Efficient Threshold Zero-Knowledge with Applications to User-Centric Protocols*

Marcel Keller[1], Gert Læssøe Mikkelsen[2], and Andy Rupp[3]

[1] University of Bristol, UK
m.keller@bristol.ac.uk
[2] Alexandra Institute, Denmark
gert.l.mikkelsen@alexandra.dk
[3] AGT Group (R&D) GmbH, Germany
arupp@agtinternational.com

Abstract. In this paper, we investigate on *threshold proofs*, a framework for distributing the prover's side of *interactive proofs of knowledge* over multiple parties. Interactive proofs of knowledge (PoK) are widely used primitives of cryptographic protocols, including important user-centric protocols, such as identification schemes, electronic cash (e-cash), and anonymous credentials.

We present a security model for threshold proofs of knowledge and develop threshold versions of well-known primitives such as range proofs, zero-knowledge proofs for preimages of homomorphisms (which generalizes PoKs of discrete logarithms, representations, p-th roots, etc.), as well as OR statements. These building blocks are proven secure in our model.

Furthermore, we apply the developed primitives and techniques in the context of user-centric protocols. In particular, we construct distributed-user variants of Brands' e-cash system and the bilinear anonymous credential scheme by Camenisch and Lysyanskaya. Distributing the user party in such protocols has several practical advantages: First, the security of a user can be increased by sharing secrets and computations over multiple devices owned by the user. In this way, losing control of a single device does not result in a security breach. Second, this approach also allows *groups of users* to jointly control an application (e.g., a joint e-cash account), not giving a single user full control.

The distributed versions of the protocols we propose in this paper are relatively efficient (when compared to a general MPC approach). In comparison to the original protocols only the prover's (or user's) side is modified while the other side stays untouched. In particular, it is oblivious to the other party whether it interacts with a distributed prover (or user) or one as defined in the original protocol.

Keywords: Multiparty computation, threshold cryptography, distributed provers, Σ-protocols, e-cash, anonymous credentials.

1 Introduction

The general idea of increasing the security of cryptographic primitives by distributing computations is not new. There is a large body of cryptographic literature dealing with threshold digital signatures [1, 22, 23, 26, 33, 44], where the security of a signature scheme is increased by splitting the signer player into several players. In this way one obtains schemes that stay secure even in case one or more (up to a certain threshold) of these players get corrupted.

We believe that in a world where people are expected to participate in more and more different and complex security protocols, there is a need for extending the idea of threshold schemes for users beyond signatures. In particular, the acceptance and deployment of user-centric cryptographic protocols including e-cash [13] and anonymous credentials [14] could profit from strengthening the security guarantees for the human user.

An important work in this vein (which however, still restricts to signatures) is due to Damgård and Mikkelsen [23]. They introduce a model where the human user in the context of a signature scheme is represented not by a single player, as often done in cryptographic literature, but by several players thereby decoupling the user and his actual equipment (e.g., a smart card and a PC). This leads to a more realistic model of the world, and, by using threshold signatures,

* This is a full version of [36], which appeared in the *6th International Conference on Information-Theoretic Security* (ICITS) 2012

it increases the security of the human user against identity theft: If only one of the user's devices gets corrupted, the adversary is still not able to make signatures on behalf of the user.

A major goal of the work at hand is to extend this approach to the more complex case of user-centric cryptographic protocols. Here, for instance, users need to be protected against loss of credentials, electronic coins, etc. due to corruption, theft, or loss of their devices. Another strongly related goal in this context is to protect a *group of users* who jointly own an e-cash account or hold a credential from misuse by a single member of the group.

To achieve the above goals we need to distribute the user's part of these protocols in an appropriate way. From an implementation point of view, it is a very important objective for new protocols to work as seamlessly as possible together with existing solutions. If the other players in the protocol, e.g., the shop accepting electronic cash, are oblivious of the fact that the user is distributed, then a distributed-user variant of the protocol can be utilized in already working environments alongside other implementations, without requiring new standardizations and/or implementations of big and system-wide changes. In practice, we aim for distributing the user's private information between n players, and let these players communicate with some additional player $\mathcal{C}$ combining these messages and acting with respect to the other party (e.g, the shop) as if it was a single user. For the purpose of implementing distributed-user protocols also on computationally weak mobile devices like smart phones or smart cards, efficiency of the protocols is of great importance.

Related Work. Pedersen [41] considers *distributed provers* in the context of undeniable signatures. However, while our goal is to improve the security against theft of the user's identity, Pedersen's focus is on robustness. Desmedt et al. [25] propose a model similar to ours. While their model is based on zero-knowledge proofs of knowledge, our model extends the properties of Σ-protocols. This allows us to easily construct protocols that have arbitrary challenge spaces, which is more difficult to achieve with the general definition of zero-knowledge. In fact, they only present a protocol where the challenge is one bit. Furthermore, our model avoids interaction among the provers, and it preserves the communication pattern of the single-prover protocols. Desmedt [24] introduced the protocol presented in Figure 1. However, he does not give a security proof for his protocol, and he considers neither applications nor more intricate proofs as we do with OR proofs and range proofs.

Further examples of related work are [23] described earlier, and the work of Simoens et al. [47]. The latter work also utilizes threshold cryptography to enhance the security of users. The authors propose threshold signatures and threshold encryption schemes, focusing on very constrained devices. In contrast to our approach they design distributed schemes from scratch where the verifier is aware of the fact that it interacts with a distributed party.

Brands [6] considers a model called *wallet with observers*, where the user player of an e-cash scheme is distributed into two different entities. The objective is to protect the bank against double spenders, and this is achieved by using tamper proof hardware. Our objective is to improve the security of the user, and we do not rely on tamper proof hardware.

Our Contribution. This work proposes a framework for threshold zero-knowledge proofs consisting of a security model as well as techniques and building blocks for constructing threshold user-centric protocols. In particular, we introduce threshold variants of numerous important Σ-protocols for proofs of knowledge which are frequently used as building blocks: This includes Schnorr's protocol [45] for discrete logarithms, Okamoto's [40] protocol for representations, Fiat-Shamir's [30] and Guillou-Quisquater's [35] protocols for modular roots, protocols for proving equalities, a protocol for proving correctness of DH keys, protocols for proving multiplicative relations of commitments, and many further protocols which are, e.g., used in e-cash and credential systems. To do so, we consider a generalization of above protocols by Maurer [39] (PoK of a preimage of a homomorphism) instead of treating each protocol individually.

As a further contribution, we develop a threshold version of the OR construction for the generalized PoK mentioned above. While coming up with a distributed PoK for preimages is not too hard, this is not the case for the OR protocol. Here tricky modifications to the computation flow on the prover's side are required. The OR construction by Cramer et al. [17] is an important building block. For example, it is an efficient way to increase the security of proofs of knowledge: It can be used to obtain a *witness hiding* [28] Σ-protocol starting from one not having this property. Moreover, by means of this construction one can turn a witness hiding Σ-protocol into a protocol providing zero-knowledge against dishonest verifiers. Furthermore, we develop threshold versions of the range proofs by Lipmaa [37] and Boudot [5] also

used in various user-centric protocols. For all of our threshold variants of the above protocols we show zero-knowledge under passive as well as active corruption of the verifier and a number of provers.

As a case study to demonstrate the usefulness of our approach we apply it to obtain several distributed user-centric protocols. More precisely, we sketch distributed-user variants of Brands' e-cash system [6], the pairing-based anonymous credential scheme by Camenisch and Lysyanskaya [10] and an identification scheme by Cramer and Damgård [15]. We note that our primary goal in the context of these applications is to improve the user's security rather than protecting its privacy (untraceability of transactions) in case the adversary gains control over a device. Nevertheless, in the appendix D we sketch how forward and backward untraceability of users could be achieved if a device becomes temporarily corrupted.

One may ask how our threshold protocols differ from a solution using full-fledged multi-party computation. The key difference is communication efficiency: Our protocols are highly efficient, most of them preserve the three move structure with respect to each prover, which is optimal for single-prover zero-knowledge proofs. We achieve this by avoiding full-featured multiparty computation, instead, we exploit the properties of linear and multiplicative secret sharing schemes. In particular, we use so-called pseudorandom secret sharing [16] to allow the provers to generate secret shared random numbers without communication. Moreover, we combine multiplicative secret sharing with pseudorandom zero-sharing to allow the provers to carry out a multiplication of two secret shared numbers without communication. This approach differs from common multiparty computation [3] involving a resharing round after every local multiplication of shares.

Since the focus of this paper is on efficiency, we refrain from considering complex protocols such as the proof of knowledge of a Hamiltonian cycle [4]. Revealing the Hamiltonian cycle in the last step of this protocol involves branching, which is known to be relatively expensive in multiparty computation. For the same reason, we do not consider universally composable zero-knowledge or related notions like non-malleable zero-knowledge [32]. To the best of our knowledge, these imply the use of techniques that do not allow an efficient implementation as multiparty computation, like hash functions in [32] or converting secret values between different domains in [27].

2 Security Model and Preliminaries

Contrary to the traditional model for user-centric protocols, where the user and his computing equipment is modeled as one player, this equipment can be represented as several players in the model we consider in this work. Computations done by a user are split up among several devices (each representing a player), which the user may carry with him. This could be for example a cell phone, smart cards, but a device might also be a server connected to the Internet.

The protocols we are going to consider in this model are *threshold proofs*, a distributed version of protocols for proofs of knowledge (PoK). While the model covers general zero-knowledge proofs of knowledge, this paper mainly considers Σ-protocols. Σ-protocols are a special class of PoK protocols, where the prover $\mathcal{P}$ starts the protocol by sending a message a to the verifier $\mathcal{V}$. $\mathcal{V}$ replies with an l-bit string e. From a, e and its private input $\mathcal{P}$ calculates a response z and sends z to $\mathcal{V}$. From a, e and z, $\mathcal{V}$ is able to verify the proof. Σ-protocols are an important primitive in many cryptographic protocols. For further details we refer to [20].

We study how the prover $\mathcal{P}$ of Σ-protocols can be distributed such that the private input is shared between n provers $\mathcal{P}_1, \ldots, \mathcal{P}_n$, representing the user's computing devices. Each of these provers communicates with a player, denoted combiner $\mathcal{C}$, combining the messages in the proof and communicating with $\mathcal{V}$ as if $\mathcal{C}$ was $\mathcal{P}$ in the original protocol. The protocol view of $\mathcal{V}$ has to be indistinguishable from the single-prover protocol. $\mathcal{C}$ could represent one of the user's devices handling the communication with the verifier, or it could model a device not controlled by the user, e.g., a specially designed terminal for receiving e-cash in a threshold e-cash scheme. Because of this, we specify $\mathcal{C}$ in our protocols such that it does not store any user-specific data.

It should be noted that generally Σ-protocols do not achieve provable zero-knowledge against a malicious verifier, they only achieve what is known as *special honest-verifier zero-knowledge*. This translates in the threshold proofs to a lack of provable zero-knowledge against an actively corrupted combiner. However, as we will see the standard solutions against malicious verifiers also imply zero-knowledge against malicious combiners.

2.1 Security Model

We assume that up to t of the n provers can be corrupted either passively or actively by an adversary, with some additional control over the verifier $\mathcal{V}$ and the combiner $\mathcal{C}$. The degree of control the adversary may have over $\mathcal{V}$ and $\mathcal{C}$ depends on the assumptions made in the original protocols regarding the corruption of $\mathcal{V}$. The security property we focus on is *zero-knowledge*, even in case of some corruptions, and *not correctness*, meaning the adversary is not allowed to gain information, but is allowed to prevent proofs from being accepted. It is, however, possible to construct robust protocols, which would lead to a lower threshold. We only consider static adversaries, although most of the protocols are secure against adaptive adversaries. We extend the properties of Σ-protocols as follows:

Definition 1 (Completeness). *If all provers and the combiner follow the protocol, the verifier accepts with overwhelming probability.*

Definition 2 (Special soundness). *From any two accepting conversations with the same initial message, the witness can be computed in polynomial time.*

Definition 2 is the same as in the single-prover case (e.g., see [20]). Informally speaking, we are not concerned with the question who actually "knows" the witness, e.g., it might be $\mathcal{C}$ interacting with $\mathcal{V}$ while ignoring the messages from the provers, or it might be just one prover.

Definition 3 (Special honest-verifier-combiner zero-knowledge with threshold t). *There exists a zero-knowledge simulator that, for any given challenge e, can simulate a protocol execution that is perfectly, statistically, or computationally indistinguishable from a real protocol execution with passive corruption of the verifier, the combiner, and up to t provers.*

Note that (single-prover) Σ-protocols in general cannot be proven to be zero-knowledge if the challenge is not independent of the initial message, and thus, the verifier and the combiner cannot be actively corrupted in our distributed Σ-protocols. However, in Definition 4 we allow the combiner to deviate partially, i.e., he is allowed to send differing challenges to the provers. This extension of the adversaries capabilities gives a model in between passive corruption and active corruption. This model is not so interesting by itself, however, we define it because a protocol that is *partial zero-knowledge* can by standard techniques be transformed into a protocol with full *zero-knowledge*.

Definition 4 (Partial zero-knowledge with threshold t). *There exists a zero-knowledge simulator that, for any set of challenges $\{e^{(i)}\}_{i\in[n]}$, can simulate a transcript that is perfectly, statistically, or computationally indistinguishable from a real execution with up to t actively corrupted provers and $e^{(i)}$ being sent to $\mathcal{P}_i$ for all $i \in [n]$. To do so, the simulator can interact with the corrupted provers in a black-box way.*

Definition 5 (Full zero-knowledge with threshold t). *Let an active adversary statically corrupt up to t provers, the combiner, and the verifier. Then there exists a zero-knowledge simulator that can simulate a transcript that is perfectly, statistically, or computationally indistinguishable from a real execution by rewindable black-box interaction with the corrupted parties.*

In Appendix G, we show that any partial zero-knowledge protocol with threshold t can be extended to a protocol that implements a UC functionality corresponding to the notion of full zero-knowledge with threshold t. The extension essentially lets the provers commit the their first messages before they receive the challenge. An adversary computing a challenge dependent on a first message would then break the hiding property of the commitment scheme.

Definition 6 (Witness hiding with threshold t). *Let a computationally bounded, active adversary statically corrupt up to t provers, the combiner, and the verifier. Then the probability that the adversary can output the witness after a polynomial number of protocol runs is negligible.*

It is clear that computational full zero-knowledge with threshold t implies witness hiding with threshold t, however, the reverse is not known to hold.

2.2 Secret Sharing

To present our protocols in a general way, we briefly introduce here an abstract definition of linear and multiplicative secret sharing schemes. For more details see Appendix A.

