Proposal: Σ-protocols

Stephan Krenn[1] and Michele Orrù[2]

[1] AIT Austrian Institute of Technology, Vienna, Austria
[2] University of California, Berkeley, United States

Abstract. Over the last years, zero-knowledge proofs of knowledge based on Σ-protocols have found numerous applications. However, up to date there is still a lack of standardization of such protocols, potentially hindering even broader deployment, and increasing the risk of insecure implementations. This document proposes a standardization effort for non-interactive Σ-protocols in prime order groups, allowing for AND and OR composition, either in compact (challenge, response) or batchable form (commitment, response). The document provides the necessary formal background, specifies the protocols in full details, provides examples, suggests concrete instantiations (e.g., regarding the selection of elliptic curves or hash functions), and provides guidelines to ease the secure and compatible implementation of Σ-protocols.

Keywords: Zero-knowledge proofs of knowledge · Σ-protocols.

1 Introduction

Zero-knowledge [GMR89] proofs of knowledge [BG93] allow a prover to convince a verifier that he knows a secret piece of information, without revealing anything else that what is already revealed by the claim itself. Many practically relevant proof goals can be realized using so-called Σ-protocols, or their non-interactive counterparts, which can be proven secure in the random oracle model without the need for a common reference string. Introduced by Schnorr [Sch91] over 30 years ago, they are now widely used in practice because of their simplicity, maturity, and versatility.

Σ-protocols played an essential component in the building of a number of cryptographic constructions, such as anonymous credentials [CMZ14], password-authenticated key exchange [HR11], signatures [Sch90], ring signatures [MP15], blind signatures [PS97], multi signatures [NRSW20], threshold signatures [KG20] and more. Lueks et al. [LKF+19], reported that solely in the years 2018-2019 editions of PETS, ACM CCS, WPES, NDSS, 7 publications use Σ-protocols. Yet, there is no standardized way of implementing them and as a result a lot of implementations have shared common pitfalls (cf. Section 2).

We make a first attempt at the standardization of Σ-protocols by proposing a framework for proving statements about discrete logarithms and related relations in prime-order groups, targeting specifically groups in elliptic curves. To the best of our knowledge, currently deployed and implemented Σ-protocols mostly rely on the following sub-set of features:

- **Discrete logarithm equality, Diffie-Hellman triples.** Proving discrete logarithm equality, DDH triples, or more in general linear relations among secrets is an essential task for some protocols. This is the case of some VOPRFs such as [JKK14], which is currently in process of standardization [DSW19], and internally employs Σ-protocols to prove discrete logarithm equality. DLEQ proofs are also being used by Privacy Pass [DGS+18], a lightweight anonymous credential that has been implemented and deployed at scale by Cloudflare, Brave Browser, and others. There are also other works, published in recent academic conferences, that rely on this type of relations: Cryptography for #metoo [KKR19]; Solidus [CZJ+17]; ClaimChain [KLI+18].
- **Knowledge of plaintexts, openings of commitments** The so-called AND-composition of dlog relations has been used over a number of protocols, including in Algebraic MACs [CMZ14]. Algebraic MACs are now used in Signal's group chats [CPZ20] and deployed to millions of users.
- **Proving knowledge of one among multiple discrete logarithms.** OR-composition of Σ-protocols has been used for range proofs [MP15], and ring signatures: it enables for private transactions in CryptoNote (Monero) [Noe15]. OR-composition is also used in other tools such as Mesh [AG19].

All these protocols can be implemented using the API that we propose in Section 4.2. A request for standardization has also been posted on the zk-proof community and received a fair amount of interest.

2 Background and Motivation

So far, there has been little literature meant to help cryptography engineers implement correctly Σ-protocols for arbitrary statements about discrete logs. Despite Schnorr signatures and zero-knowledge proofs have already been standardized (respectively in [JL17] and [Hao17]), there is still no formal, established way to implement Σ-protocols and their composition. Additionally, as academic papers focus on proving the security of Σ-protocols in generic cyclic groups, the implementation details are

often times overlooked, and, as a result, a lot of insecure implementations have been published in the past. To name a few, well-known pitfalls that have led to insecure implementations:

- **Implementation over non-prime order groups led to small subgroup attacks**. The cyclic group where proofs are being implemented must be carefully selected. Implementing Σ-protocol over Weierstrass curves is appealing for its performances and straight-line scalar multiplication formulæ. However, the presence of a small cofactor could be fatal for security. In 2017, Monero [tt17] disclosed a vulnerability in Cryptonote-based e-currencies that would allow double-spending due to the use of `curve25519` [Ber06] in place of a prime-order curve.
- **Leakage, partial leakage, or reuse of the commitment is fatal.** The first message in a Σ-protocol, sometimes called *nonce* or *commitment*, must be uniformly distributed in order to preserve zero-knowledge. Leakage of even just of a few bits of the nonce could allow for the complete recovery of the witness [HGS01,Ble00,ANT+20]. A blatant instance of this mistake was uncovered in 2010, by the hacker group `fail0verflow` (for EC-DSA). They showed how SONY was reusing the same nonce for digitally signing PlayStation 3 games[3]. The member could exploit this to calculate the private key, and create valid signatures.
- **A weak implementation of Fiat–Shamir heuristic compromises adaptive security.** The second message in a Σ-protocol, sometimes called *challenge*, if computed non-interactively must include not only the commitment, but the full statement and the group description.

  If the random oracle is invoked solely on the commitment (c.f. weak Fiat-Shamir transformation [BPW12, Def. 2]), then it is possible, given a proof π, to produce another proof π' for a different statement without knowing its witness. For instance, if the group generator G is not included in the hash input, then it is possible to prove statements under a different generator αG, for any $\alpha \in \mathbb{Z}_p$. Similarly, if $X \in \mathbb{G}$, part of the statement, is not included in the query to the random oracle, it is possible to compute proofs for βX, for any $\beta \in \mathbb{Z}_p$. Mistakes of this class were uncovered by Bernhard et al. [BPW12], by Haines et al. [HLPT20], and by Cortier et al. [CGY20]. They all showed how some voting systems, respectively Helios, Scytl-SwissPost, and Bele-

[3] https://media.ccc.de/v/27c3-4087-en-console_hacking_2010

nios, incorrectly implemented the Fiat–Shamir heuristic, allowing for tampering of votes.

We stress that is not straightforward to implement Σ-protocols given the currently available engineering literature. For instance, the current standardized `ed25519` cannot be immediately adapted to a zero-knowledge proof in a secure way, because of its malleability [BDL+12, p. 7], and the different behavior on the batched and compressed form[4].

Hence, we hereby propose the creation of a working group that includes both recent cryptanalytic insights as well as the (partial) solutions described in other known standardization documents.

