all zero. Write $\boldsymbol{\beta}_i = (\mathbf{w}, \boldsymbol{\beta}') \in \mathbb{Z}_q^n \times \mathbb{Z}_q^{n'-n}$. Then expanding $\hat{\mathbf{b}}_i$ out using (34) gives

$$\left[\mathbf{v}_1^{\xi_i \mathbf{w}} \cdot \mathbf{v}_2^{\xi_i^2 \mathbf{w}} \cdots \mathbf{v}_\ell^{\xi_i^\ell \mathbf{w}}\right] \cdot \left[\mathbf{b}^{\boldsymbol{\beta}_i} \cdot \mathbf{c}^{\boldsymbol{\gamma}_i} \cdot g^{\delta_i}\right] = 1.$$

This is a relation among the group elements in $\mathbf{V}$ and the randomly sampled group elements in $\mathbf{b}, \mathbf{c}, g$. Now, because of the restriction on how $\mathcal{P}_1$ operates, the prover $(\mathcal{P}_1, \mathcal{P}_2)$ along with the IPA extractor can be used to break the AVDL assumption. Hence, this case can happen with at most negligible probability.

Case 2. *Ext* could fail in step 4. This means that two of the extracted witnesses, say numbers i and j, are valid and satisfy

$$(\hat{\mathbf{b}}_i)^{\boldsymbol{\beta}_i} \cdot \mathbf{c}^{\boldsymbol{\gamma}_i} \cdot g^{\delta_i} = \hat{h}_i \qquad\qquad (\hat{\mathbf{b}}_j)^{\boldsymbol{\beta}_j} \cdot \mathbf{c}^{\boldsymbol{\gamma}_j} \cdot g^{\delta_j} = \hat{h}_j$$

and $(\boldsymbol{\beta}_i, \boldsymbol{\gamma}_i, \delta_i) \neq (\boldsymbol{\beta}_j, \boldsymbol{\gamma}_j, \delta_j)$. Write $\boldsymbol{\beta}_i = (\mathbf{w}_i, \boldsymbol{\beta}_i') \in \mathbb{Z}_q^n \times \mathbb{Z}_q^{n'-n}$. Then expanding out $\hat{\mathbf{b}}_i, \hat{\mathbf{b}}_j$ and $\hat{h}_i, \hat{h}_j$ using (34), and dividing one relation by the other gives:

$$\left[\mathbf{v}_1^{\xi_i \mathbf{w}_i - \xi_j \mathbf{w}_j} \cdots \mathbf{v}_\ell^{(\xi_j^\ell \mathbf{w}_i) - (\xi_j^\ell \mathbf{w}_j)}\right] \cdot \left[\mathbf{b}^{\boldsymbol{\beta}_i - \boldsymbol{\beta}_j} \cdot \mathbf{c}^{\boldsymbol{\gamma}_i - \boldsymbol{\gamma}_j} \cdot g^{\delta_i - \delta_j}\right] = u_1^{\xi_i - \xi_j} \cdots u_\ell^{\xi_i^\ell - \xi_j^\ell}$$

This is a non-zero relation among the group elements in $\mathbf{V}$ and $\mathbf{u}'$ along with the randomly sampled group elements in $\mathbf{b}, \mathbf{c}, g$. Then because of the restriction on how $\mathcal{P}_1$ operates, the prover $(\mathcal{P}_1, \mathcal{P}_2)$ along with the IPA extractor can be used to break the AVDL assumption. Hence, this case can happen with at most negligible probability.

Case 3. Finally, we argue that the witness output in step 6 is a valid witness for the $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$. At this point we know that the common extracted witness $(\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ is a valid witness for all $(\ell+1)$ $\mathcal{R}_{\text{IP}}$ instances $(\hat{\mathbf{b}}_i, \mathbf{c}, g, \hat{h}_i)$, namely

$$(\hat{\mathbf{b}}_i)^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^{\delta} = \hat{h}_i \quad \text{for } i \in [\ell+1].$$

Write $\boldsymbol{\beta} = (\mathbf{w}, \boldsymbol{\beta}') \in \mathbb{Z}_q^n \times \mathbb{Z}_q^{n'-n}$. Then expanding one last time using (34) gives

$$\mathbf{v}_1^{\xi_i \mathbf{w}} \cdots \mathbf{v}_\ell^{\xi_i^\ell \mathbf{w}} \cdot \mathbf{b}^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^{\delta} = h \cdot u_1^{\xi_i} \cdots u_\ell^{\xi_i^\ell} \quad \text{for } i \in [\ell+1]. \tag{39}$$

By taking appropriate linear combinations of these $(\ell+1)$ relations, we can separate out the constituent relations and show that (i) $\mathbf{v}_i^{\mathbf{w}} = u_i$ for $i \in [\ell]$ and (ii) $\mathbf{b}^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^{\delta} = h$. For example, there are unique scalars $\lambda_1, \ldots, \lambda_{\ell+1} \in \mathbb{Z}_q$ such that

$$\sum_{i=1}^{\ell+1} \lambda_i \xi_i^\ell = 1 \qquad \text{and} \qquad \sum_{i=1}^{\ell+1} \lambda_i \xi_i^j = 0 \quad \text{for } j = 0, 1, \ldots, \ell-1.$$

By taking the product of all $\ell+1$ equations in (39), where equation i is raised to the power of λ_i for $i = 1, 2, \ldots, \ell+1$, we obtain that $\mathbf{v}_\ell^{\mathbf{w}} = u_\ell$. There are similar scalars for all the other constituent relations.

Since $\mathbf{v}_i^{\mathbf{w}} = u_i$ for all $i \in [\ell]$ we have that $\mathbf{V}^{\mathbf{w}} = \mathbf{u}'$. Moreover, the fact that $\mathbf{b}^{\boldsymbol{\beta}} \cdot \mathbf{c}^{\boldsymbol{\gamma}} \cdot g^{\delta} = h$, and the soundness of the bulletproofs compiler, implies that $\mathbf{a}^{\mathbf{w}} = \mathbf{u}$ and $P(\mathbf{w}) = 1$. Hence $\mathbf{w}$ is a valid witness for the $\mathcal{R}_{\text{eBP}}$ instance $(P, \mathbf{a}, u, \mathbf{V}, \mathbf{u}')$, as required.

An imperfect IPA extractor. It remains to argue how to apply the argument above when the IPA extractor is not perfect. We use a standard approach of repeated trials. We repeat the body of the loop in step (1) of Figure 11 until $\ell+1$ witnesses are extracted. Let $\epsilon_2'(\lambda) := \big(\epsilon_1(\lambda) - \kappa(\lambda)\big)/2q(\lambda)$. We know that with probability at least $\epsilon_2'(\lambda)$ over the choice of $\mathbf{b}, \mathbf{c}, g$, each iteration succeeds in extracting a witness with probability at least $\epsilon_2'(\lambda)$. Therefore, with probability at least $\epsilon_2'(\lambda)$, the expected number of iterations until $\ell+1$ witnesses are extracted is $(\ell+1)/\epsilon_2'(\lambda)$. It follows that the resulting extractor satisfies the bounds in (38). □