Definition 7 (Linear secret sharing). *A linear secret sharing scheme for a ring consists of two algorithms:*

Share *Takes a secret value and some random values as input and outputs one or more shares for every player, using a linear operation on the inputs.*

Reconstr *Takes all shares given by* Share *to a qualified set of players as input and outputs the secret, using a linear operation on the inputs.*

The shares given to any unqualified set are perfectly or statistically indistinguishable from a sharing of zero.

Since Reconstr is linear, the sum of two share vectors output by Share is a sharing of the sum of the secrets. Therefore, computing a secret sharing of the output of any linear function from a secret sharing of the inputs can be done without communication. Desmedt et al. [25] describe how a linear secret sharing scheme for $\mathbb{Z}_m$ implies a secret sharing scheme for a group G of order m.

The simplest example of a linear secret sharing scheme is additive secret sharing, where the shares are simply random numbers adding up to the secret. It works for any ring, and can also be defined over a group, where the shares are reconstructed by applying the group operation on the shares. The only qualified set is the set of all players.

Definition 8 (Multiplicative secret sharing). *A multiplicative secret sharing scheme is a linear secret sharing scheme that in addition provides the following algorithm:*

Mult *Takes all shares of two secrets, given to a certain player, and outputs a value, using a bilinear operation on the two share vectors. The sum of the outputs of all players is the product of the two secrets.*

One of the first and most common linear secret sharing schemes is Shamir's secret sharing scheme [46] for finite fields. Shamir's secret sharing with threshold $t < n/2$ is multiplicative. Cramer et al. [18] show how to construct a multiplicative integer secret sharing scheme with threshold $t < n/2$.

The notion of multiplicative secret sharing schemes can be extended to cyclic groups $G = \langle g \rangle$, in the sense that the sum and the product of g^s and $g^{s'}$ are defined to be $g^{s+s'}$ and $g^{ss'}$, respectively. Since exponentiation with base g is homomorphic as well as Share and Reconstr are linear, those algorithms trivially can be applied in this setting. However, $g^{ss'}$ is in general not computable from g^s and $g^{s'}$ without knowing s or s'. Nevertheless, if the exponents of the second input vector are known, $\mathsf{Mult}'(g^{s_i}, s_i') := g^{\mathsf{Mult}(s_i, s_i')}$ can be computed because Mult is bilinear. Multiplicative integer secret sharing can be used with any cyclic group, whereas Shamir secret sharing over $\mathbb{Z}_{|G|}$ can be used if G has prime order.

Pseudorandom Secret Sharing (PRSS) and Zero Sharing (PRZS). Pseudorandom secret sharing was introduced by Cramer et al. [16]. It is a way for a number of players to generate a secret shared value without communication. PRSS is practical as long as the number of players is relatively small (as we can assume for our applications). We will refer to the sharing of a random number as pseudorandom secret sharing (PRSS) and to the additive sharing of 0 as pseudorandom zero sharing (PRZS). Furthermore, we denote by PRZS' an alternative version of the latter, which outputs a secret sharing of the neutral element 1 of a group. Appendix A provides details on the constructions used in this paper. For the sake of readability of the protocol descriptions, we omit that all the mentioned schemes are given a set of keys when used. We also omit that a variable changing from invocation to invocation (like a counter) is used internally so that, by every invocation of $\mathsf{PRSS}(i)$ or $\mathsf{PRZS}(i)$, $\mathcal{P}_i$ obtains its share of a new random number or its new share of 0, respectively.

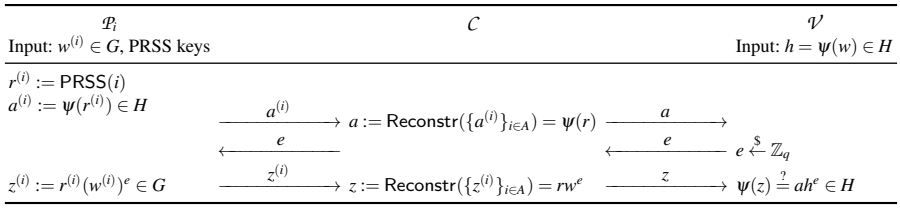

Fig. 1. Distributed proof of knowledge of a preimage $w \in G$ of an element $h \in H$ under a group homomorphism $\psi : G \to H$

3 Threshold Building Blocks

3.1 Proofs of Knowledge of Preimages of Homomorphisms

Let G and H be two finite Abelian groups both written multiplicatively and let $\psi : G \to H$ be a homomorphism. Furthermore, let $h \in H$ and q be a prime such that $u \in G$ with $\psi(u) = h^q$ is known. If H has prime order q (and thus is cyclic), this condition is fulfilled for $u = 1$ because $\psi(1) = 1 = h^q$. A proof of knowledge of a preimage $w \in G$ of h under ψ works as follows (cf. Figure 9 in the appendix): The prover starts by sending $a := \psi(r)$ for a random $r \in G$, the verifier sends a random challenge $e \in \mathbb{Z}_q$, the prover responds with $z := rw^e$, and the verifier checks whether $\psi(z) = ah^e$. This protocol is due to Maurer [39] and generalizes proofs of knowledge of discrete logarithms, p-th roots, and many more. Using u of the form mentioned above, Maurer proves special soundness.

Desmedt [24] formulated a distributed version of this protocol (see Figure 1) using a linear secret sharing of the group elements r and w. Each prover executes the computations as in the original protocol with the only difference that it uses its shares $r^{(i)}$ and $w^{(i)}$ instead of r and w. The combiner $\mathcal{C}$ collects the messages $a^{(i)}$ and $z^{(i)}$ from the provers and computes a and z by reconstructing as soon as there are enough shares available. A denotes the qualified set of provers that are the first to send a share. If the linear scheme allows it, the provers are not required to be completely present.

Lemma 1. *The protocol in Figure 1 achieves completeness, special soundness, and computational partial zero-knowledge with the same threshold as the utilized secret sharing scheme.*

Proof. Completeness follows from the protocol description, and special soundness is proven in the same way as for the single-prover protocol. The partial zero-knowledge simulator implicitly runs the simulator of the original protocol, adapting to deviating provers and a partially deviating combiner. See Appendix B.1.

3.2 OR Construction for Proofs of Knowledge of Preimages

The OR construction is an important basic building block for Σ-protocols. It allows a prover to show that it knows a witness for one out of two given inputs *but without* revealing to which input the witness corresponds. The OR construction is itself a Σ-protocol. We consider the OR construction for proofs of preimages as described in the previous section. Figure 2 shows a protocol proving the knowledge of a ψ-preimage of one out of two given elements while hiding for which of the elements a preimage is known, i.e., the bit b is hidden.

To obtain a distributed version of this protocol where a prover $\mathcal{P}_i$ is not aware of the secret preimage w_b *and* the bit b, the first step is to make the computations on the prover's side uniform. That means the way one computes a_b, e_b, and z_b should not differ from the way to compute $a_{\bar{b}}$, $e_{\bar{b}}$, and $z_{\bar{b}}$, where $\bar{b} = 1 - b$. Figure 10 in the appendix shows such an alternative version of the protocol but still for the case of a single prover. Secret sharing of the secrets of the prover finally leads to a distributed version of the protocol.

Figure 3 shows our threshold protocol for groups of prime order q. We use Shamir's secret sharing over $\mathbb{Z}_q$ with threshold $t < n/2$ and the abstract notion of multiplicative secret sharing introduced in Section 2.2. $b^{(i)}$, $\bar{b}^{(i)}$, and $1^{(i)}$ denote a secret sharing of b, $\bar{b}$, and 1 over $\mathbb{Z}_q$, where $b \in \{0,1\}$ and $\bar{b} := 1 - b$. $1^{(i)}$ is only needed to convert shares into an additive secret sharing using Mult and can be computed using fixed randomness. Thus, $1^{(i)}$ can be seen as defined by the secret sharing scheme. $v_0^{(i)}$ and $v_1^{(i)}$ denote a secret sharing of v_0 and v_1 over G, where $v_b := w_b$ and $v_{\bar{b}} := 1$. We

$\mathcal{P}$		$\mathcal{V}$
Input: $b \in \{0,1\}, w_b \in G$		Input: $h_0 = \psi(w_0), h_1 = \psi(w_1) \in H$
$r_b, z_{\bar{b}} \xleftarrow{\$} G, e_{\bar{b}} \xleftarrow{\$} \mathbb{Z}_q$		
$a_b := \psi(r_b) \in H$		
$a_{\bar{b}} := \psi(z_{\bar{b}}) h_{\bar{b}}^{-e_{\bar{b}}} \in H$	$\xrightarrow{a_0, a_1}$	
	$\xleftarrow{e}$	$e \xleftarrow{\$} \mathbb{Z}_q$
$e_b := e - e_{\bar{b}} \in \mathbb{Z}_q$		
$z_b := r_b w_b^{e_b} \in G$	$\xrightarrow{e_0, e_1, z_0, z_1}$	$e \stackrel{?}{=} e_0 + e_1 \in \mathbb{Z}_q$
		$\psi(z_0) \stackrel{?}{=} a_0 h_0^{e_0} \in H$
		$\psi(z_1) \stackrel{?}{=} a_1 h_1^{e_1} \in H$

Fig. 2. Proof of knowledge of a ψ-preimage of h_b, given $h_0, h_1 \in H$

$\mathcal{P}_i$		$\mathcal{C}$		$\mathcal{V}$
Input: $b^{(i)}, \bar{b}^{(i)}, 1^{(i)} \in \mathbb{Z}_q$, PRSS and PRZS keys, $v_0^{(i)}, v_1^{(i)} \in G$, $h_0, h_1 \in H$				Input: $h_0, h_1 \in H$
$r_0^{(i)}, r_1^{(i)} \xleftarrow{\$} G$				
$d_0^{(i)} := \mathsf{PRSS}(i)$				
$d_1^{(i)} := \mathsf{PRSS}(i)$				
$a_0^{(i)} := \psi(r_0^{(i)}) \cdot h_0^{-\mathsf{Mult}(b^{(i)}, d_0^{(i)})} \cdot \mathsf{PRZS}'(i)$				
$a_1^{(i)} := \psi(r_1^{(i)}) \cdot h_1^{-\mathsf{Mult}(\bar{b}^{(i)}, d_1^{(i)})} \cdot \mathsf{PRZS}'(i)$	$\xrightarrow{a_0^{(i)}, a_1^{(i)}}$	$a_0 := \prod_i a_0^{(i)}$		
		$a_1 := \prod_i a_1^{(i)}$	$\xrightarrow{a_0, a_1}$	
$e_0^{(i)} := \mathsf{Mult}(b^{(i)}, d_0^{(i)}) + \mathsf{Mult}(\bar{b}^{(i)}, 1^{(i)}) e$	$\xleftarrow{e}$		$\xleftarrow{e}$	$e \xleftarrow{\$} \mathbb{Z}_q$
$\quad - \mathsf{Mult}(\bar{b}^{(i)}, d_1^{(i)}) + \mathsf{PRZS}(i) \in \mathbb{Z}_q$				
$e_1^{(i)} := \mathsf{Mult}(b^{(i)}, 1^{(i)}) e - \mathsf{Mult}(b^{(i)}, d_0^{(i)})$				
$\quad + \mathsf{Mult}(\bar{b}^{(i)}, d_1^{(i)}) + \mathsf{PRZS}(i) \in \mathbb{Z}_q$				
$z_0^{(i)} := r_0^{(i)} \cdot \mathsf{Mult}'(v_0^{(i)}, 1^{(i)})^e / \mathsf{Mult}'(v_0^{(i)}, d_1^{(i)}) \cdot \mathsf{PRZS}'(i)$				
$z_1^{(i)} := r_1^{(i)} \cdot \mathsf{Mult}'(v_1^{(i)}, 1^{(i)})^e / \mathsf{Mult}'(v_1^{(i)}, d_0^{(i)}) \cdot \mathsf{PRZS}'(i)$	$\xrightarrow{e_0^{(i)}, e_1^{(i)}, z_0^{(i)}, z_1^{(i)}}$	$e_0 := \sum_i e_0^{(i)}$		
		$e_1 := \sum_i e_1^{(i)}$		
		$z_0 := \prod_i z_0^{(i)}$		
		$z_1 := \prod_i z_1^{(i)}$	$\xrightarrow{e_0, e_1, z_0, z_1}$	$e \stackrel{?}{=} e_0 + e_1 \in \mathbb{Z}_q$
				$\psi(z_0) \stackrel{?}{=} a_0 h_0^{e_0}$
				$\psi(z_1) \stackrel{?}{=} a_1 h_1^{e_1}$

Fig. 3. Distributed version of the proof of knowledge of a ψ-preimage of h_b, given $h_0, h_1 \in H$. Assumption: G is is of prime order q.

share these over G because the discrete logarithm of w_b might not be known in the setup phase. $r_0^{(i)}$ and $r_1^{(i)}$ can be understood as an additive secret sharing of some random group elements r_0 and r_1.

PRSS refers to pseudorandom secret sharing for the multiplicative secret sharing scheme used, and PRZS and PRZS$'$ refer to pseudorandom zero sharing for additive secret sharing. The reason for using PRZS is that the outputs of Mult and Mult$'$ could reveal information about the inputs, i.e., the outputs do not form a random sharing of the product of the secrets. Therefore, we use PRZS to get a truly (pseudo)random additive secret sharing before sending outputs of Mult or Mult$'$ to $\mathcal{C}$.

In the following, we show that the view of $\mathcal{V}$ is the same as in the protocol described in Figure 2. e_0, e_1 are random numbers that sum up to e, and z_0 and z_1 pass the check by the verifier:

$$e_0+e_1=\sum_i e_0^{(i)}+e_1^{(i)}=(bd_0+\bar{b}e-\bar{b}d_1)+(be-bd_0+\bar{b}d_1)=\begin{cases}(e-d_1)+(d_1) & b=0\\ (d_0)+(e-d_0) & b=1,\end{cases}$$

$$z_0=\prod_i z_0^{(i)}=\prod_i r_0^{(i)}\cdot \mathsf{Mult}'(v_0^{(i)},1^{(i)})^e/\mathsf{Mult}'(v_0^{(i)},d_1^{(i)})\cdot \mathsf{PRZS}'(i)=r_0v_0^e/v_0^{d_1}=r_0v_0^{e-d_1}$$

$$=\begin{cases}r_0v_0^{bd_0+\bar{b}(e-d_1)}=r_0v_0^{e_0} & b=0\\ r_0v_{\bar{b}}^{e-d_1}=r_01^{e-d_1}=r_01^{e_0}=r_0v_0^{e_0} & b=1\end{cases}$$

The same can be proven for z_1.