3 Notation and Terminology

For the purpose of this document, the following notation will be used:

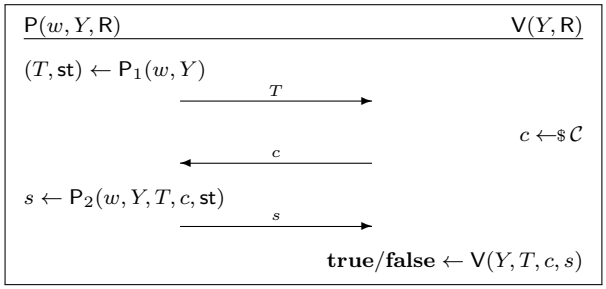
λ	main security parameter
$a, x, w, \ldots$	elements mod p
$\mathbb{G} = \langle G \rangle$	cyclic group of prime order p generated by G
$G, H, Y, \ldots$	group elements in $\mathbb{G}$
$\mathsf{P}, \mathsf{V}, \ldots$	potentially randomized algorithms
$x := 1$	assignment of the value 1 to x
$x \leftarrow\!\!\$\, \mathcal{S}$	assignment of a uniformly random element in $\mathcal{S}$ to x
$x \leftarrow \mathsf{Alg}(in)$	assignment of the output of a randomized algorithm Alg on input in to x
R	binary relation
$\mathcal{L}(\mathsf{R})$	language induced by a binary relation R
$\lvert y \rvert$	bitlength of a string
$\mathbb{N}, \mathbb{Z}$	non-negative natural numbers and integers, respectively

3.1 Formal Definitions

In the following we provide the formal definitions required for the remainder of this document.

Definition 1. *For two groups $\mathbb{G}_1, \mathbb{G}_2$, a function $\varphi : \mathbb{G}_1 \to \mathbb{G}_2 : x \mapsto \varphi(x)$ is a* (group) homomorphism*, if and only if for all $a, b \in \mathbb{G}_1$ it holds that $\varphi(a + b) = \varphi(a) + \varphi(b)$.*

[4] https://hdevalence.ca/blog/2020-10-04-its-25519am



Protocol 1: Generic flow of a Σ-protocol.

Σ-Protocols

In the following, we formally describe the class of Σ-protocols, which covers all protocols considered in the remainder of this document. For an in-depth discussion of the underlying theory we refer to Cramer [Cra97].

Definition 2. *Let* R *be a binary relation and let* $(Y, w) \in \mathsf{R}$*. An interactive two-party protocol specified by algorithms* $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ *is called a* Σ-protocol *for* R *with challenge set* $\mathcal{C}$*, public input* Y*, and private input* w*, if and only if it satisfies the following conditions:*

3-move form: *The protocol is of the following form (cf. also Protocol 1):*

- *The prover computes* $(T, \mathsf{st}) \leftarrow \mathsf{P}_1(w, Y)$ *and sends* T *to the verifier, while keeping* st *secret.*
- *The verifier draws* $c \leftarrow\!\!\$\, \mathcal{C}$ *and returns it to the prover.*
- *The prover computes* $s \leftarrow \mathsf{P}_2(w, Y, c, \mathsf{st})$ *and sends* s *to the verifier.*
- *The verifier accepts the protocol run, if and only if* $\mathsf{V}(Y, T, c, s) =$ **true**, *otherwise it rejects.*

The protocol transcript (T, c, s) *is called* accepting *if the verifier accepts the protocol run.*

Completeness: *If* $(Y, w) \in \mathsf{R}$*, then the verifier always outputs* **true**.

k-special soundness: *There exists an efficient algorithm* Ext *which, given* Y *and* k *accepting protocol transcripts* (T, c_i, s_i) *for* $i = 1, \ldots, k$ *for public input* Y *with the same first message but pairwise distinct challenges (i.e.,* $c_i \neq c_j$ *for* $i \neq j$*), returns* w *such that* $(Y, w) \in \mathsf{R}$.

Special honest-verifier zero-knowledge: *There exists an efficient algorithm* Sim*, which on input* Y *and a challenge* $c \in \mathcal{C}$*, outputs transcripts of the form* (T, c, s) *whose distribution is indistinguishable from accepting protocol transcripts generated by real protocol runs on public input* y *and with challenge* c.

Note that Σ-protocols were originally introduced for the case $k = 2$ only. However, we use the above generalized definition as certain practically relevant optimization techniques require $k > 2$.

A Σ-protocol is said to have *unpredictable commitments* if the probability of generating a collision in the first message is negligible, i.e., if there is a negligible function $\mathsf{negl}(\lambda)$ such that for all $(Y, w) \in \mathsf{R}$ it holds that:

$$\Pr\left[T' = T'' : T' \leftarrow \mathsf{P}_1(w, Y), T'' \leftarrow \mathsf{P}_1(w, Y)\right] \leq \mathsf{negl}(\lambda) \ .$$

A Σ-protocol is said to have *perfectly unique responses* if it is infeasible to find two distinct valid responses for a given first message and fixed challenge, i.e., there exist no values Y, T, c, s', s'' with $s' \neq s''$ such that $\mathsf{V}(Y, T, c, s') = \mathsf{V}(Y, T, c, s'') = \mathbf{true}$. In case that it is only computationally infeasible (also for a quantum attacker) to find two responses for given Y, T, c, the protocol is said to have *(computationally) unique responses*.

Proof Systems and Proofs of Knowledge

The concept of interactive proofs of knowledge was first mentioned by Goldreich et al. [GMR85], and then refined by Feige et al. [FFS87]. The definitions in the following are due to Bellare and Goldreich [BG93].

Intuitively, a proof system is sound, if it is not possible to make the verifier accept for statements for which no valid witness exists, cf. also [ZKP19, 1.6.2].

Definition 3. *Let* R *be a binary relation,* $\sigma : \mathbb{N} \to [0, 1]$*, and let* P *and* V *specify a probabilistic interactive protocol, where at least* V *is polynomial time. The protocol is called* sound *with* soundness error σ*, if for every* $Y \notin \mathcal{L}(\mathsf{R})$*, and every interactive algorithm* P^**, the probability that* P^* *makes* V *output* **true** *on common input* Y *is bounded above by* $\sigma(|Y|)$*.*

Informally, an interactive protocol is a proof of knowledge, if every party that is able to make the verifier accept with sufficiently high probability needs to know a valid witness or would be able to compute such a witness, cf. also [ZKP19, 1.6.3].

Definition 4. *Let* R *be a binary relation,* $\kappa : \mathbb{N} \to [0, 1]$ *and let* P *and* V *specify a probabilistic interactive protocol, where at least* V *is polynomial time. The protocol is then called a* proof of knowledge *for* R *with* knowledge error κ*, if the following conditions are satisfied:*

Completeness: *If* $(Y, w) \in \mathsf{R}$*, then the verifier (on input* Y*) always outputs* **true** *in an interaction with the prover (on input* (Y, w)*).*

Knowledge soundness: *There exists a probabilistic algorithm* Ext *(the* knowledge extractor*) and a polynomial* $\mathsf{poly}(\lambda)$ *such that the following holds: for every interactive algorithm* P^* *and every* $Y \in \mathcal{L}(\mathsf{R})$*, let* $\varepsilon(Y, \mathsf{P}^*)$ *be the probability that* P^* *makes* V *output* **true** *on common input* Y*. If* $\varepsilon(Y, \mathsf{P}^*) > \kappa(|Y|)$*, then* Ext*, having rewindable black-box access to* P^**, outputs* w' *satisfying* $(Y, w') \in \mathsf{R}$ *in an expected number of steps bounded by* $\frac{\mathsf{poly}(|Y|)}{\varepsilon(Y,\mathsf{P}^*)-\kappa(|Y|)}$.

Informally, a proof system is simulation sound, if no adversary is able to generate wrong proofs, even if it can request arbitrary fake proofs from a simulator. The following definition is taken from Unruh [Unr17].

Definition 5 (Informal). *A non-interactive proof system* (P, V) *is* strongly simulation sound *with respect to a simulator* Sim*, if and only if for any (potentially quantum) polynomial-time algorithm* $\mathcal{A}$*, the following probability is negligible:*

$$\Pr\left[\mathsf{V}^{H'}(Y, \pi) = \textbf{true} \wedge Y \notin \mathcal{L}(\mathsf{R}) \wedge (Y, \pi) \notin \mathsf{S} : (Y, \pi) \leftarrow \mathcal{A}^{H,\mathsf{Sim}}(1^\lambda)\right] ,$$

where π *denotes the proof output by the prover. Here,* H *is a random oracle that the adversary has access to, and* H' *is the state of the random oracle after the* $\mathcal{A}$*'s output. Furthermore,* S *is a list of all input/output pairs of* $\mathcal{A}$*'s invocations of* Sim*.*

Similarly, the proof system is weakly simulation sound *if the following probability is negligible:*

$$\Pr\left[\mathsf{V}^{H'}(Y, \pi) = \textbf{true} \wedge Y \notin \mathcal{L}(\mathsf{R}) \wedge Y \notin \mathsf{S} : (Y, \pi) \leftarrow \mathcal{A}^{H,\mathsf{Sim}}(1^\lambda)\right] ,$$

where S *now only contains the inputs of* $\mathcal{A}$*'s invocations of* Sim*.*

Finally, simulation extractability models the property that any adversary being able to generate a valid proof for a statement Y needs to know a valid witness w for Y, even if it had seen many simulated proofs before. We omit a formal definition here, and refer, e.g., to Unruh [Unr17].

4 Constructions for Σ-Protocols

This section is structured in two main parts: in Section 4.1, we study the generic construction of Σ-protocols in prime-order groups. In Section 4.2, we study AND and OR-composition of the basic protocol. For in-depth discussions of such protocols we refer to [Ban05,Mau09,Mau15].

4.1 Basic Σ-Protocols in Prime-Order Groups

A basic Σ-protocol for the relation:

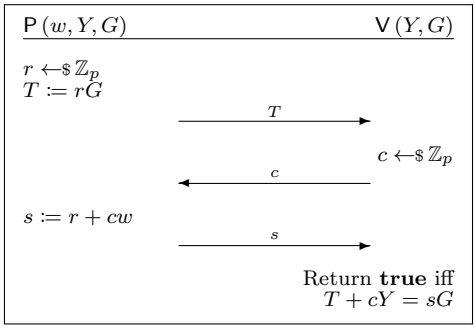
$$\mathsf{R} = \{((Y_1, \ldots, Y_m), (w_1, \ldots, w_n)) : (Y_1, \ldots, Y_m) = \varphi(w_1, \ldots, w_n)\}$$

for a group homomorphism $\varphi : \mathbb{Z}_p^n \to \mathbb{G}^m$ is given by the following algorithms:

1. The prover's first algorithm $\mathsf{P}_1(\mathbf{w}, \mathbf{Y})$ consists of the following steps:
   (a) It chooses random elements $r_1, \ldots, r_n \leftarrow_\$ \mathbb{Z}_p$.
   (b) It then computes $(T_1, \ldots, T_m) \leftarrow \varphi(r_1, \ldots, r_n)$.
   (c) The algorithm sets $\mathsf{st} \coloneqq (r_1, \ldots, r_n)$ and $\mathbf{T} \coloneqq (T_1, \ldots, T_m)$.
   (d) It finally outputs $(\mathbf{T}, \mathsf{st})$.
2. The prover's second algorithm $\mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, \mathsf{st})$ proceeds as follows:
   (a) It checks that $c \in \mathbb{Z}_p$ and aborts if this is not the case.
   (b) It then parses $(r_1, \ldots, r_n) \coloneqq \mathsf{st}$ and $(w_1, \ldots, w_n) \coloneqq \mathbf{w}$.
   (c) It computes its response as $s_i \coloneqq r_i + cw_i$ for $i = 1, \ldots, n$.
3. The verifier's algorithm $\mathsf{V}(\mathbf{Y}, \mathbf{T}, c, \mathbf{s})$ proceeds as follows:
   (a) It parses $(s_1, \ldots, s_n) \coloneqq \mathbf{s}$, $(T_1, \ldots, T_m) \coloneqq \mathbf{T}$, and $(Y_1, \ldots, Y_m) = \mathbf{Y}$.
   (b) It checks that $s_i \in \mathbb{Z}_p$ for $i = 1, \ldots, n$ and $T_j \in \mathbb{G}$ for $j = 1, \ldots, m$, and outputs 0 if this is not the case.
   (c) It checks whether $(T_1 + cY_1, \ldots, T_m + cY_m) = \varphi(s_1, \ldots, s_n)$, and outputs **true** if this is the case; otherwise, V outputs **false**.
4. The required simulator $\mathsf{Sim}(\mathbf{Y}, c)$ for a basic Σ-protocol works as follows:
   (a) It parses $(Y_1, \ldots, Y_m) \coloneqq \mathbf{Y}$.
   (b) It chooses $s_1, \ldots, s_n \leftarrow_\$ \mathbb{Z}_p$.
   (c) It sets $(T_1, \ldots, T_m) \coloneqq \varphi(s_1, \ldots, s_n) - c(Y_1, \ldots, Y_m)$.
   (d) Finally, the algorithm outputs the simulated transcript by setting $(\mathbf{T}, c, \mathbf{s}) \coloneqq ((T_1, \ldots, T_m), c, (s_1, \ldots, s_n))$.