Applying standard multiparty computation secure against a passive adversary [3] would require one communication round among the provers per round of multiplication operations. Multiparty computation secure against an active adversary usually requires even more communication. This is what we save by directly publishing the output of Mult and Mult', re-randomized by PRZS and PRZS', respectively.

Lemma 2. *The protocol for prime order groups as described above achieves completeness, special soundness, and computational partial zero-knowledge with threshold $t<n/2$.*

Proof. Completeness follows from the protocol description, and special soundness is proven in the same way as for the single-prover protocol. The partial zero-knowledge simulator implicitly runs the simulator of the original protocol, adapting to deviating provers and a partially deviating combiner. The essence of the proof is a smart combination of multiparty and zero-knowledge simulation techniques. See Appendix B.2.

Lemma 3. *The protocol described above is witness hiding with threshold t if the corresponding single-prover OR proof is witness hiding.*

Proof. The proof is done by reduction, i.e., we present a simulator that simulates the set of honest provers by accessing the honest prover of the single-prover protocol internally. See Appendix B.3.

3.3 Range Proofs

In this section, we will outline how to adapt the range proof by Lipmaa [37] to our distributed setting. All protocols in this section use the Fujisaki-Okamoto commitment scheme [21, 31] $\mathsf{com}(x,r):=g^xh^r$ for $g,h\in\mathbb{Z}_N^*$ and $r\in_R[0,2^sN]$ (s is a security parameter). Clearly, com is a homomorphic function from $\mathbb{Z}\times\mathbb{Z}$ to $\mathbb{Z}_N^*$. Therefore, one might want to construct a proof of knowledge of a preimage as in Section 3.1. However, $\mathbb{Z}\times\mathbb{Z}$ is not finite, and thus, the technique does not apply directly. Nevertheless, Damgård and Fujisaki [21] showed that a similar protocol achieves soundness if the strong RSA assumption holds for $\mathbb{Z}_N^*$. The protocol can efficiently be distributed like the protocol in Section 3.1 by using an additive secret sharing of x and r over the integers.

An essential building block of the range proof is the proof that a committed value is a square. However, $\mathsf{com}(x^2,r)$ is not a homomorphic function in x and r, and therefore, the approach of the protocol in Section 3.1 cannot be used. The following construction solves the problem: $\mathcal{P}$ chooses a random r_1, and computes $r_2:=r-r_1x$ and $C:=\mathsf{com}(x,r_1)=g^xh^{r_1}$. Then, $\mathsf{com}(x^2,r)=g^{x^2}h^r=g^{x^2}h^{xr_1+r_2}=C^xh^{r_2}$. The function $(x,r_1,r_2)\mapsto(g^xh^{r_1},C^xh^{r_2})$ is a homomorphism; thus, a zero-knowledge proof can be constructed similarly to the proof of knowledge of a committed value if the strong RSA assumption holds for $\mathbb{Z}_N^*$. See Figure 6 in Appendix E for details. Damgård and Fujisaki [21] prove that a more general version of the protocol (proof that a commitment hides a product of two other committed values) achieves computational soundness and statistical honest-verifier zero-knowledge.

Since the protocol involves products of secret values, linear secret sharing does not suffice to get a distributed protocol. However, one can use the same techniques as for the OR proof, namely multiplicative secret sharing, PRSS, and PRZS. Again, this allows to minimize the communication. See Appendix E for the detailed protocol.

Now we describe how to prove that a committed number x lies in a certain range $[a,b]$. Clearly, proving $x\in[a,b]$ is equivalent to showing that $x-a$ and $b-x$ are non-negative. Furthermore, Lagrange's four-square theorem states that

every non-negative integer can be written as the sum of four squares of integers. Together with the proof of a square, this directly allows to implement a range proof. First, the prover computes $\alpha_1,\ldots,\alpha_4,\beta_1,\ldots,\beta_4$ such that $\sum_{i=1}^4 \alpha_i^2 = x-a$ and $\sum_{i=1}^4 \beta_i^2 = b-x$, and the commitments $\mathsf{com}(\alpha_i^2, r_{1i})$, $\mathsf{com}(\beta_i^2, r_{2i})$ for all $i \in [1,4]$ such that $\sum_{i=1}^4 r_{1i} = r$ and $\sum_{i=1}^4 r_{2i} = -r$. Then he sends the commitments to the verifier and uses the proof of square described above to prove that all these commitments hide a square. Finally, the verifier checks that $\mathsf{com}(x,r)/g^a = \sum_{i=1}^4 \mathsf{com}(\alpha_i^2, r_{1i})$ and that $g^b/\mathsf{com}(x,r) = \sum_{i=1}^4 \mathsf{com}(\alpha_i^2, r_{2i})$. Lipmaa [37] proposes an optimized proof based on the same ideas.

The proof can be adapted to our distributed setting if the provers hold a multiplicative secret sharing of $\alpha_1,\ldots,\alpha_4,\beta_1,\ldots,\beta_4$. The distributed generation of these shares using the algorithm in [37] requires very expensive MPC due to branching, but they can be generated in a setup phase because they only depend on the committed value and the range boundaries. Boudot's range proof described in Appendix F only requires the computation of integer square roots at that point, but has other drawbacks like not being exact.

4 Applications to User-Centric Protocols

4.1 E-Cash with Threshold Wallets

In this section we describe a threshold version of the e-cash scheme by Brands [6]. In the standard version, a user $\mathcal{U}$ opens an account in a bank $\mathcal{B}$. The user can then withdraw electronic money and later use it in a shop $\mathcal{S}$. $\mathcal{S}$ may then deposit the electronic money to his own bank account. Moreover, even if $\mathcal{B}$ and $\mathcal{S}$ cooperate they cannot link electronic coins to a specific user. In the threshold version the user $\mathcal{U}$ is split up into several entities $\mathcal{U}_1$ to $\mathcal{U}_n$ and a combiner $\mathcal{C}$ acting as the user with respect to $\mathcal{B}$ and $\mathcal{S}$. The global setup of the system and the protocol for $\mathcal{S}$ depositing the e-cash in $\mathcal{B}$ does not include the user and are therefore not changed.

Opening an account. The public key of $\mathcal{B}$ are three generators g, g_1, and $g_2 \in G_q$, where G_q is a cyclic group of prime order q. The private key of $\mathcal{B}$ is a random number $x \in \mathbb{Z}_q^*$. When $\mathcal{U}$ wants to open an e-cash account at $\mathcal{B}$, he samples $u_1 \xleftarrow{\$} \mathbb{Z}_q$ and uses u_1 as secret key. From u_1, $\mathcal{U}$ calculates $I := g_1^{u_1}$, sends I to $\mathcal{B}$ and proves knowledge of u_1 such that $I = g_1^{u_1}$. If the proof is accepted, $\mathcal{B}$ calculates $z := (Ig_2)^x$ and sends z to $\mathcal{U}$. The bank stores I as the account number of $\mathcal{U}$, and $\mathcal{U}$ stores u_1 and z. This protocol can easily be extended to a threshold protocol, where u_1 is generated distributedly as an additive secret shared value. From the shares of u_1, the players $\mathcal{U}_i$ and $\mathcal{C}$ can calculate I distributedly. The proof of knowledge of u_1 is distributed as in the protocol from Section 3.1 (cf. Figure 1).

Withdrawing coins from the bank. This protocol can be viewed as a Σ-protocol, where $\mathcal{B}$ acts as prover, proving knowledge of its own secret key to $\mathcal{U}$. As a result $\mathcal{U}$ ends up with a blinded signature $sig(A,B)$ from $\mathcal{B}$ on a coin (A,B). This coin along with the signature can later be used for payment. Distributing a verifier might be straightforward, however, in this case the signature contains a hash value $\mathsf{H}(\cdot)$, and most hash functions are far from linear. They can therefore not be distributed easily. Nevertheless, we can still protect the user from theft of coins in case some $\mathcal{U}_i$ is corrupted, although this might jeopardize the anonymity of $\mathcal{U}$. The threshold protocol is sketched in Figure 4. Some random values used for anonymizing the protocol are generated by a pseudorandom function (PRF). Distributing the PRF key allows generating commonly known pseudorandom values without reconstruction. The value B has, due to hashing, to be known by each $\mathcal{U}_i$, however, B is computed from x_1 and x_2, and not by the PRF. Therefore, it has to be reconstructed by $\mathcal{C}$. We blind B by ρ_B to maintain anonymity.

If the adversary does not control any of $\mathcal{U}_1,\ldots,\mathcal{U}_n$ nor the combiner $\mathcal{C}$, then the protocol is equivalent to the original protocol, and therefore has the same level of security in that case. In case of corruption, we do not ensure anonymity, nevertheless, no information regarding neither u_1, x_1 nor x_2 is leaked, and therefore, even after withdrawal with corrupted players, the adversary still has only negligible chance of paying afterward.

Spending coins. In this part of the original protocol $\mathcal{U}$ sends the coin $\langle A,B,sig(A,B)\rangle$ to $\mathcal{S}$. $\mathcal{U}$ proves that the coin is valid, obtained by the user with secret key u_1, and that $\mathcal{U}$ has knowledge of u_1. This is basically a Σ-protocol proving representation of A with respect to g_1 and g_2, using x_1 and x_2 for blinding. Therefore, we can distribute it by the protocol from Section 3.1 (cf. Figure 1). Please note that B and not $B^{(i)}$ should be sent from each $\mathcal{U}_i$ to $\mathcal{C}$. Otherwise, ρ_B would be leaked to $\mathcal{C}$, and anonymity is revoked.

Theorem 1 summarizes the properties of our threshold version of Brands' e-cash scheme.

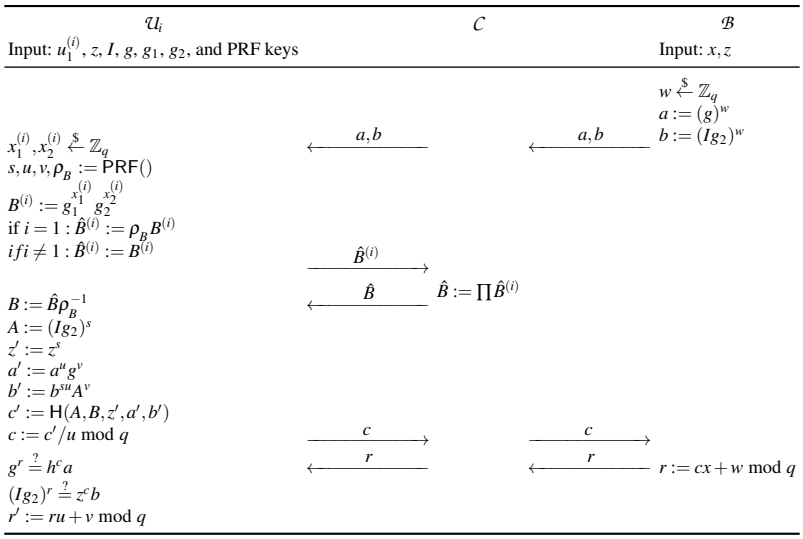

Fig. 4. Distributed version of the withdrawal protocol

Theorem 1. *The threshold scheme preserves the following properties from the original scheme [6, Section 5]*

- ***Anonymity of*** $\mathcal{U}$***:*** *The anonymity of the original scheme [6, Corollary 12] is preserved against a passive adversary controlling* $\mathcal{C}$*,* $\mathcal{B}$ *and* $\mathcal{S}$*.*
- ***Security against double spenders, and forgery:*** *Security for* $\mathcal{B}$ *and* $\mathcal{S}$ *against double spenders [6, Proposition 10] and against forged coins [6, Corollary 9, Proposition 7 and 13] is preserved.*
- ***Security against theft:*** *Security for* $\mathcal{U}$ *against theft of electronic coins [6, Proposition 14 and 15] is preserved against an adversary passively corrupting* $n-1$ *of the user players, the combiner* $\mathcal{C}$*,* $\mathcal{B}$ *and* $\mathcal{S}$*.*

Proof. Security for $\mathcal{B}$ and $\mathcal{S}$ follows since it is oblivious for them that they execute the threshold version. Since u_1 is secret shared, the adversary cannot steal coins, and anonymity of $\mathcal{U}$ can be reduced to anonymity in the original scheme. For more details see Appendix B.4.

Robustness. To allow payments even in case some of the user players $\mathcal{U}_i$ are not present in the payment protocol, the following changes suffice. First the protocol for opening an account has to generate the users secret key u_1 as being shared with a linear secret sharing scheme that has lower threshold than the additive one. This can be done by using PRSS. The same has to be done with the blinding values x_1 and x_2 in the withdrawal protocol. In addition, $\mathcal{C}$ has to reconstruct the value of $\hat{B}$ taking into account that it is computed from the shares of x_1 and x_2, which is possible because the reconstruction of x_1 and x_2 is a linear operation. This enables a threshold version of the payment protocol where only a subset of the players $\mathcal{U}_i$ equivalent to the threshold value of the secret sharing scheme needs to be present.

4.2 Increasing the Security of Anonymous Credentials

Credentials are certificates of qualification (e.g., driver's licenses) or authorization of some kind (e.g., e-tickets) that are attached to a specific user. Credential systems allow their users ($\mathcal{U}$) to obtain credentials from organizations (O) and show possession of these credentials to verifiers ($\mathcal{V}$). In an anonymous credential system different transactions involving the same user cannot be linked.

Camenisch and Lysyanskaya [9,38] show that an anonymous credential scheme can be immediately composed from a commitment scheme, a signature scheme, and efficient protocols for (1) proving the equality of two committed values, (2) obtaining a signature on a committed value (without opening the commitment), and (3) proving knowledge of a signature on a committed value. In [10] the same authors propose efficient instantiations of the above building blocks

over bilinear group settings. In the following we shortly describe these components and sketch threshold versions of the zero-knowledge protocols.

Let G and G_T be two cyclic groups of prime order q (which are both written multiplicatively in the following). A pairing $e : G \times G \to G_T$ is a function with the following properties:

- *Bilinearity*: $\forall (a,b) \in G^2$ and $(x,y) \in \mathbb{Z}_q^2$, it holds that $e(a^x, b^y) = e(a,b)^{xy}$.
- *Non-degeneracy*: $\exists a, b \in G$ such that $e(a,b) \neq 1$.
- e is efficiently computable.

The credential system in [10] makes use of Pedersen commitments [42] over G, which are information-theoretically hiding and computationally binding under the DL assumption. Here, given two generators $g, h \in G$, a commitment M to $m \in \mathbb{Z}_q$ is computed by choosing $r \xleftarrow{\$} \mathbb{Z}_q$ and setting $M := g^m h^r$. In the scope of the credential system, a commitment to a secret m chosen by a user serves as a pseudonym for the user when interacting with an organization. For different organizations, different pseudonyms are used. Usually, all pseudonyms of a user are required to be commitments to the *same* m, which is some master secret key (used outside the credential system).