Proving linear relations among witnesses. While the above protocol allows one to efficiently prove knowledge of a pre-image under a homomorphism, many protocols found in the literature require one to prove relations among witnesses. Specifically, they require to prove relations like the following:

$$\mathsf{R} = \left\{ ((Y_1, \ldots, Y_m), (w_1, \ldots, w_n)) : \begin{array}{c} (Y_1, \ldots, Y_m) = \varphi(w_1, \ldots, w_n) \\ A(w_1, \ldots, w_n) = (b_1, \ldots, b_k) \end{array} \right\},$$



Protocol 2: Proving knowledge of a discrete logarithm.

where the matrix $A \in \mathbb{Z}_p^{k \times n}$ and vector $(b_1, \ldots, b_k) \in \mathbb{Z}_p^k$ specify the system of linear equations.

Proving such a relation can easily by achieved by modifying the above protocol as follows:

- In Item 1a, the prover draws the randomnesses such that they satisfy the system of equations, i.e., such that $A(r_1, \ldots, r_n) = (b_1, \ldots, b_k)$.
- In Item 3b, the verifier additionally checks that $A(s_1, \ldots, s_n) = (c + 1)(b_1, \ldots, b_k)$ and outputs **false** if this is not the case.
- In Item 4b, the simulator draws the responses such that they satisfy the verification equations, i.e., such that $A(s_1, \ldots, s_n) = (c + 1)(b_1, \ldots, b_k)$.

Examples

Example 1 (DLOG). Let $\mathbb{G}$ be a group over an elliptic curve with prime order p. Proving knowledge of the discrete logarithm w of a point Y in base G means proving knowledge of $w \in \mathbb{Z}_p$ such that $Y = wG$. For a description of this proof goal in the general case of residue classes, we also refer to [ZKP19, 1.4.1].

Using the above notation, we have $\varphi : \mathbb{Z}_p^2 \to \mathbb{G} : x \mapsto xG$. The protocol flow is then as depicted in Protocol 2.

For a given challenge $c \in \mathbb{Z}_p$, the simulator chooses $s \leftarrow\$ \mathbb{Z}_p$, and sets $T \leftarrow sG - cY$. It then outputs the simulated transcript (T, c, s).

Example 2 (DLEQ). Let $\mathbb{G}$ be a group over an elliptic curve with prime order p. Proving equality of the known discrete logarithm w of Y_1 in base G and Y_2 in base H means proving knowledge of $(w_1, w_2) \in \mathbb{Z}_p$ such that $Y_1 = w_1 G$ and $Y_2 = w_2 H$, and $w_1 = w_2$.

Using the above notation, we have $\varphi : \mathbb{Z}_p \to \mathbb{G}^2 : (x_1, x_2) \mapsto (x_1 G, x_2 H)$. The linear system of equations $A(w_1, w_2) = b$ is given by $w_1 - w_2 = 0$. The protocol flow is then as depicted in Protocol 3.

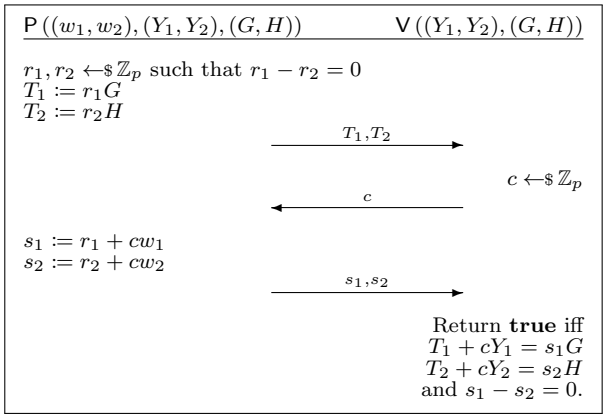
For a given challenge $c \in \mathbb{Z}_p$, the simulator chooses $s_1, s_2 \leftarrow\!\!\$\, \mathbb{Z}_p$ such that $s_1 - s_2 = 0$, and sets $T_1 \leftarrow s_1 G - cY_1$ and $T_2 \leftarrow s_2 H - cY_2$. It then outputs the simulated transcript $((T_1, T_2), c, (s_1, s_2))$.

Example 3 (DLEQ; alternative). The same proof goal as in the previous example can also be achieved by considering a slightly different homomorphism, which directly encodes the linear relation, that is $\varphi : \mathbb{Z}_p \to \mathbb{G}^2 : x \mapsto (xG, xH)$. The protocol flow is then as depicted in Protocol 4.

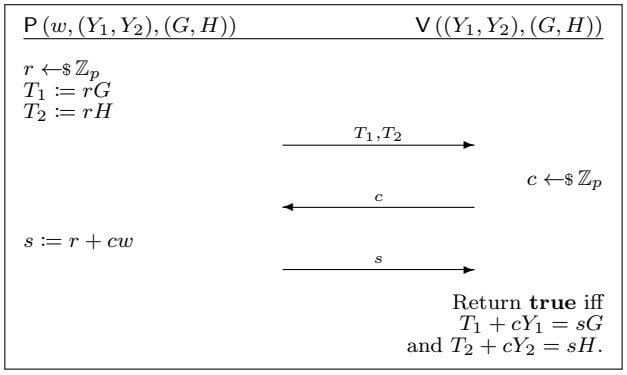
For a given challenge $c \in \mathbb{Z}_p$, the simulator chooses $s \leftarrow\!\!\$\, \mathbb{Z}_p$, and sets $T_1 \leftarrow sG - cY_1$ and $T_2 \leftarrow sH - cY_2$. It then outputs the simulated transcript $((T_1, T_2), c, s)$.

Remark 1. The two protocols illustrated prove the same statement, but achieve this via different approaches. The first protocol uses the more general framework of liner relations among witnesses introduced above. Alternatively, as shown in the second protocol, the linear relation can also be realized by directly incorporating the linear relation into the proof goal, thereby achieving a slightly shorter proof size (as the prover's last message consists of one element of $\mathbb{Z}_p$ less).

However, when proving inhomogeneous linear relations, incorporating the relation into the homomorphism also requires some additional computations in the target group, which in certain cases might compensate the advantage of a smaller proof size.



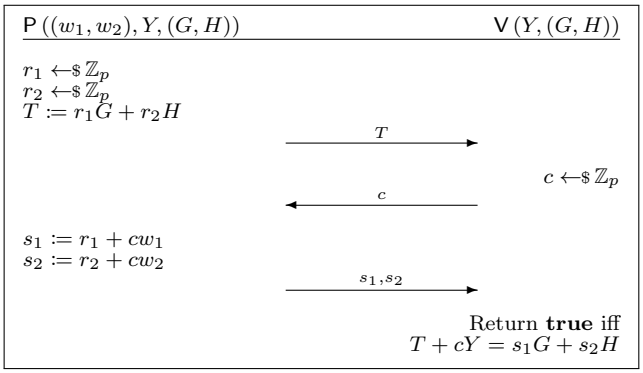
Protocol 3: Proving knowledge of equality of two discrete logarithms.



Protocol 4: Proving knowledge of equality of two discrete logarithms (alternative).

Example 4 (REP). Let $\mathbb{G}$ be a group over an elliptic curve of prime order p. Proving knowledge of a valid opening of a Pedersen commitment means proving knowledge of $w_1, w_2 \in \mathbb{Z}_p$ such that $Y = w_1G + w_2H$.

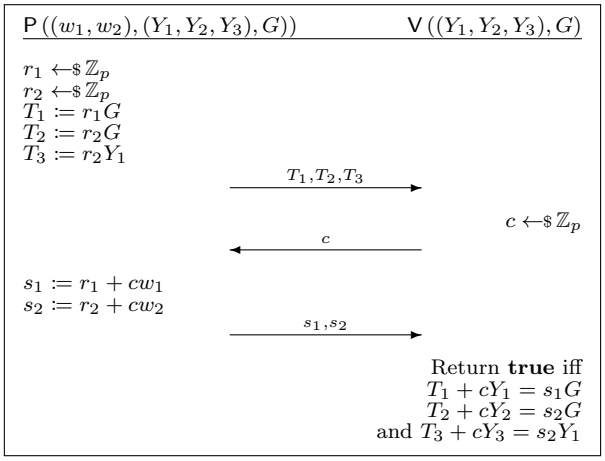
Using the above notation, we have $\varphi : \mathbb{Z}_p^2 \to \mathbb{G} : (x_1, x_2) \mapsto x_1G + x_2H$. The protocol flow is then as depicted in Protocol 5.



Protocol 5: Proving knowledge of representation.

For a given challenge $c \in \mathbb{Z}_p$, the simulator chooses $s_1, s_2 \leftarrow\$ \mathbb{Z}_p$, and sets $T \coloneqq s_1G + s_2H - cY$. It then outputs the simulated transcript $(T, c, (s_1, s_2))$.

Example 5 (DH). Let $\mathbb{G}$ be a group over an elliptic curve with prime order p. Proving knowledge of the exponents of a valid Diffie-Hellman triple means proving knowledge of $w_1, w_2 \in \mathbb{Z}_p$ such that $Y_1 = w_1G$, $Y_2 = w_2G$, and $Y_3 = w_1w_2G$. Yet, the mapping $\mathbb{Z}_p^2 \to \mathbb{G}^3 : (x_1, x_2) \mapsto (x_1G, x_2G, x_1x_2G)$ is not a homomorphism, and consequently the basic protocol presented before cannot be deployed directly. However, the re-



Protocol 6: Proving knowledge of representation.

quired multiplicative relation can be proven by observing that the proof goal is equivalent to $Y_1 = w_1 G$, $Y_2 = w_2 G$, and $Y_3 = w_2 Y_1$, leading the homomorphism $\varphi : \mathbb{Z}_p^2 \to \mathbb{G}^3 : (x_1, x_2) \mapsto (x_1 G, x_2 G, x_2 Y_1)$.

The protocol flow is then as depicted in Protocol 6.

For a given challenge $c \in \mathbb{Z}_p$, the simulator chooses $s_1, s_2 \leftarrow\!\$\, \mathbb{Z}_p$, and sets $T_1 \coloneqq s_1 G - cY_1$, $T_2 \coloneqq s_2 G - cY_2$, and $T_3 \coloneqq s_2 Y_1 - cY_3$. It then outputs the simulated transcript $((T_1, T_2, T_3), c, (s_1, s_2))$.

As shown in this example, and in contrast to linear relations, multiplicative relations among witnesses typically require a reformulation of the proof goal in order to be compatible with the generic protocol presented above. We refer, e.g., to Krenn [Kre12] for generic techniques.

4.2 Composition of Σ-Protocols

In this section, we recap composition techniques of Σ-protocols. Specifically, we define mechanisms for proving knowledge of multiple independent witnesses (*AND composition*), and for proving knowledge for one out of a set of witnesses (*OR composition*). Without loss of generality, the techniques presented in the following focus on the composition of two protocols; proving knowledge of more than two witnesses, or for out of a larger set of witnesses, can directly be achieved by iteratively deploying the techniques.

For the rest of this section, we let $(\mathsf{P}_1^0, \mathsf{P}_2^0, \mathsf{V}^0)$ and $(\mathsf{P}_1^1, \mathsf{P}_2^1, \mathsf{V}^1)$ be the specifications of two Σ-protocols for two relations R^0 and R^1, and let Sim^0 and Sim^1 be their simulators.

Furthermore, for the ease of presentation, we assume that all protocols to be composed are based on groups of the same prime order p. If this is not the case, the constructions can easily be adapted by setting the challenge set to the smaller of the two challenge sets.

AND Composition. In the following we explain how to construct a Σ-protocol proving knowledge of multiple independent witnesses. That is, the algorithms specified below constitute a Σ-protocol for the following relation:

$$\mathsf{R}^{\wedge} = \left\{ ((Y^0, Y^1), (w^0, w^1) : (Y^0, w^0) \in \mathsf{R}^0 \ \wedge \ (Y^1, w^1) \in \mathsf{R}^1 \right\} .$$

1. The prover's first algorithm $\mathsf{P}_1(\mathbf{w}, \mathbf{Y})$ consists of the following steps:
   (a) The algorithm parses $(w^0, w^1) \coloneqq \mathbf{w}$ and $(Y^0, Y^1) \coloneqq \mathbf{Y}$.
   (b) It computes $(T^0, \mathsf{st}^0) \leftarrow \mathsf{P}_1^0(w^0, Y^0)$ and $(T^1, \mathsf{st}^1) \leftarrow \mathsf{P}_1^1(w^1, Y^1)$.
   (c) The algorithm outputs $(\mathbf{T}, \mathsf{st}) \coloneqq ((T^0, T^1), (\mathsf{st}^0, \mathsf{st}^1))$.
2. The prover's second algorithm $\mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, \mathsf{st})$ proceeds as follows:
   (a) It checks that $c \in \mathbb{Z}_p$ and aborts if this is not the case.
   (b) It parses $(\mathsf{st}^0, \mathsf{st}^1) \coloneqq \mathsf{st}$
   (c) It computes $s^0 \leftarrow \mathsf{P}_2^0(w^0, Y^0, c, \mathsf{st}^0)$ and $s^1 \leftarrow \mathsf{P}_2^1(w^1, Y^1, c, \mathsf{st}^1)$.
   (d) It outputs $\mathbf{s} \coloneqq (s^0, s^1)$.
3. The verifier's algorithm $\mathsf{V}(\mathbf{Y}, \mathbf{T}, c, \mathbf{s})$ proceeds as follows:
   (a) It parses $(s^0, s^1) \coloneqq \mathbf{s}$.
   (b) The algorithm outputs $\mathsf{V}^0(Y^0, T^0, c, s^0) \wedge \mathsf{V}^1(Y^1, T^1, c, s^1)$.
4. The required simulator $\mathsf{Sim}(\mathbf{Y}, c)$ works as follows:
   (a) It parses $(Y^0, Y^1) \coloneqq \mathbf{Y}$.
   (b) It chooses $c \leftarrow\!\!\$\, \mathbb{Z}_p$.
   (c) It computes $(T^0, c, s^0) \leftarrow \mathsf{Sim}^0(Y^0, c)$ and $(T^1, c, s^1) \leftarrow \mathsf{Sim}^1(Y^1, c)$.
   (d) Finally, the algorithm then outputs $(\mathbf{T}, c, \mathbf{s}) \coloneqq ((T^0, T^1), c, (s^0, s^1))$.