Credentials are signatures on pseudonyms. To this end, the following signature scheme is proposed in [10], which is secure under the LRSW assumption.

- Gen samples a random generator g of G, $x, y, z \xleftarrow{\$} \mathbb{Z}_q$, and computes $X = g^x, Y = g^y, Z = g^z, W = Y^z$. It returns the signer's secret key $sk = (x,y,z)$ and public key $pk = (g, X, Y, Z, W)$.
- $\mathsf{Sign}(sk, (m,r))$ chooses $\alpha \xleftarrow{\$} \mathbb{Z}_q$ and computes $a = g^\alpha$, $A = a^z$, $b = a^y$, $B = A^y$, $c = a^{x+xym} A^{xyr}$, where $(m,r) \in \mathbb{Z}_q^2$ is the given message. It returns the signature $\sigma = (a, A, b, B, c)$.
- $\mathsf{Verify}(pk, (m,r), \sigma)$ returns 'accept', if the following equations are satisfied: $e(a,Z) = e(g,A)$, $e(a,Y) = e(g,b)$, $e(A,Y) = e(g,B)$, and $e(X,a)e(X,b)^m e(X,B)^r = e(g,c)$.

Let us now consider the protocols described above forming a credential system. For our threshold versions of these protocols we always distribute the user's side of the protocols over parties $\mathcal{U}_1, \ldots, \mathcal{U}_n$.

Proving the equality of two committed values. This is a zero-knowledge proof of knowledge of values (a_1, a_2, a_3) such that $C = g^{a_1} h^{a_2}$ and $C' = g^{a_1} h^{a_3}$, for given C and C'. Since (a_1, a_2, a_3) can be viewed as a preimage of (C, C') under the homomorphism $\psi : \mathbb{Z}_q^3 \to G^2$ where $\psi(\alpha_1, \alpha_2, \alpha_3) = (g^{\alpha_1} h^{\alpha_2}, g^{\alpha_1} h^{\alpha_3})$, Figure 1 immediately yields a threshold version of this protocol for parties $\mathcal{U}_1, \ldots, \mathcal{U}_n$ acting as provers.

Obtaining a signature on a committed value (obtaining a credential). This is a protocol between $\mathcal{U}$ and $\mathcal{O}$. The party $\mathcal{U}$ holds (m,r) and is given $\mathcal{O}$'s public signature key pk as input. $\mathcal{O}$ holds its secret signature key sk and is given commitment $M = g^m Z^r$ as input. The protocol consists of two steps. First, $\mathcal{U}$ proves in zero-knowledge its knowledge of a representation of M with respect to g and Z. Again, a threshold version of this subprotocol can be immediately obtained from the protocol in Figure 1. Second, if $\mathcal{O}$ accepts the previous proof, it generates a signature σ on (m,r) using the signature scheme described above. Since (m,r) is not given, the component c of the signature is computed only using M as $c = a^x M^{\alpha xy}$. Then, $\mathcal{U}$ checks the validity of σ according to Verify. Note that only for checking the last equation the knowledge of (m,r) is required. So to distribute the user's side of this subprotocol over $\mathcal{U}_1, \ldots, \mathcal{U}_n$, we only need to compute the left-hand side of this equation in a distributed way: $\mathcal{U}_i$ computes $w^{(i)} := e(X,a)e(X,b)^{m^{(i)}} e(X,B)^{r^{(i)}} e(g,g)^{\mathsf{PRZS}(i)}$, where $m^{(i)}$, $r^{(i)}$ are additive shares of m and r, respectively. Then it sends $w^{(i)}$ to the combiner $\mathcal{C}$ who computes the product of these shares and checks the corresponding equation. It is easy to see that the threshold protocol for obtaining a signature on a committed value is zero-knowledge with respect to adversaries passively corrupting $\mathcal{O}$, $\mathcal{C}$, and a static set of $n-1$ parties $\mathcal{U}_i$.

Proving knowledge of a signature on a committed value (showing a credential). This is a protocol executed between a user $\mathcal{U}$ and a verifier $\mathcal{V}$. $\mathcal{U}$ is given (m,r), a signature σ on these values, and the public key pk of the signer. $\mathcal{V}$ is only given pk. The player $\mathcal{U}$ first re-randomizes the given signature. To this end, it chooses $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$, computes $\tilde{a} = a^{r_1}$, $\tilde{A} = A^{r_1}$, $\tilde{b} = b^{r_1}$, $\tilde{B} = B^{r_1}$, $\hat{c} = c^{r_1 r_2}$, and sends $\tilde{\sigma} = (\tilde{a}, \tilde{A}, \tilde{b}, \tilde{B}, \hat{c})$ to $\mathcal{V}$. $\mathcal{V}$ is now able to verify all equations for $\tilde{\sigma}$ defined in Verify except for the last which involves (m,r). To assure $\mathcal{V}$ also of the validity of the last equation, $\mathcal{U}$

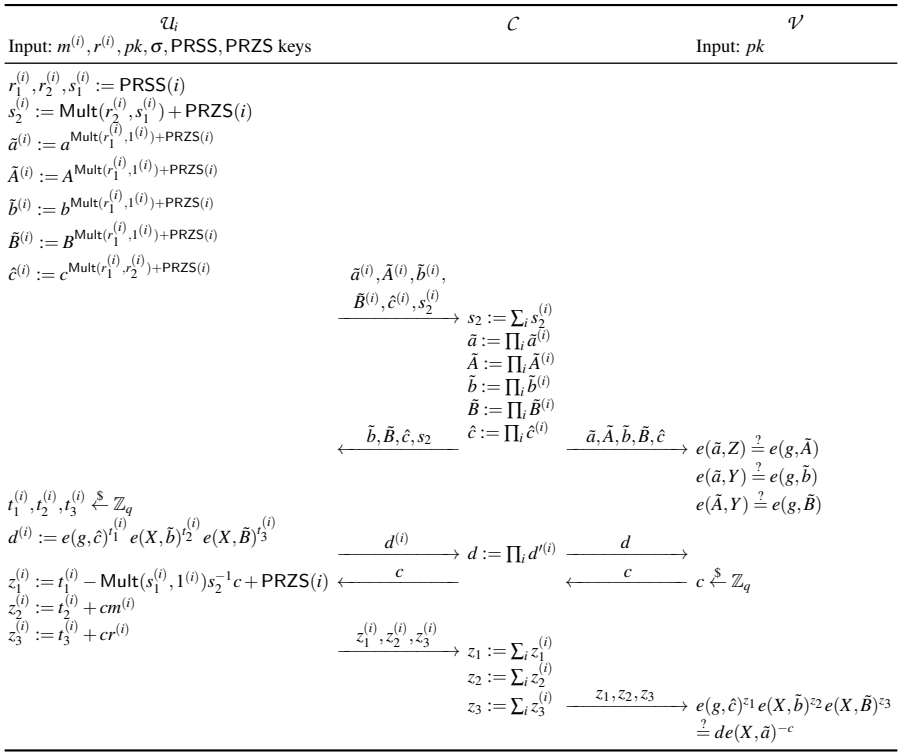

Fig. 5. Distributed version of the protocol for proving knowledge of a signature on a committed value

proves in zero-knowledge the knowledge of (r_2^{-1}, m, r) such that $e(X,\tilde{a})e(X,\tilde{b})^m e(X,\tilde{B})^r = e(g,\hat{c})^{r_2^{-1}}$. Note that this is equivalent to proving knowledge of a representation of $e(X,\tilde{a})^{-1}$ with respect to $e(X,\tilde{b})$, $e(X,\tilde{B})$, and $e(g,\hat{c})$. The threshold version of the whole protocol is shown in Figure 5. Note that in order to compute $\hat{c}$ a multiplicative secret sharing (e.g., Shamir's secret sharing) of r_1 and r_2 is needed. Furthermore, in order to compute an additive sharing of r_2^{-1} from a multiplicative sharing of r_2 (without reconstructing r_2), another trick is used: We choose a random value s_1 by means of PRSS and compute $s_2 := r_2 s_1$ in a distributed manner. Note that s_2 which is reconstructed by the combiner does not leak information about r_2. Then s_2 is given to every $\mathcal{U}_i$ who uses this value to compute its additive share $\mathsf{Mult}(s_1^{(i)}, 1^{(i)}) s_2^{-1}$ of r_2^{-1}.

It is easy to see that the protocol in Figure 5 achieves completeness and special soundness. It is also zero-knowledge in the case of adversaries passively corrupting $\mathcal{V}$ and $\mathcal{C}$. However, as soon as a single $\mathcal{U}_i$ is passively corrupted, the adversary knows σ and thus it is not a zero-knowledge proof of the signature anymore. But this does only affect the anonymity of a user, which we do not aim to increase here. More precisely, even if σ is known, the protocol does not leak information about (m,r) as long as not more than $n-1$ of the players $\mathcal{U}_i$ are passively corrupted. So an adversary is not able to steal a user's identity or a credential in this case. Theorem 2 (informally) summarizes the properties which can be achieved using the proposed threshold protocols.

Theorem 2. *The threshold versions of the protocols described above can be used to build a credential system with the following main properties:*

- *Anonymity for users in presence of a passive adversary controlling $\mathcal{C}$, $\mathcal{O}$ and $\mathcal{V}$.*
- *Security of a user's master secret key against a static adversary passively corrupting $n-1$ of the user players $\mathcal{U}_i$, the combiner $\mathcal{C}$, $\mathcal{O}$ and $\mathcal{V}$.*
- *Security for $\mathcal{O}$ and $\mathcal{V}$ against forgery of credentials.*

4.3 Identification

Cramer and Damgård [15] show that OR proofs can be used to construct an identification scheme resilient against man-in-the-middle attacks. Both parties are assumed to have elements in the domain of a homomorphic one-way function as private keys, whereas the public keys are the value of the private keys under the function. The identification protocol is then simply a proof of knowledge of one of the private keys of the two parties. The construction in Section 3.2 directly allows to distribute the private key among a set of provers. On the other hand, having a set of verifiers requires a further assumption. If the challenge is chosen as the sum of challenge shares chosen by the verifiers, a cheating verifier in an asynchronous network could adapt his challenge to the other challenges such that the sum would be a challenge desired by the prover. This problem can be solved be either letting the verifiers commit to their challenge share in advance, or by computing the challenge as the output of a hash function, as in the Fiat-Shamir heuristic [30]. The latter is only secure in the random oracle model.

References

1. Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 593–611. Springer (2006)
2. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell [8], pp. 390–420
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC. pp. 1–10. ACM (1988)
4. Blum, M.: How to prove a theorem so no one else can claim it. In: Gleason, A.M. (ed.) Proceedings of the International Congress of Mathematicians. pp. 1444–1451 (1986)
5. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel [43], pp. 431–444
6. Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In: Stinson, D.R. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 773, pp. 302–318. Springer (1993)
7. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)
8. Brickell, E.F. (ed.): Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings, Lecture Notes in Computer Science, vol. 740. Springer (1993)
9. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN. Lecture Notes in Computer Science, vol. 2576, pp. 268–289. Springer (2002)
10. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M.K. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3152, pp. 56–72. Springer (2004)
11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. pp. 136–145 (2001)
12. Chan, A.H., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. In: EUROCRYPT. pp. 561–575 (1998)
13. Chaum, D.: Blind signatures for untraceable payments. In: CRYPTO. pp. 199–203 (1982)
14. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM 28(10), 1030–1044 (1985)
15. Cramer, R., Damgård, I.: Fast and secure immunization against adaptive man-in-the-middle impersonation. In: EUROCRYPT. pp. 75–87 (1997)
16. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC. Lecture Notes in Computer Science, vol. 3378, pp. 342–362. Springer (2005)
17. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994)
18. Cramer, R., Fehr, S., Ishai, Y., Kushilevitz, E.: Efficient multi-party computation over rings. In: Biham, E. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 2656, pp. 596–613. Springer (2003)
19. Damgård, I.: Efficient concurrent zero-knowledge in the auxiliary string model. In: Preneel [43], pp. 418–430
20. Damgård, I.: On Σ-protocols, Course Notes. Aarhus University (2010)
21. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 2501, pp. 125–142. Springer (2002)
22. Damgård, I., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 2045, pp. 152–165. Springer (2001)
23. Damgård, I., Mikkelsen, G.L.: On the theory and practice of personal digital signatures. In: Jarecki, S., Tsudik, G. (eds.) Public Key Cryptography. Lecture Notes in Computer Science, vol. 5443, pp. 277–296. Springer (2009)
24. Desmedt, Y.: Threshold cryptosystems (invited talk). In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT. Lecture Notes in Computer Science, vol. 718, pp. 3–14. Springer (1992)

25. Desmedt, Y., Crescenzo, G.D., Burmester, M.: Multiplicative non-abelian sharing schemes and their application to threshold cryptography. In: Pieprzyk, J., Safavi-Naini, R. (eds.) ASIACRYPT. Lecture Notes in Computer Science, vol. 917, pp. 21–32. Springer (1994)
26. Desmedt, Y., Frankel, Y.: Shared generation of authenticators and signatures (extended abstract). In: Feigenbaum [29], pp. 457–469
27. Dodis, Y., Shoup, V., Walfish, S.: Efficient constructions of composable commitments and zero-knowledge proofs. In: Wagner, D. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 5157, pp. 515–535. Springer (2008)
28. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: STOC. pp. 416–426. ACM (1990)
29. Feigenbaum, J. (ed.): Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings, Lecture Notes in Computer Science, vol. 576. Springer (1992)
30. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
31. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Jr., B.S.K. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 1294, pp. 16–30. Springer (1997)
32. Garay, J.A., MacKenzie, P.D., Yang, K.: Strengthening zero-knowledge protocols using signatures. J. Cryptology 19(2), 169–209 (2006)
33. Gennaro, R., Rabin, T., Jarecki, S., Krawczyk, H.: Robust and efficient sharing of RSA functions. J. Cryptology 13(2), 273–300 (2000)
34. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC. pp. 291–304. ACM (1985)
35. Guillou, L., Quisquater, J.J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: EUROCRYPT. pp. 123–128 (1988)
36. Keller, M., Mikkelsen, G.L., Rupp, A.: Efficient threshold zero-knowledge with applications to user-centric protocols. In: ICITS 2012. Lecture Notes in Computer Science, Springer (2012)
37. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Laih, C.S. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 2894, pp. 398–415. Springer (2003)
38. Lysyanskaya, A.: Signature Schemes and Applications to Cryptographic Protocol Design. Ph.D. thesis, Massachusetts Institute of Technology (2002)
39. Maurer, U.M.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT. Lecture Notes in Computer Science, vol. 5580, pp. 272–286. Springer (2009)
40. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell [8], pp. 31–53
41. Pedersen, T.P.: Distributed provers with applications to undeniable signatures. In: EUROCRYPT. pp. 221–242 (1991)
42. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum [29], pp. 129–140
43. Preneel, B. (ed.): Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding, Lecture Notes in Computer Science, vol. 1807. Springer (2000)
44. Rabin, T.: A simplified approach to threshold and proactive RSA. In: Krawczyk, H. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 1462, pp. 89–104. Springer (1998)
45. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer (1989)
46. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
47. Simoens, K., Peeters, R., Preneel, B.: Increased resilience in threshold cryptography: Sharing a secret with devices that cannot store shares. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing. Lecture Notes in Computer Science, vol. 6487, pp. 116–135. Springer (2010)

A Linear Secret Sharing

Definition 9. *A set $\Gamma \subset 2^{[n]}$ is called a monotone access structure on $[n]$ if for all $A \in \Gamma$ and $A' \supset A$, A' is also in Γ.*

The sets in Γ are called qualified, and sets in Γ^c are called unqualified. A set $A \in \Gamma^c$ is called maximal unqualified if, from $A' \supset A$ with $A' \in \Gamma^c$, it follows that $A' = A$.