OR Composition. In the following we explain how to construct a Σ-protocol proving knowledge of one out of a set of witnesses. That is, the algorithms specified below constitute a Σ-protocol for the following relation:

$$\mathsf{R}^{\vee} = \left\{ ((Y^0, Y^1), (w^0, w^1) : (Y^0, w^0) \in \mathsf{R}^0 \ \vee \ (Y^1, w^1) \in \mathsf{R}^1 \right\} .$$

In the following protocol specification, let j be such that w^j is known to the prover, whereas without loss of generality w^{1-j} is assumed to be unknown to the prover.

1. The prover's first algorithm $\mathsf{P}_1(\mathbf{w}, \mathbf{Y})$ consists of the following steps:
   (a) It parses $(w^0, w^1) \coloneqq \mathbf{w}$ and $(Y^0, Y^1) \coloneqq \mathbf{Y}$, where $w^{1-j} = \perp$.
   (b) It computes $(T^j, \mathsf{st}^j) \leftarrow \mathsf{P}_1^j(Y^j, w^j)$.
   (c) It computes a simulated transcript for the unknown witness by choosing $c^{1-j} \leftarrow_\$ \mathbb{Z}_p$ and setting $(T^{1-j}, c^{1-j}, s^{1-j}) \leftarrow \mathsf{Sim}^{1-j}(Y^{1-j}, c^{1-j})$.
   (d) The algorithm outputs $(\mathbf{T}, \mathsf{st}) \coloneqq ((T^0, T^1), (\mathsf{st}^j, c^{1-j}, s^{1-j}))$.
2. The prover's second algorithm $\mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, \mathsf{st})$ proceeds as follows:
   (a) It checks that $c \in \mathbb{Z}_p$ and aborts if this is not the case.
   (b) It parses $(\mathsf{st}^j, c^{1-j}, s^{1-j}) \coloneqq \mathsf{st}$.
   (c) It computes $c^j \coloneqq c - c^{1-j}$, and sets $s^j \leftarrow \mathsf{P}_2^j(w^j, Y^j, c^j, \mathsf{st}^j)$.
   (d) It computes $s^0 \leftarrow \mathsf{P}_2^0(w^0, Y^0, c, \mathsf{st}^0)$ and $s^1 \leftarrow \mathsf{P}_2^1(w^1, Y^1, c, \mathsf{st}^1)$.
   (e) It outputs $\mathbf{s} \coloneqq (s^0, s^1, c^0)$.
3. The verifier's algorithm $\mathsf{V}(\mathbf{Y}, \mathbf{T}, c, \mathbf{s})$ proceeds as follows:
   (a) It parses $(s^0, s^1, c^0) \coloneqq \mathbf{s}$.
   (b) It sets $c^1 \coloneqq c - c^0$.
   (c) The algorithm outputs $\mathsf{V}^0(Y^0, T^0, c^0, s^0) \wedge \mathsf{V}^1(Y^1, T^1, c^1, s^1)$.
4. The required simulator $\mathsf{Sim}(\mathbf{Y}, c)$ works as follows:
   (a) It parses $(Y^0, Y^1) \coloneqq \mathbf{Y}$.
   (b) It chooses a random c^0 in $\mathbb{Z}_p$ and computes $c^1 \coloneqq c - c^0$.
   (c) It then computes $(T^0, c^0, s^0) \leftarrow \mathsf{Sim}^0(T^0, c^0)$ and $(T^1, c^1, s^1) \leftarrow \mathsf{Sim}^1(Y^1, c^1)$.
   (d) Finally, the algorithm then outputs $(\mathbf{T}, c, \mathbf{s}) \coloneqq ((T^0, T^1), c, (s^0, s^1, c^0))$.

4.3 Achieving Non-Interactivity – The Fiat-Shamir Transform

All protocols described so far require three messages being exchanged between the prover and the verifier. However, communication rounds are often considered expensive from an efficiency point of view, and for many applications interactivity is not desirable.

Remark 2. The interactive version of Σ-protocols presented in this paper is unfit for practical applications. In this submission, we focus on the non-interactive transform.

We describe two ways for achieving non-interactivity. Both variants require identical computations on the prover's side. The first allows for efficient batching, but requires point validation and (depending on the hash function, or the elliptic curve chosen) might lead to slightly larger proof sizes. The second requires no point validation and only has the hash value. We call the first variant *batchable*, and the latter *short*. Both constructions are based on the seminal work of Fiat and Shamir [FS87] and

subsequent work, e.g., by Bernhard et al. [BPW12]. The underlying idea of the so-called Fiat-Shamir transform is to simulate the verifier's random challenge by means of a random oracle, depending on the first message computed by the prover. More precisely, the challenge is computed as $c \coloneqq \mathsf{H}(\mathbf{Y}, \mathbf{T}, \mathsf{ctx}, \tau)$, where Y is the public image for which knowledge of a witness is proven, T is the prover's first message from the Σ-protocol, τ is a (possibly empty, variable-length) label attached to the proof, and the context string ctx is a tuple $\mathsf{ctx} = (\mathsf{domsep}, \mathsf{curve}, \mathsf{gens}, \mathsf{id})$ containing the following application-specific information:

- domsep, a string determining uniquely the name of the target application (and the proof type);
- curve, a label identifying uniquely the group (e.g. `secp-256`)
- gens, a full description of the algebraic setting and proof goal (e.g., group generators, and constants fixed in the protocol) to achieve non-malleability;
- id, local context information (e.g., session identifiers of higher level protocols) to avoid replay-attacks, or shared randomness or a timestamp to guarantee freshness of the proof.

The tag τ contains an arbitrary string, possibly empty, that the prover can attach to the proof. This can allow to construct e.g. signatures and ring signatures. We discuss more in details the content the hash in Section 6.

Batchable form

1. The prover's algorithm $\mathsf{P}_{\text{batch}}(\mathbf{w}, \mathbf{Y}, \mathsf{ctx}, \tau)$ works as follows:
   (a) The algorithm first computes $(\mathbf{T}, \mathsf{st}) \leftarrow \mathsf{P}_1(\mathbf{w}, \mathbf{Y})$.
   (b) It computes the challenge by setting $c \coloneqq \mathsf{H}(\mathbf{T}, \mathbf{Y}, \mathsf{ctx}, \tau)$.
   (c) The algorithm defines $\mathbf{s} \leftarrow \mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, \mathsf{st})$.
   (d) The algorithm outputs $(\mathbf{T}, \mathbf{s})$.
2. The verifier's algorithm $\mathsf{V}_{\text{batch}}(\mathbf{Y}, \mathsf{ctx}, \tau, (\mathbf{T}, \mathbf{s}))$ works as follows:
   (a) It recomputes the challenge as $c \leftarrow \mathsf{H}(\mathbf{Y}, \mathbf{T}, \mathsf{ctx}, \tau)$.
   (b) It outputs whatever $\mathsf{V}(\mathbf{Y}, \mathbf{T}, c, \mathbf{s})$ outputs.

Verification equation. The batchable form can take advantage of the following fact. Given the single verification equations of the form:

$$T_i + c_i Y_i = \sum_j s_{i,j} G_j$$

for $i = 0, \ldots, \ell - 1$, the verifier can sample a random vector of coefficients $a \in \mathbb{Z}_p^\ell$ instead test that:

$$\left(\sum_{i=0}^{\ell-1} a_i T_i\right) + \left(\sum_{i=0}^{\ell-1} a_i \cdot c_i Y_i\right) = \left(\sum_{i=0}^{\ell-1} a_i \cdot s_i G_i\right).$$

If the matrix $G_{i,j}$ of generators has identical rows, by linearity the right-hand side can be computed as a single scalar product. If the statements Y_i's have identical rows, by linearity the second term in the equation can be computed as a single scalar product. The random vector $\mathbf{a}$ can be *deterministically* generated by fixing $a_0 := 1$ and setting $(a_1, \ldots, a_{\ell-1}) := \mathsf{PRG}(\mathbf{Y}, \mathbf{T}, c, \mathbf{s})$ [WNR18].

Short form

1. The prover's algorithm $\mathsf{P}_{\text{short}}(\mathbf{w}, \mathbf{Y}, \mathsf{ctx}, \tau)$ works as follows:
   (a) The algorithm first computes $(\mathbf{T}, \mathsf{st}) \leftarrow \mathsf{P}_1(\mathbf{w}, \mathbf{Y})$.
   (b) It computes the challenge by setting $c := \mathsf{H}(\mathbf{T}, \mathbf{Y}, \mathsf{ctx}, \tau)$.
   (c) The algorithm defines $\mathbf{s} \leftarrow \mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, \mathsf{st})$.
   (d) The algorithm outputs $(c, \mathbf{s})$.
2. The verifier's algorithm $\mathsf{V}_{\text{short}}(\mathbf{Y}, \mathsf{ctx}, \tau, (c, \mathbf{s}))$ works as follows:
   (a) The algorithm recomputes the first message $\mathbf{T}$ by running the appropriate steps of the simulator Sim.
   More precisely, we stress that, on input a challenge c, all simulators introduced so far first draw a response s, and then compute the commitment T from c and s. Instead of sampling s, the algorithm now uses its input value to recompute T according to the remaining steps of the simulator.
   (b) The algorithm checks that $c = \mathsf{H}(\mathbf{T}, \mathbf{Y}, \mathsf{ctx}, \tau)$ and rejects if this is not the case.

4.4 Achieving Concurrent Zero-Knowledge

The basic protocols presented above cannot directly be used for practical applications, as the zero-knowledge property does not hold against malicious verifiers. While for most applications the Fiat-Shamir heuristic is the preferred way to also address this challenge, there might be cases where interactive protocols are still preferable, e.g., for deniability reasons. The following construction, due to Damgård [Dam00], allows one to turn any Σ-protocol into a concurrently zero-knowledge version also against malicious adversaries.

In the following, $(\mathsf{Commit}, \mathsf{VerCommit})$ is a commitment scheme, where the interfaces are as specified in Benarroch et al. [BCF19, Section 4.3]. The commitment scheme used in the following is required to be a trapdoor commitment scheme (cf., e.g., Fischlin [Fis01] for an in-depth discussion of such schemes), with a natural candidate being the scheme introduced by Pedersen [Ped92].

1. The verifier's first algorithm $\mathsf{V}_1(\mathbf{Y})$ proceeds as follows:
   (a) It samples $c \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes $(com, o) \coloneqq \mathsf{Commit}(ck, c)$.
   (b) It sets $\mathsf{st}_\mathsf{V} \coloneqq (c, o)$.
   (c) It sends com to the prover.
2. The prover's first algorithm $\mathsf{P}_1(\mathbf{w}, \mathbf{Y}, com)$ consists of the following steps:
   (a) The algorithm computes $(\mathbf{T}', \mathsf{st}') \coloneqq \mathsf{P}_1(\mathbf{w}, \mathbf{Y})$.
   (b) It outputs $(\mathbf{T}, \mathsf{st}) \coloneqq (\mathbf{T}', (\mathsf{st}', com))$.
3. The verifier's second algorithm $\mathsf{V}_1(\mathbf{Y}, \mathbf{T}, \mathsf{st}_\mathsf{V})$ proceeds as follows:
   (a) It parses $\mathsf{st}_\mathsf{V} = (c, o)$.
   (b) It sends (c, o) to the prover.
4. The prover's second algorithm $\mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, o, \mathsf{st})$ proceeds as follows:
   (a) The algorithm parses $\mathsf{st} = (\mathsf{st}', com)$.
   (b) It checks that $\mathsf{VerCommit}(com, c, o) = \mathbf{true}$ and aborts otherwise.
   (c) It outputs $s \coloneqq \mathsf{P}_2(\mathbf{w}, \mathbf{Y}, c, \mathsf{st}')$.
5. The verifier's final algorithm $\mathsf{V}_3(\mathbf{Y}, \mathbf{T}, c, s)$ proceeds as follow:
   (a) The algorithm outputs $\mathsf{V}(\mathbf{Y}, \mathbf{T}, c, s)$.

5 Security Considerations

In the following, we give a concise overview of the most important security guarantees provided by the constructions presented above. For full details and proofs, we refer to the original literature.