Definition 10. *Let R be a ring and Γ a monotone access structure on $[n]$. A linear secret sharing scheme on R is defined by a matrix $M \in R^{d \times e}$, where every row of M is assigned to a player. We denote by M_A the rows assigned to the players in $A \subset [n]$. Then, the following holds for every $A \subset [n]$:*

- *If $A \in \Gamma$, there is a so-called reconstruction vector $\lambda_A \in R^d$ such that $M_A^T \lambda_A = (1,0,\ldots,0)^T$.*
- *If $A \notin \Gamma$, there is a so-called sweeping vector $\kappa_A \in R^e$ with first entry 1 such that $M_A \kappa_A = 0$.*

Secret sharing of s is done by choosing random values $r_2,\ldots,r_e$, computing $M \cdot (s, r_2, \ldots, r_e)^T$, and distributing the entries of the resulting vector according to the row assignment of M. The reconstruction vectors ensure that every qualified set of players can reconstruct the secret, and the sweeping vectors ensure that every unqualified set gets only negligible information about the secret if the random values were chosen appropriately. That is, from the entire field for finite fields, and from an interval exponentially larger than any entry in any κ_A for $\mathbb{Z}$.

We now describe how a linear secret sharing scheme for $\mathbb{Z}_m$ implies a secret sharing scheme for a group G of order m. Let g be an element of G and s_i be the shares of a secret $s \in \mathbb{Z}_m$. Then, g^{s_i} is a secret sharing of g^s. For a qualified set A, let $\{\lambda_i\}_{i\in A}$ be the reconstruction parameters. Reconstruction then works as follows:

$$\prod (g^{s_i})^{\lambda_i} = g^{\sum s_i \lambda_i \bmod m} = g^s$$

because $g^m = 1$, i.e., the modulo reduction in the exponent leaves the group element unchanged. It is easy to see that $g^{s_i} g^{s_i'} = g^{s_i + s_i'}$ is a share of $g^{s+s'}$ if s_i and s_i' are shares of s and s', respectively, and that s does not have to be known to secret share g^s because Share is linear. Similarly, a linear secret sharing scheme for $\mathbb{Z}$ implies a secret sharing scheme for any cyclic group. It is straightforward to extend the construction to secret sharing schemes with several shares per player.

A.1 Examples

Shamir's Secret Sharing. Shamir's secret sharing works for finite fields and the ring $\mathbb{Z}_m$ if the smallest prime factor of m is greater than the number of players. The shares are points on a random polynomial, which evaluates to the secret in 0. Reconstruction works using Lagrange interpolation, and thus, the qualified sets are those strictly bigger than the degree of the polynomial.

Shamir's secret sharing fulfills the above definition with $d = n$, $e = t+1$, M being the Vandermonde matrix of the points assigned to the players, and Γ consisting of all sets in $[n]$ with size at least $t+1$ (threshold t). The reconstruction vectors are defined by the Lagrange interpolation; i.e., the entry of λ_A for player $i \in A$ is $\prod_{j\in A\setminus\{i\}} \frac{x_j}{x_j - x_i}$. The sweeping vector κ_A is defined by the coefficients α_i of a polynomial $p(x) = \sum_{i\le t} \alpha_i x^i$ evaluating to 0 in x_i for all $i \in A$ and to 1 in 0. Such a polynomial exists for $|A| \le t$. Furthermore, $\alpha_0 = 1$ because of the latter constraint.

It is easy to see that Shamir's secret sharing with threshold $t < n/2$ achieves the properties of multiplicative secret sharing scheme, with Share and $\mathsf{Reconstr}$ being the normal sharing and reconstruction procedures. Let Mult be defined as follows: $\mathsf{Mult}(s_i, s_i') := a_i s_i s_i'$, where a_i is defined as the Lagrange interpolation coefficient for a polynomial of degree $2t$. Lagrange interpolation is possible because $2t < n$.

Multiplicative Integer Secret Sharing. Cramer et al. [18] show how to construct a multiplicative integer secret sharing scheme with threshold $t < n/2$. Roughly speaking, they construct a Shamir-like secret sharing scheme for an extension ring of $\mathbb{Z}$, and they show that this scheme induces a scheme for $\mathbb{Z}$ with the same properties. $\mathbb{Z}$ does not allow the direct construction of a Shamir's secret sharing scheme because $\{1,-1\}$ are the only invertible elements of $\mathbb{Z}$, and Shamir's secret sharing requires $x_i - x_j$ to be invertible for all $i \ne j \in [n]$.

A.2 Pseudorandom Secret Sharing

The goal of PRSS is to get a secret sharing of a pseudorandom number without communication. PRSS be can implemented for every linear secret sharing scheme as defined above. For every maximal unqualified set, let r_A be a (pseudo)random number known only to A^c. $r_A M \kappa_A$ is a secret sharing of r_A, with all shares of A being zero. Thus, all players can compute their shares of r_A without communication. The parties then can compute a secret sharing of $r := \sum_A r_A$ by summing up the shares of all r_A's because the scheme is linear. r is not known to any unqualified set because it lacks at least one summand. Usually, r_A is computed as output of a pseudorandom function with a secret key known to all players in A^c and a common public input.

A.3 Pseudorandom Zero Sharing

Here the goal is to get a pseudorandom secret sharing of zero, with the actual sharing being hidden from every unqualified set. In some of our protocols, we need pseudorandom zero sharing with threshold $t < n/2$ for additive secret sharing. The threshold for additive secret sharing is $n-1$, however, we use PRZS to hide the output of multiplicative secret sharing, which has threshold $t < n/2$. Additive secret sharing can be defined by the $n \times n$-matrix

$$M = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & \ddots & \ddots & & \\ & & -1 & 1 & \\ 1 & & & -1 \end{pmatrix}$$

and the reconstruction vector $\lambda_{[n]} = (1,\ldots,1)^T$. The sweeping vectors are $\kappa_A = (1,0,\ldots,0,1,\ldots,1)$, where there are $i-1$ zeros for some $i \notin A$.

Pseudorandom zero sharing with threshold $t < n/2$ can be done as follows: For every maximal unqualified player set A, the players in A^c choose a (pseudo)random vector $\mathbf{r}_A$ of length $n-t-1$, multiply it with the $n-t \times n-t-1$-matrix

$$\begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \\ & & & -1 \end{pmatrix},$$

and define their shares as the entries of the result. Clearly, the shares sum up to zero, and hence, the sum of all sharings is a sharing of zero. Any unqualified set does not know at least one $\mathbf{r}_A$ and thus cannot derive the shares of A^c if $\mathbf{r}_A$ was chosen appropriately. That is, from the entire field for finite fields and from a large enough interval for $\mathbb{Z}$. As with PRSS, $\mathbf{r}_A$ usually is computed as output of a pseudorandom function with a secret key known to all players in A^c and a common public input.

The above construction directly extends to any group. Let $\mathbf{r}_A = (r_1,\ldots,r_{n-t-1})^T$ be chosen as vector of random elements in a group. Then, let the matrix operate canonically on $\mathbf{r}_A$, i.e., with the resulting vector being $(r_1, r_1^{-1}r_2, \ldots, r_{n-t-1}^{-1})$. It is easy to see that the product of all elements of the output vector is 1, and therefore an additive secret sharing of 1. As above, $\mathbf{r}_A$ can be computed using a pseudorandom function. We denote this construction by PRZS'.

B Proofs

B.1 Proof of Lemma 1

Completeness follows from the homomorphic property of ψ and the linearity of $\mathsf{Reconstr}$:

$$\begin{aligned} \psi(z) &= \psi\big(\mathsf{Reconstr}\big(\{z^{(i)}\}_{i\in A}\big)\big) \\ &= \psi\Big(\mathsf{Reconstr}\Big(\big\{r^{(i)}\cdot\big(w^{(i)}\big)^e\big\}_{i\in A}\Big)\Big) \\ &= \mathsf{Reconstr}\Big(\big\{\psi\big(r^{(i)}\big)\big\}_{i\in A}\Big)\cdot\mathsf{Reconstr}\Big(\psi\big(w^{(i)}\big)^e\big\}_{i\in A}\Big) \\ &= \mathsf{Reconstr}\Big(\big\{a^{(i)}\big\}_{i\in A}\Big)\cdot\psi(w)^e = ah^e. \end{aligned}$$

Special soundness follows directly from the original protocol. To motivate the partial zero-knowledge simulator, we first show that the protocol achieves special honest-verifier-combiner zero-knowledge with the same threshold t as the utilized secret sharing scheme. The simulator works as follows:

1. Run the simulator of the original protocol on h and e to get a and z.
2. Compute a secret sharing of z: $z^{(i)} := \mathsf{Share}(z)$.
3. For the corrupted provers, choose $w^{(i)}$ uniformly at random and compute $a^{(i)} := \psi(z^{(i)}/(w^{(i)})^e)$.
4. For the honest provers, choose random $a^{(i)}$ such that $\mathsf{Reconstr}(\{a^{(i)}\}_{i\in 1,\dots,n}) = a$. This is possible by the definition of linear secret sharing because the set of corrupted provers is not qualified in the secret sharing scheme.

Unlike the simulation, a real execution uses pseudo-random values for $a^{(i)}$ and $z^{(i)}$. Apart from that, the algorithm simulates the protocol with up to $n-1$ passively corrupted provers because the view of $\mathcal{V}$ is the computationally indistinguishable to the original protocol, $a^{(i)}$ is a random sharing of a, and $z^{(i)}$ is a random sharing of z consistent with $a^{(i)}$ and $w^{(i)}$ for the corrupted provers. Furthermore, the $w^{(i)}$ given to the (at most $n-1$) corrupted provers are random values independent of w. Therefore, the view of the corrupted provers is also computationally indistinguishable.

The partial zero-knowledge simulator for a set J of actively corrupted provers with $|J| < t$ works as follows:

1. Choose $w^{(i)}$ uniformly at random for all $i \in J$.
2. Run the protocol with the corrupted provers to get $a^{(i)}$ and $z^{(i)}$ for all $i \in J$. The only input to the provers is $w^{(i)}$ and $e^{(i)}$, which are fixed.
3. Choose $\{\tilde{h}^{(i)}\}_{i\notin J}$ randomly in G such that $\mathsf{Reconstr}(\{\tilde{h}^{(i)}\}_{i\notin J} \cup \{\psi(w^{(i)})\}_{i\in J}) = h$. This is possible by the definition of secret sharing. It follows that $\prod_{i\in A\setminus J}(\tilde{h}^{(i)})^{\lambda_A^{(i)}} = h/\prod_{i\in J\cap A}\psi(w^{(i)})^{\lambda_A^{(i)}}$ for all reconstruction vectors $\lambda_A = \{\lambda_A\}_{i\in A}$ because the reconstruction is unique for a correct sharing.
4. For the honest provers, choose $z^{(i)} \in G$ at random, and then let $a^{(i)} := \psi(z^{(i)})(\tilde{h}^{(i)})^{-e^{(i)}}$.

Let A be the set of players used by $\mathcal{C}$ for reconstruction, and let $\lambda_A = \{\lambda_A^{(i)}\}_{i\in A}$ the reconstruction vector for A. The simulated transcript is accepting for some challenge e if $e^{(i)} = e$ for all i and if the corrupted provers stick to the protocol:

$$\begin{aligned}
\psi\big(\mathsf{Reconstr}(\{z^{(i)}\}_{i\in A})\big) &= \psi\Big(\prod_{i\in A}(z^{(i)})^{\lambda_A^{(i)}}\Big)\\
&= \prod_{i\in J\cap A}\psi\big((z^{(i)})^{\lambda_A^{(i)}}\big)\cdot\prod_{i\in A\setminus J}\psi\big((z^{(i)})^{\lambda_A^{(i)}}\big)\\
&= \prod_{i\in J\cap A}(a^{(i)})^{\lambda_A^{(i)}}\psi\big((w^{(i)})^{\lambda_A^{(i)}}\big)^e\cdot\prod_{i\in A\setminus J}(a^{(i)})^{\lambda_A^{(i)}}\big((\tilde{h}^{(i)})^{\lambda_A^{(i)}}\big)^e\\
&= \prod_{i\in A}(a^{(i)})^{\lambda_A^{(i)}}\cdot\Big(\prod_{i\in J\cap A}\psi\big((w^{(i)})^{\lambda_A^{(i)}}\big)\cdot\prod_{i\in A\setminus J}(\tilde{h}^{(i)})^{\lambda_A^{(i)}}\Big)^e\\
&= \prod_{i\in A}(a^{(i)})^{\lambda_A^{(i)}}\cdot\Big(\prod_{i\in J\cap A}\psi\big(w^{(i)}\big)^{\lambda_A^{(i)}}\cdot h/\prod_{i\in J\cap A}\psi\big(w^{(i)}\big)^{\lambda_A^{(i)}}\Big)^e\\
&= \prod_{i\in A}(a^{(i)})^{\lambda_A^{(i)}}\cdot h^e.
\end{aligned}$$

Furthermore, the distribution is computationally indistinguishable from the one of a real execution because the corrupted provers are involved (the random $w^{(i)}$ reflect that they are independent from w from the point of view of the adversary) and the communication with honest provers is simulated similarly to the 1-prover setting, assuming $\psi^{-1}(\tilde{h}^{(i)})$ as a share of the witness. However, the latter is not used in the simulation. The assumed shares of the honest provers together with the generated shares of the corrupted provers reconstruct to a preimage of h. Let $\lambda = \lambda_1,\dots,\lambda_n$ be the reconstruction vector for the set of all players. Then,

$$\begin{aligned}
\mathsf{Reconstr}\big(\{w^{(i)}\}_{i\in J\cap A}\cup\{\psi^{-1}(\tilde{h}^{(i)})\}\big) &= \prod_{i\in J}(w^{(i)})^{\lambda^{(i)}}\cdot\prod_{i\notin J}\psi^{-1}\big(\tilde{h}^{(i)}\big)^{\lambda^{(i)}}\\
&= \prod_{i\in J}(w^{(i)})^{\lambda^{(i)}}\cdot\psi^{-1}\Big(\prod_{i\notin J}(\tilde{h}^{(i)})^{\lambda^{(i)}}\Big)\\
&= \prod_{i\in J}(w^{(i)})^{\lambda^{(i)}}\cdot\psi^{-1}\Big(h/\prod_{i\in J}\psi\big(w^{(i)}\big)^{\lambda^{(i)}}\Big) = \psi^{-1}(h).
\end{aligned}$$

The view of the corrupted provers is computationally indistinguishable by the same argument as above.

B.2 Proof of Lemma 2

Completeness follows directly from the protocol description, and special soundness from the same property of the original protocol. The partial zero-knowledge simulator for a set J of corrupted provers with $|J| < n/2$ works as follows:

1. For the corrupted provers, generate the necessary PRSS and PRZS keys, $v_0^{(i)}$, $v_1^{(i)}$, $b^{(i)}$, and $\bar{b}^{(i)}$ as sharings of 0, and $1^{(i)}$ as a sharing of 1.
2. Run the protocol with the corrupted provers, giving the values from the previous step and later $e^{(i)}$ as input. This gives $a_0^{(i)}$, $a_1^{(i)}$, $e_0^{(i)}$, $e_1^{(i)}$, $z_0^{(i)}$, and $z_1^{(i)}$ for $i \in J$.
3. For $i \in J$, compute $\tilde{a}_0^{(i)}$, $\tilde{a}_1^{(i)}$, $\tilde{e}_0^{(i)}$, $\tilde{e}_1^{(i)}$, $\tilde{z}_0^{(i)}$, and $\tilde{z}_1^{(i)}$ as an honest prover would, using the keys and sharings generated previously.
4. For the honest provers, choose $e_0^{(i)}$, $z_0^{(i)}$, and $z_1^{(i)}$ uniformly at random. Furthermore, choose $\{B^{(i)} \in \mathbb{Z}_q\}_{i \notin J}$ uniformly at random such that
$$\sum_{i \notin J} B^{(i)} + \sum_{i \in J} \mathsf{Mult}\big(b^{(i)} + \bar{b}^{(i)}, 1^{(i)}\big) = 1$$
and such that $\{B^{(i)}\}_{i \notin J}$ and $\{\mathsf{Mult}(b^{(i)} + \bar{b}^{(i)}, 1^{(i)})\}_{i \in J}$ form a possible output distribution of $\mathsf{Mult}(\cdot, 1^{(i)})$. This is possible because J is not a qualified set, and thus, $\{b^{(i)} + \bar{b}^{(i)}\}_{i \in J}$ can be completed with shares such that a reconstruction would give any value by Definition 10. Finally, choose $\tilde{e}_1^{(i)}$ at random such that
$$\sum_{i \notin J} \tilde{e}_1^{(i)} = \sum_{i \in J} \Big(\mathsf{Mult}\big(b^{(i)} + \bar{b}^{(i)}, 1^{(i)}\big) e^{(i)} - \tilde{e}_0^{(i)} - \tilde{e}_1^{(i)}\Big) - \sum_{i \notin J} e_0^{(i)},$$
and let $e_1^{(i)} := \tilde{e}_1^{(i)} + B^{(i)} e^{(i)}$.
5. For the honest provers, choose $\tilde{h}_0^{(i)}$ at random such that
$$\prod_{i \notin J} \tilde{h}_0^{(i)} \cdot \prod_{i \in J} \Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big), 1^{(i)}\big) \cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)}, 1^{(i)})}\Big) = 1 \in H$$
and such that $\{\tilde{h}_0^{(i)}\}_{i \notin J}$ and $\{\mathsf{Mult}'(\psi(v_0^{(i)}), 1^{(i)}) \cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)}, 1^{(i)})}\}_{i \in J}$ form a possible output distribution of $\mathsf{Mult}'(\cdot, 1^{(i)})$. Again, this is possible by Definition 10. Furthermore, choose $\tilde{a}_0^{(i)}$ at random such that
$$\prod_{i \notin J} \tilde{a}_0^{(i)} \cdot \prod_{i \in J} \tilde{a}_0^{(i)} = \prod_{i \notin J} \psi\big(z_0^{(i)}\big) \cdot \prod_{i \in J} \psi\big(\tilde{z}_0^{(i)}\big) \cdot h_0^{-(\sum_{i \notin J} e_0^{(i)} + \sum_{i \in J} \tilde{e}_0^{(i)})} \cdot \prod_{i \in J} \Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big), 1^{(i)}\big) \cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)}, 1^{(i)})}\Big)^{-e^{(i)}},$$
and let $a_0^{(i)} := \tilde{a}_0^{(i)} (\tilde{h}_0^{(i)})^{-e^{(i)}}$.
6. Repeat the previous step analogously to get $a_1^{(i)}$ for $i \notin J$.

As in Lemma 1, the generated transcript is accepting if for some e, $e^{(i)} = e$ for all i, and if the corrupted provers stick to the protocol. On one hand,
$$\begin{aligned}
\sum_i e_0^{(i)} + \sum_i e_1^{(i)} &= \sum_{i \in J} \Big(\tilde{e}_0^{(i)} + \tilde{e}_1^{(i)}\Big) + \sum_{i \notin J} \Big(e_0^{(i)} + \tilde{e}_1^{(i)} + B^{(i)} e\Big) \\
&= \sum_{i \in J} \Big(\tilde{e}_0^{(i)} + \tilde{e}_1^{(i)}\Big) + \sum_{i \in J} \Big(\mathsf{Mult}\big(b^{(i)} + \bar{b}^{(i)}, 1^{(i)}\big) e - \tilde{e}_0^{(i)} - \tilde{e}_1^{(i)}\Big) + \sum_{i \notin J} B^{(i)} e \\
&= \left(\sum_{i \in J} \mathsf{Mult}\big(b^{(i)} + \bar{b}^{(i)}, 1^{(i)}\big) + \sum_{i \notin J} B^{(i)}\right) e \\
&= e
\end{aligned}$$

and on the other,

$$\begin{aligned}
\psi\Big(\prod_i z_0^{(i)}\Big) &= \prod_{i\in J}\psi\big(z_0^{(i)}\big)\cdot\prod_{i\notin J}\psi\big(\tilde{z}_0^{(i)}\big)\\
&= \prod_{i\notin J}\tilde{a}_0^{(i)}\cdot\prod_{i\in J}\tilde{a}_0^{(i)}\cdot h_0^{\left(\sum_{i\notin J}e_0^{(i)}+\sum_{i\in J}\tilde{e}_0^{(i)}\right)}\cdot\prod_{i\in J}\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)^{e}\\
&= \prod_{i\notin J}a_0^{(i)}\big(\tilde{h}_0^{(i)}\big)^{e}\cdot\prod_{i\in J}a_0^{(i)}\cdot h_0^{\left(\sum_{i\notin J}e_0^{(i)}+\sum_{i\in J}e_0^{(i)}\right)}\cdot\prod_{i\in J}\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)^{e}\\
&= \prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}\cdot\left(\prod_{i\notin J}\tilde{h}_0^{(i)}\cdot\prod_{i\in J}\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)\right)^{e}\\
&= \prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}
\end{aligned}$$

because $\tilde{e}_0^{(i)} = e_0^{(i)}$, $\tilde{a}_0^{(i)} = a^{(i)}$, and $\tilde{z}_0^{(i)} = z_0^{(i)}$ for all $i \in J$. Analogously, one can prove that $\psi(\prod_i z_1^{(i)}) = \prod_i a_1^{(i)}\cdot h_1^{\sum_i e_0^{(i)}}$.

Note that the sharings given to the corrupted parties have the same distribution as in a real execution because they are independent of the shared values. Furthermore, all messages of the honest parties are generated independently of the actual behavior of the corrupted parties, and uniformly at random under some constraints of the combined values. This is similar to a real execution, where this holds due to the use of PRZS. However, because PRZS uses a pseudorandom function, we achieve only computational security. To attain the accepting constraint, the same trick as in the single-prover zero-knowledge simulator is utilized: e_0 and e_1 are generated randomly under the constraint $e_0 + e_1 = e$, z_0 and z_1 are generated randomly, and a_0 and a_1 are then computed to satisfy the constraints $\psi(z_0) = a_0 h_0^{e_0}$ and $\psi(z_1) = a_1 h_1^{e_1}$.

Finally, we show that the distribution of the simulation is computationally indistinguishable to the distribution of a real execution if differing $e^{(i)}$ are sent by the combiner. Because of the use of PRZS, every single value sent a prover is just a pseudorandom value that sums or multiplies up to a certain value, and even a_0, a_1, e_0, e_1, z_0, and z_1 are just random values that satisfy the same constraints as in the single-prover protocol. Therefore, one only needs to check whether $\sum_i e_0^{(i)} + \sum_i e_1^{(i)}$, $\psi(\prod_i z_0^{(i)})/(\prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}})$, and $\psi(\prod_i z_1^{(i)})/(\prod_i a_1^{(i)}\cdot h_0^{\sum_i e_1^{(i)}})$ are distributed computationally indistinguishable in the simulation and a real execution. We only do this for non-deviating provers because their deviation is simulated correctly in any case, as argued above. In the simulation,

$$\begin{aligned}
\sum_i e_0^{(i)} + \sum_i e_1^{(i)} &= \sum_{i\in J}\Big(\tilde{e}_0^{(i)} + \tilde{e}_1^{(i)}\Big) + \sum_{i\notin J}\Big(e_0^{(i)} + \tilde{e}_1^{(i)} + B^{(i)}e^{(i)}\Big)\\
&= \sum_{i\in J}\Big(\tilde{e}_0^{(i)} + \tilde{e}_1^{(i)}\Big) + \sum_{i\in J}\Big(\mathsf{Mult}\big(b^{(i)} + \bar{b}^{(i)}, 1^{(i)}\big)e_0^{(i)} - e_1^{(i)}\Big) + \sum_{i\notin J}B^{(i)}e\\
&= \sum_{i\in J}\mathsf{Mult}\big(b^{(i)} + \bar{b}^{(i)}, 1^{(i)}\big)e^{(i)} + \sum_{i\notin J}B^{(i)}e^{(i)}
\end{aligned}$$

and

$$\begin{aligned}
\psi\Big(\prod_i z_0^{(i)}\Big)/\Big(\prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}\Big) &= \prod_{i\in J}\psi\big(z_0^{(i)}\big)\cdot\prod_{i\notin J}\psi\big(\tilde{z}_0^{(i)}\big)\cdot\Big(\prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}\Big)^{-1}\\
&= \prod_{i\notin J}\tilde{a}_0^{(i)}\cdot\prod_{i\in J}\tilde{a}_0^{(i)}\cdot h_0^{(\sum_{i\notin J}e_0^{(i)}+\sum_{i\in J}\tilde{e}_0^{(i)})}\cdot\prod_{i\in J}\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)^{e^{(i)}}\\
&\quad\cdot\Big(\prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}\Big)^{-1}\\
&= \prod_{i\notin J}a_0^{(i)}\big(\tilde{h}_0^{(i)}\big)^{e^{(i)}}\cdot\prod_{i\in J}a_0^{(i)}\cdot h^{(\sum_{i\notin J}e_0^{(i)}+\sum_{i\in J}e_0^{(i)})}\cdot\prod_{i\in J}\Big(\mathsf{Mult}'(\psi(v_0^{(i)}),1^{(i)})\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)^{e^{(i)}}\\
&\quad\cdot\Big(\prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}\Big)^{-1}\\
&= \prod_{i\notin J}(\tilde{h}_0^{(i)})^{e^{(i)}}\cdot\prod_{i\in J}\Big(\mathsf{Mult}'(\psi(v_0^{(i)}),1^{(i)})\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)^{e^{(i)}}.
\end{aligned}$$

In a real execution,

$$\begin{aligned}
\sum_i e_0^{(i)}+\sum_i e_1^{(i)} &= \sum_i\Big(\mathsf{Mult}\big(b^{(i)},d_0^{(i)}\big)+\mathsf{Mult}\big(\bar{b}^{(i)},1^{(i)}\big)e^{(i)}-\mathsf{Mult}\big(\bar{b}^{(i)},d_1^{(i)}\big)\\
&\qquad+\mathsf{Mult}\big(b^{(i)},1^{(i)}\big)e^{(i)}-\mathsf{Mult}\big(b^{(i)},d_0^{(i)}\big)+\mathsf{Mult}\big(\bar{b}^{(i)},d_1^{(i)}\big)\Big)\\
&= \sum_i\Big(\mathsf{Mult}\big(\bar{b}^{(i)},1^{(i)}\big)+\mathsf{Mult}\big(b^{(i)},1^{(i)}\big)\Big)e^{(i)}
\end{aligned}$$

and

$$\begin{aligned}
\psi\Big(\prod_i z_0^{(i)}\Big)/\Big(\prod_i a_0^{(i)}\cdot h_0^{\sum_i e_0^{(i)}}\Big) &= \psi\Big(\prod_i\Big(r_0^{(i)}\cdot\mathsf{Mult}'\big(v_0^{(i)},1^{(i)}\big)^{e^{(i)}}\cdot\mathsf{Mult}'\big(v_0^{(i)},d_1^{(i)}\big)^{-1}\Big)\Big)\\
&\quad\cdot\Big(\prod_i\Big(\psi\big(r_0^{(i)}\big)\cdot h_0^{-\mathsf{Mult}(b^{(i)},d_0^{(i)})}\Big)\cdot h_0^{\sum_i\mathsf{Mult}(b^{(i)},d_0^{(i)})+\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})e^{(i)}-\mathsf{Mult}(\bar{b}^{(i)},d_1^{(i)})}\Big)^{-1}\\
&= \prod_i\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)^{e^{(i)}}\cdot\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),d_1^{(i)}\big)^{-1}\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})e^{(i)}+\mathsf{Mult}(\bar{b}^{(i)},d_1^{(i)})}\Big)\\
&= \prod_i\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)^{e^{(i)}}\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})e^{(i)}}\Big)\cdot\psi(v_0)^{-d_1}\cdot h_0^{\bar{b}d_1}\\
&= \prod_i\Big(\mathsf{Mult}'\big(\psi\big(v_0^{(i)}\big),1^{(i)}\big)\cdot h_0^{-\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}\Big)^{e^{(i)}}
\end{aligned}$$

because $\psi(v_0)=h_0^{\bar{b}}$. For the same reason, and since $b+\bar{b}=1$, $B^{(i)}$ and $\tilde{h}_0^{(i)}$ are by construction distributed like $\mathsf{Mult}(b^{(i)}+\bar{b}^{(i)},1^{(i)})$ and $\mathsf{Mult}'(\psi(v_0^{(i)}),1^{(i)})^{-1}\cdot h_0^{\mathsf{Mult}(\bar{b}^{(i)},1^{(i)})}$, respectively, in a real execution. It follows that the distributions are indistinguishable.

B.3 Proof of Lemma 3

We will present a simulator that acts as an interface between the honest prover of the single-prover protocol and the adversary. Clearly, any witness for the distributed protocol is also a witness for the single-prover protocol. Therefore, the success probability of the adversary in the distributed protocol is the same as the success probability of the combination of the simulator and the adversary, hence, the success probability of the adversary is at most the success probability of the most successful adversary for the single-prover protocol, which is negligible by assumption.

The simulator works as follows for a set J of corrupted provers. As a setup, generate the necessary PRSS and PRZS keys, $v_0^{(i)}$, $v_1^{(i)}$, $b^{(i)}$, and $\bar{b}^{(i)}$ as sharings of 0, and $1^{(i)}$ as a sharing of 1, all for the corrupted provers. This is possible without knowing the witness or b because the shares of the corrupted provers are independent from both by the properties of secret sharing. For every protocol run requested by the adversary, do the following:

1. Start a protocol run with the honest prover of the single-prover protocol and receive a_0, a_1.
2. For the corrupted provers, compute $\{a_0^{(i)}, a_1^{(i)}\}_{i\in J}$ as if they were honest, and choose random $\{a_0^{(i)}, a_1^{(i)}\}_{i\notin J}$ for the honest provers such that $\prod_i a_0^{(i)} = a_0$ and $\prod_i a_1^{(i)} = a_1$.
3. Send $\{a_0^{(i)}, a_1^{(i)}\}_{i\in[n]}$ to the adversary, and receive $\{e^{(i)}\}_{i\in[n]}$.
4. Pick some $i^* \notin J$. Send $e^{(i^*)}$ to the honest prover of the single-prover protocol and receive z_0, z_1.
5. For the corrupted provers, compute $\{\tilde{e}_0^{(i)}, \tilde{z}_0^{(i)}\}_{i\in J}$ as if they were honest and got challenge $e^{(i^*)}$, and choose random $\{\tilde{e}_0^{(i)}, \tilde{z}_0^{(i)}\}_{i\notin J}$ for the honest provers such that $\prod_i \tilde{e}_0^{(i)} = e_0$ and $\prod_i \tilde{z}_0^{(i)} = z_0$. For $i \notin J$, let
$$e_0^{(i)} := \tilde{e}_0^{(i)} + \left(e^{(i)} - e^{(i^*)}\right) B_0^{(i)}$$
and
$$z_0^{(i)} := \tilde{z}_0^{(i)} \cdot \left(V_0^{(i)}\right)^{e^{(i)} - e^{(i^*)}}$$
for $B_0^{(i)} \in \mathbb{Z}_q$ and $V_0^{(i)} \in G$ randomly chosen according to the output distribution of $\mathsf{Mult}(\cdot, 1^{(i)})$ and $\mathsf{Mult}'(\cdot, 1^{(i)})$ and in accordance with $\mathsf{Mult}(\bar{b}^{(i)}, 1^{(i)})$ and $\mathsf{Mult}(v_0^{(i)}, 1^{(i)})$ for $i \in J$, respectively. Compute $\{e_1^{(i)}, z_1^{(i)}\}_{i\notin J}$ analogously.
6. Send $\{e_0^{(i)}, e_1^{(i)}, z_0^{(i)}, z_1^{(i)}\}_{i\notin J}$ to the adversary.

It is easy to see that the resulting transcript (including the non-deviating messages for the corrupted provers) is accepting if for some $e \in \mathbb{Z}_q$, $e^{(i)} = e$ for all $i \in [n]$. Furthermore, $B_0^{(i)}$ and $V_0^{(i)}$ can be seen as $\mathsf{Mult}(\tilde{b}_0^{(i)}, 1^{(i)})$ and $\mathsf{Mult}(v_0^{(i)}, 1^{(i)})$, respectively. $B_0^{(i)}$ and $V_0^{(i)}$ are hidden from the adversary by $\tilde{e}_0^{(i)}$ and $\tilde{z}_0^{(i)}$, which are chosen uniformly. In a real execution, pseudorandom zero sharing is used to hide them, and thus, the simulation is only computationally indistinguishable. However, the adversary is computationally bounded. Furthermore,

$$\begin{aligned}\sum_{i\in J} \tilde{e}_0^{(i)} + \sum_{i\notin J} e_0^{(i)} &= \sum_i \tilde{e}_0^{(i)} + \sum_{i\notin J} \left(e^{(i)} - e^{(i^*)}\right) B_0^{(i)} \\ &= e_0 + \sum_{e^{(i)} \neq e^{(i^*)}} (e^{(i)} - e^{(i^*)}) \mathsf{Mult}\left(\bar{b}^{(i)}, 1^{(i)}\right).\end{aligned}$$

It is not hard to see that this corresponds to the distribution of a real execution. The same can be proven for

$$\sum_{i\in J} \tilde{e}_1^{(i)} + \sum_{i\notin J} e_1^{(i)},$$
$$\prod_{i\in J} \psi\left(\tilde{z}_0^{(i)}\right) \cdot \prod \psi\left(z_0^{(i)}\right) / \left(\prod_i a_0^{(i)} \cdot h_0^{\sum_{i\in J} \tilde{e}_0^{(i)} + \sum_{i\notin J} e_0^{(i)}}\right),$$

and

$$\prod_{i\in J} \psi\left(\tilde{z}_1^{(i)}\right) \cdot \prod \psi\left(z_1^{(i)}\right) / \left(\prod_i a_1^{(i)} \cdot h_1^{\sum_{i\in J} \tilde{e}_1^{(i)} + \sum_{i\notin J} e_1^{(i)}}\right).$$

B.4 Proof of Theorem 1

We will prove the anonymity property by reducing the anonymity of the threshold version to anonymity of the original protocol. The reduction communicates with a (possibly corrupt) bank $\mathcal{B}^*$, a (possibly corrupt) combiner $\mathcal{C}^*$, and a (possibly corrupt) shop $\mathcal{S}^*$ as the user players $\mathcal{U}_1, \ldots \mathcal{U}_n$. In addition it communicates with users of the original protocol, playing the role of $\mathcal{B}$ and $\mathcal{S}$. Since the adversary only has passive corruption of $\mathcal{C}$, $\mathcal{C}$ will pass on messages from $\mathcal{B}^*$ and

$\mathcal{S}^*$ unmodified to the user players. The reduction passes these on to the users of the original protocol. Messages from the users in the original protocol is passed on to $\mathcal{C}^*$, some of them as secret shared values. The only communication left is the reconstruction of $\hat{B}$, however, it can easily be simulated since ρ_B perfectly hides B as long as $\mathcal{C}^*$ does not see the individual shares $B^{(i)}$ of B. It is easy to see that if an attack on the anonymity in the threshold e-cash exists, then an equivalent attack exists on the anonymity of Brands' original scheme.

The adversary cannot learn information about u_1 during opening of an account, because u_1 is generated secret shared, and the rest of the protocol is a Σ-protocol distributed by the protocol of Figure 1. The value u_1 is not used during withdrawal, and adversary cannot learn information about x_1 and x_2 during withdrawal either (that would break the representation assumption the original scheme is based on [6]). The adversary cannot learn information about u_i in the payment protocol since it is also a Σ-protocol distributed by the protocol of Figure 1. Since payment is a proof of knowledge of u_1 the adversary cannot steal electronic coins from $\mathcal{U}$.

Security against forged coins and double spending follows in the real protocol from the special soundness of the payment protocol and is therefore preserved due to Lemma 1.

C General Proofs of Knowledge

Bellare and Goldreich [2] introduced the conventional definition for proofs of knowledge, which were introduced by Goldwasser [34] earlier. Informally, a protocol is a proof of knowledge if there exists a so-called knowledge extractor that can extract a witness by rewindable block-box access to any possible prover $\mathcal{P}^*$ in time roughly the inverse of the success probability of $\mathcal{P}^*$.

Clearly, our communication model with n provers $\mathcal{P}_1, \ldots, \mathcal{P}_n$, a combiner $\mathcal{C}$ and the verifier $\mathcal{V}$ can be also be used for general proofs of knowledge. Similar to Definition 2, the definition of a proof of knowledge can be directly adapted to a distributed-prover protocol by giving the knowledge extractor rewindable black-box access to the combiner and the provers. As with Σ-protocols, we are not concerned who "knows" the witness. It is easy to see that the distributed version of a proof of knowledge achieves the adapted definition. Furthermore, Definitions 1 (completeness), 5 (full zero-knowledge), and 6 (witness hiding) also apply for general proofs of knowledge, whereas Definitions 2 (special soundness), 3 (special honest-verifier-combiner zero-knowledge), and 4 (partial zero-knowledge) depend on the three-move structure of Σ-protocols. We believe that, using multiparty computation, every zero-knowledge proof of knowledge can be turned into a distributed-prover proof of knowledge which is full zero-knowledge with threshold $t < n/3$. However, the resulting protocol might require communication between the provers and might not be as efficient as the protocols presented in this paper.

D Forward and Backward Untraceability

In user-centric protocols we often want to preserve the untraceability of a user (i.e., different protocol executions involving the same user should not be linkable) even if the state of the device executing the user's side of the protocol is revealed at some point in time. This privacy notion is commonly called *backward and forward untraceability*. One usually considers an adversary who passively corrupts a user device at a certain time τ but otherwise may only eavesdrop communication at times $\tau' < \tau$ and $\tau'' > \tau$. Then by backward untraceability we mean that the adversary is not able to tell whether a protocol execution at time τ' involves this user device. By forward untraceability we mean that the adversary cannot decide whether the user device is involved in a protocol execution at time τ'' assuming that the adversary missed eavesdropping the communication of this device at least once between times τ and τ''. Unfortunately, a commonly accepted formal definition of these properties is still missing.

Nevertheless, it would be nice if threshold user-centric protocols, as introduced here, also satisfied the above properties, where we allow the passive corruption of up to $n-1$ devices of the same user and the combiner at time τ. However, the use of pseudorandom secret and zero sharing with *fixed* keys constitutes a problem. More precisely, PRSS and PRZS involves a PRF (cf. Appendix A.2), whose past and future outputs can be computed once the key and the input to the function are known. In this way, an adversary revealing the state of some device at time τ can usually predict future and past messages and thus trace the device. For instance, it is easy to see that in this case in the protocol shown in Figure 5 all messages from $\mathcal{U}_i$ would be computable.

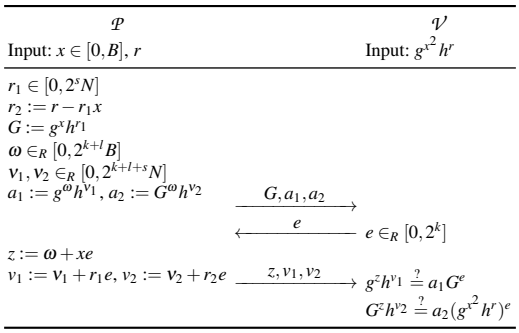

Fig. 6. Proof that a commitment hides a square

In order to achieve backward and forward untraceability one needs to update the keys of all PRFs involved in the protocol after every execution. For forward untraceability the idea is to use some external randomness like a challenge sent by the other party. To obtain backward untraceability the update function should be one-way. Hence, a simple solution could be the following: Let PRF be a pseudorandom function used for PRSS and PRZS and PRF' be another one whose output and key lengths correspond to the key length of PRF. A new key K_i for PRF (and PRF') could then be computed as $K_i := \mathsf{PRF}'_{K_{i-1}}(c)$, where c is the challenge received. A more formal treatment of backward and forward untraceability for threshold protocols is left as future work.

E Details of the Range Proof

Figure 6 shows the details of the proof that a commitment hides a square. Adapting to distributed provers can be done by using multiplicative integer secret sharing as introduced in Section 2.2, and PRSS for this secret sharing scheme to choose r_1 and ω. This is required because the proof involves multiplications involving x, r_1, and ω, with the latter two differing from protocol run to protocol run. Figure 7 gives the details. $x^{(i)}$, $r^{(i)}$, and $1^{(i)}$ denote a multiplicative secret sharing of $x \in [0,B]$, r, and 1, respectively. s, k, l, and m are security parameters, e.g., m is the security parameter of the sharing scheme. As in the distributed OR proof, we use $\mathsf{Mult}(\cdot, 1^{(i)})$ to convert a multiplicative secret sharing into an additive one, and PRZS to randomize the output of Mult. Note that we use PRSS and PRZS with varying intervals here. $\mathsf{PRSS}(i,[0,b])$ denotes that $\mathcal{P}_i$ chooses pseudorandom inputs in the interval $[0,b]$. For any maximal unqualified set, the resulting secret shared number will then be computationally indistinguishable to a uniform number in some interval $[c,c+b]$. This reflects the indistinguishability property of the single-prover protocol. For $\mathsf{PRZS}(i)$, the pseudorandom inputs are chosen in an interval exponential bigger than the maximal output of the adjacent Mult, which is required to achieve the desired indistinguishability. We omit a formal proof of the security because it is similar to the one in Section 3.1.

F Boudot's Range Proof

In this section, we will outline how to adapt the range proof by Boudot [5] to our distributed setting. All protocols in this section use the Fujisaki-Okamoto commitment scheme $\mathsf{com}(x,r) := g^x h^r$ for $g,h \in \mathbb{Z}_N^*$. Clearly, com is a homomorphic function from $\mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}_N^*$. Therefore, a protocol as in Section 3.1 can be constructed. This protocol cannot be proven to be sound in the same way because there is not necessarily a prime q such that a preimage of $\mathsf{com}(x,r)^q$ can be found. However, soundness can be achieved if the strong RSA assumption for $\mathbb{Z}_N^*$ holds.

Chan et al. [12] proposed an extension to a range proof for x, as described in Figure 8. Note that it is only proven that x is in $[-2^{k+l}b, 2^{k+l}b]$ although x has to be in $[0,b]$ to achieve zero-knowledge. The ratio of the two intervals is called the expansion rate.

A distributed version be the protocol can be obtained by letting the players choose ω and v as a random additive secret sharing, similar as in Figure 1. Because $\mathbb{Z}$ is not finite, the resulting ω and v are not distributed uniformly.

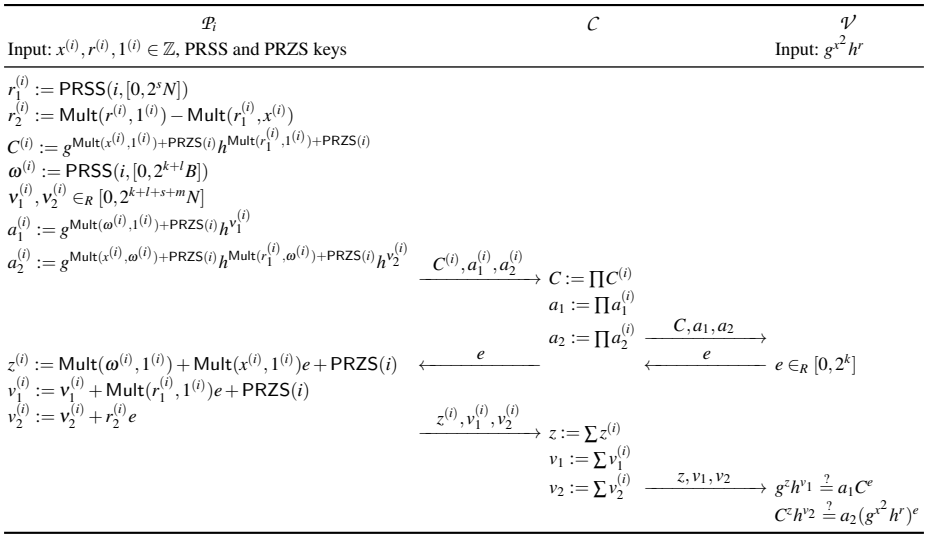

Fig. 7. Distributed proof that a commitment hides a square

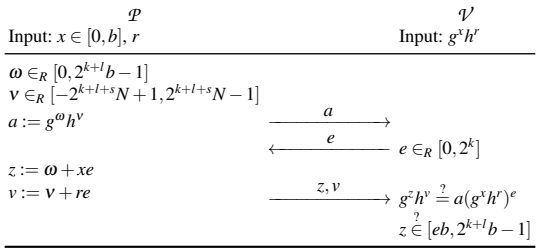

Fig. 8. CFT range proof

However, if every prover chooses his share of ω and v in the same interval as they are chosen in the original protocol, z and v still are distributed statistically close to ω and v, respectively. Thus, the distributed protocol is statistical honest-verifier-combiner zero-knowledge. The only difference to the single-prover protocol is that the expansion rate is enlarged by the number of players.

Boudot uses the above protocol as a building block for a range proof with expansion rate 1. A further building block is the proof of square described in Section 3.3. To use the CFT proof and the square proof in combination as in [5], multiplicative secret sharing also has to be used in the CFT proof. This is straightforward because of the linearity, however, the expansion rate is enlarged by $\binom{n}{t}$ instead of n due to the usage of PRSS. Furthermore, Boudot's proof that $\mathsf{com}(x, r)$ hides $x \in [a, b]$ involves the computation of $\lfloor \sqrt{x-a} \rfloor$ and $\lfloor \sqrt{b-x} \rfloor$ as well the choice of a random number in $[-2^s N + 1, 2^s N - 1]$. The former can be done in a setup phase if a and b are fixed, and the latter can be replaced by choosing a sum of random numbers in the mentioned interval with PRSS. This does not affect the security or the completeness because the chosen number is only used as randomness for a commitment, and because the interval of the hiding randomness in the relevant zero-knowledge proofs will be expanded in the same way since PRSS is also used there.

G Universally Composable Zero-Knowledge

In this section, we present a UC functionality that can be implemented using a Σ-protocol Π that achieves partial zero-knowledge with threshold t as in Definition 4, that is, all protocols in this paper. Since the combiner clearly is

more powerful than the verifier in those protocols, we combine the entities in one verifier V for simplicity, while the provers $P_1, \ldots, P_n$ remain as in the previous chapter. Note that we cannot prove the usual universally composable notion of soundness because Σ-protocols do not offer straight-line extractability as Ω-protocols do. We briefly introduce the framework of universal composability and trapdoor commitments. We will use the latter in our protocol.

G.1 Universal Composability

The framework of universal composability was introduced by Canetti [11]. It models protocols as system of interactive Turing machines of parties, a communication resource, and an environment. The properties of a protocol are formulated as a so-called ideal functionality to which the adversary has limited access. The adversary also can corrupt parties according to restrictions depending on the security guarantees of a protocol. A protocol is said to securely implement a certain functionality if there exists a simulator acting as the adversary on the ideal functionality such that any environment acting as the adversary on the simulator or the corrupted parties and interacting with the honest parties cannot distinguish between the protocol and simulation. The central theorem of the framework says that using a protocol securely implementing a functionality is as secure as using the functionality. If there is a simulator that makes functionality $\mathcal{F}_0$ indistinguishable to a protocol Π that uses the functionality $\mathcal{F}_1$ internally, we say that Π securely implements $\mathcal{F}_0$ in the $\mathcal{F}_1$-hybrid model.

An essential functionality in this framework is $\mathcal{F}_{\mathsf{CRS}}^{D}$, which provides a so-called common reference string chosen according to the distribution D. It is often used to have a public key that cannot be influenced be the adversary and where the random bits that would be needed to generate it are unknown to the adversary.

G.2 Trapdoor Commitments

Commitment schemes allows a party to commit itself to a message without revealing it. When the message is revealed later, it can be checked whether it is same as used for the commitment. Trapdoor commitments schemes were introduced by Brassard et al. [7] under the name chameleon blobs. They are common in the construction of zero-knowledge protocols [7, 19, 27, 32]. A trapdoor commitment schemes comes with the following procedures:

Set-up Generate a public key pk and trapdoor τ.

Commitment Choose randomness d according to a distribution $D_{\mathsf{Com}_{pk}}$ and compute the commitment $c := \mathsf{Com}_{pk}(d,m)$ to a message m.

Opening Reveal (d,m), which allows to check whether $c = \mathsf{Com}_{pk}(d,m)$.

Trapdoor opening Given (τ,d,m,m'), compute "randomness" d' such that $\mathsf{Com}_{pk}(d',m') = \mathsf{Com}_{pk}(d,m)$. In other words, the trapdoor τ allows to open a commitment $\mathsf{Com}_{pk}(d,m)$ to any message m'.

Any commitment scheme has a hiding and a binding property, representing the security requirements for the sender and the receiver of the commitment, respectively. It is straight-forward to prove that not both properties can be achieved unconditionally by the same scheme. In the case of a trapdoor commitment scheme, the binding property clearly only holds for computationally bounded adversaries.

Hiding A commitment does not reveal any information about m, that is, $\mathsf{Com}_{pk}(d,m)$ and $\mathsf{Com}_{pk}(d',m')$ are indistinguishable for every m and m' and independent d and d'.

Binding There is only a negligible chance of opening a commitment to two different messages, that is, computing (m,m',d,d') such that $\mathsf{Com}_{pk}(d,m) = \mathsf{Com}_{pk}(d',m')$ if the trapdoor τ is not known and, moreover, no two different openings of a commitment are known.

See Garay et al. [32] for a more technical definition and a concrete implementation of simulation-sound trapdoor commitment schemes that they use in their construction of UC zero-knowledge proofs. To generate the public key of such schemes, it is common to use $\mathcal{F}_{\mathsf{CRS}}^{D}$ with D being the distribution of the public keys in order to prevent the adversary from knowing the trapdoor.

G.3 Functionality, Protocol, and Simulation

To model the setup assumptions, we design a generator functionality $\mathcal{F}_{\mathsf{Gen}}^{R,\Pi}$ that guarantees the availability of a witness and further keys if needed. But first, we present the functionality $\mathcal{F}_{\mathsf{ZK}}^{R}$.

1. Choose $(x,w) \in R$.
2. If there is a corrupted prover, wait for a message from the adversary that can be `OK` or `FAIL`.
3. Output x to V if $(x,w) \in R$ and the adversary did not send `FAIL`, and $\perp$ otherwise.

This functionality clearly corresponds to the definition of full zero-knowledge because the only the result of the witness check is revealed to the adversary. The generator functionality $\mathcal{F}_{\mathsf{Gen}}^{R,\Pi}$ is straight-forward.

1. Choose $(x,w) \in R$ and key material if needed by the protocol Π.
2. Generate a secret sharing $\{(x_i,w_i)\}_{i\in[n]}$ from (x,w).
3. Send (x_i,w_i) and the necessary key material to P_i.

Our protocol makes use of a trapdoor commitment Com_{pk} to allow a zero-knowledge simulation.

1. P_i receives (x_i,w_i) from $\mathcal{F}_{\mathsf{Gen}}^{R,\Pi}$.
2. P_i computes a_i according to Π, chooses $d_i \in D_{\mathsf{Com}_{pk}}$, and computes $c_i := \mathsf{Com}_{pk}(d_i,a_i)$.
3. P_i sends (x_i,c_i) to V.
4. V chooses a random challenge $e \in \mathbb{Z}_q$ and sends it to all provers.
5. P_i computes z_i according to Π and sends (d_i,a_i,z_i) to V.
6. V reconstructs a and z, and checks z and whether $c_i = \mathsf{Com}_{pk}(d_i,a_i)$ for all $i \in [n]$.

We present a simulator for a set A of at most t corrupted provers and a possibly corrupted V. The appearance of $\{e_i\}_{i\in[n]}$ arises from the fact that a corrupted V might send different challenges to the provers.

1. Generate common reference string and trapdoor for Com_{pk}.
2. For all $i \in A$, choose key material if necessary and $w_i \in G$ uniformly at random, send (x_i,w_i) and the key material to P_i, and receive (x_i,c_i) from P_i.
3. If V is corrupted:
   (a) Use the trapdoor of Com_{pk} to generate equivocal commitments $\{c_i\}_{i\notin A}$. Send them $\{c_i\}_{i\in[n]}$ to V, and receive $\{e_i\}_{i\in[n]}$.
   (b) Simulate Π with the partial zero-knowledge simulator using $\{e_i\}_{i\in[n]}$. Note that the difference between Π and this protocol does not matter because $\{e_i\}_{i\in[n]}$ is fixed in both cases. That is, in Π, P_i sends a_i, receives e_i, and replies with z_i, whereas, in this protocol, P_i sends c_i, receives e_i, and replies with (d_i,a_i,z_i). Since e_i is fixed, it is straightforward to use P_i from Π in the partial zero-knowledge simulator.
   (c) Use the trapdoor of Com_{pk} to open $\{c_i\}_{i\notin A}$ to $\{a_i\}_{i\notin A}$, that is, to generate $\{d_i\}_{i\notin A}$ such that $c_i = \mathsf{Com}_{pk}(d_i,a_i)$ for all $i \notin A$.
4. If V is not corrupted, but at least one prover is:
   (a) Choose e as an honest prover would.
   (b) Compute $\{\tilde{a}_i,\tilde{z}_i\}_{i\in A}$ according to Π using e, $\{w_i\}_{i\in[n]}$, and the key material generated earlier.
   (c) For ever corrupted P_i, receive c_i, send e, and receive (d_i,a_i,z_i).
   (d) Check whether $c_i = \mathsf{Com}_{pk}(d_i,a_i)$ for all $i \in A$ and whether $\{(a_i,z_i)\}_{i\in A}$ matches $\{\tilde{a}_i,\tilde{z}_i\}_{i\in A}$ generated earlier.[1] If any check fails, send `FAIL` to $\mathcal{F}_{\mathsf{ZK}}^{R}$. Otherwise, send `OK`.

Clearly, the simulator lets the verifier reject in the same situation as the protocol would. Indistinguishability follows from partial zero-knowledge.

[1] With linear secret sharing, this can be seen as follows. Let $\lambda_{[n]}$ be the reconstruction vector. If $\sum_{i\in A}\lambda_{[n],i}a_i \neq \sum_{i\in A}\lambda_{[n],i}\tilde{a}_i$, the reconstruction will be different depending on $\{a_i\}_{i\in A}$ or $\{\tilde{a}_i\}_{i\in A}$ is used.

$\mathcal{P}$		$\mathcal{V}$
Input: $w \in G$		Input: $h = \psi(w) \in H$
$r \xleftarrow{\$} G$		
$a := \psi(r) \in H$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{\$} \mathbb{Z}_q$
$z := rw^e \in G$	$\xrightarrow{\quad z \quad}$	$\psi(z) \stackrel{?}{=} ah^e \in H$

Fig. 9. Proof of knowledge of a preimage $w \in G$ of an element $h \in H$ under a group homomorphism $\psi : G \to H$

$\mathcal{P}$		$\mathcal{V}$
Input: $b \in \{0,1\}, w_b \in G$ $h_b = \psi(w_b), h_{\bar{b}} \in H$		Input: $h_0, h_1 \in H$
$r_0, r_1 \xleftarrow{\$} G, d_0, d_1 \xleftarrow{\$} \mathbb{Z}_q$		
$a_0 := \psi(r_0) h_0^{-bd_0} \in H$		
$a_1 := \psi(r_1) h_1^{-\bar{b}d_1} \in H$	$\xrightarrow{\quad a_0, a_1 \quad}$	
$e_0 := bd_0 + \bar{b}(e - d_1) \in \mathbb{Z}_q$	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{\$} \mathbb{Z}_q$
$e_1 := b(e - d_0) + \bar{b}d_1 \in \mathbb{Z}_q$		
$v_b := w_b \in G$		
$v_{\bar{b}} := 1 \in G$		
$z_0 := r_0 v_0^{e_0} \in G$		
$z_1 := r_1 v_1^{e_1} \in G$	$\xrightarrow{\quad e_0, e_1, z_0, z_1 \quad}$	$e \stackrel{?}{=} e_0 + e_1 \in \mathbb{Z}_q$
		$\psi(z_0) \stackrel{?}{=} a_0 h_0^{e_0} \in H$
		$\psi(z_1) \stackrel{?}{=} a_1 h_1^{e_1} \in H$

Fig. 10. Alternative version of the proof of knowledge of a ψ-preimage of h_b, given $h_0, h_1 \in H$. Computations on prover's side have been unified, i.e., to compute a_0, a_1, e_0, e_1 and z_0, z_1 the same computations are performed.