Σ-Protocols. The protocol presented in Section 4.1 is a Σ-protocol, if $\mathbb{G}$ is a cyclic group of prime order p, and $\varphi : \mathbb{Z}_p^n \to \mathbb{G}^m$ is a group homomorphism.
If φ is non-trivial (there exists $\mathbf{x} \in \mathbb{Z}_p^n$ such that $\varphi(\mathbf{x}) \neq (1, \ldots, 1) \in \mathbb{G}^m$), the protocol has unpredictable commitments. Finally, if φ is an injective function, the protocol has unique responses.
For details we refer, e.g., to [Cra97].

Composition. The composed protocols presented in Section 4.2 are Σ-protocols if the individual protocols are Σ-protocols, see, e.g., [CDS94].

If at least one of the composed protocols has unpredictable commitments, then so have the protocols for $\mathsf{R}^{\wedge}$ and $\mathsf{R}^{\vee}$.

If both protocols have unique responses, then so has the protocol for $\mathsf{R}^{\wedge}$. Note that unique responses are not preserved under OR-compositions.

Proof of Knowledge. All protocols presented in Sections 4.1 and 4.2 are proofs of knowledge, see Damgård [Dam04]. The non-interactive protocol (cf. Section 4.3) is secure against unbounded malicious provers, provided that they make a polynomial number of queries to the random oracle [PS00]. Existence of a straight-line extractor for non-interactive Σ-protocols was proposed by Fuchsbauer et al. [FKL18], but under the (stronger) setting of the algebraic group model.

Concurrent Zero-Knowledge. The protocol in Section 4.4 is concurrently zero-knowledge against arbitrary verifiers if the used commitment scheme is a trapdoor commitment scheme, cf. [Dam00].

Completeness. For all the Σ-protocols presented above, the Fiat-Shamir transform is complete.

For a proof we refer to Unruh [Unr17], which uses the fact that all presented protocols are complete and have unpredictable commitments; Unruh also gives a surprising counter-example in case that commitments are not unpredictable.

Zero-Knowledge. For all Σ-protocol presented above, the Fiat-Shamir transform yields a zero-knowledge protocol in the random oracle model.

For a proof we refer to Faust et al. [FKMV12] and Unruh [Unr17]. These security guarantees also hold against a quantum attacker, and rely on the honest-verifier zero-knowledge property of Σ-protocols and the fact that the commitments in all our protocols are unpredictable.

Soundness. If a Σ-protocol has a negligible soundness error, then the Fiat-Shamir transform is sound according to Definition 3. This security guarantee also holds against quantum attackers.

For a proof we refer to Pointcheval and Stern [PS00] and Unruh [Unr17].

Simulation soundness. In all protocols presented in Sections 4.1 and 4.2, the Fiat-Shamir transform is weakly simulation-sound. If the protocol additionally has unique responses, then the Fiat-Shamir transform is strongly simulation-sound. These security guarantees also holds against quantum attackers.

For a proof we refer to Unruh [Unr17] and Faust et al. [FKMV12].

Table 1. Summary of open-source projects that implement Σ-protocols, along with features supported. INT stands for *interactive*; FS stands for *Fiat-Shamir*. Most of this table's data comes from the work of Lueks et al. [LKF+19].

Project	Language	AND-composition	OR-composition	INT	FS
`Cashlib` [MEK+10]	C++	✓	✗	✗	✓
`Emmy` [XLA]	Go	✗	✗	✓	✗
`Kyber` [DED]	Go	✓	✓	✓	✓
`SCAPI` [EFLL12]	C++	✓	✓	✓	✓
`YAZKC` [ABB+10,ABB+12]	C	✓	✓	✓	✓
`zkp` [dV19]	Rust	✓	✗	✗	✓
`zksk` [LKF+19]	Python	✓	✓	✓	✓

Simulation extractability. For all protocols presented in Sections 4.1 and 4.2, the non-interactive Fiat-Shamir transform is simulation extractable [BPW12, Thm. 1].

6 Instantiation

In this section, we consider the concrete security of Σ-protocols and illustrate the main points that should be considered when implementing them. In Table 1, we report the implementations that we are currently aware of. This table is greatly inspired from the remarkable bibliographic work of Lueks et al. [LKF+19].

Choice of a group. We advise for the use of prime-order elliptic curves of size either 256 or 512 bits, depending on the desired security of the upper layers in the protocol[5]. Concretely, we see fit NIST's `p-521` and `p-256` [NIS00]; SECG's `secp256k1` [SEC00], or prime-order group abstractions over non prime-order groups, such as as Decaf [Ham15] and Ristretto [dVGT+20][6]. We note that there exists already standards for removing small cofactors in elliptic curves [Zuc00]. However we believe that the complexity added by integrating the handling of the cofactor

[5] For instance, proving a DH relation with one fixed group element such as a public key, might expose the protocol to cryptanalytic attacks such as Brown-Gallant [BG04] and Cheon's attack [Che06], and some implementations might opt for larger curve sizes. We consider these attacks out of scope for this standardization effort, and believe this analysis should be deferred to the concrete security study of the larger protocol.

[6] We are aware of the threats reported in `https://safecurves.cr.yp.to/`. It's however not clear what should be the alternatives, and if dropping their support would hinder adoption.

within the verification equations is deleterious for the other supported curves.

We think pairing-friendly curves, such as BLS12-381 [Bow17], should also be considered for inclusion under relevant assumptions for bilinear groups, such as XDH when dealing with DH triples.

Choice of the hash functions. We advise for the use of consolidated, cryptographically secure hash functions such as `SHA2`, `SHA3`, and `BLAKE2`, `BLAKE3`[7]. In the non-interactive Σ-protocols, the image of the random oracle is assumed to be over the field, and translating the output of the hash function (in $\{0,1\}^*$) into an integer mod p requires care to preserve indifferentiability. To the best of our knowledge, the best options for performing this operation are the following:

(a) Truncating the output of the hash to 128 bits, and interpreting it as a small integer mod p. Despite being the most efficient option at our disposal, it is unclear if the collision probability of 2^{-64} could be too small for certain applications;

(b) Truncating the output of the hash to 256 bits, and then reducing mod p. This is by far the most natural approach; however, the output distribution could be potentially far from uniformly random, in terms of statistical distance. Consider e.g. a prime p that is close to $3/4 \cdot 2^{256}$. Reducing a random 256-bit integer modulo this p yields a value that is in the range $[0, p/3]$ with probability roughly $1/2$ [FHSS+20]. We remark that, nonetheless, min-entropy wise this will cost at most one bit of security. Namely, the chances of guessing a specific challenge is still than negligible despite the bias.

(c) Truncating the output of the hash to $\lceil \log_2 p \rceil + \kappa$ bits (where κ is the targeted statistical security level), and then reducing mod p. This is essentially the procedure `hash_to_field` as described in [FHSS+20, Section 5]. This approach leads to a random value statistically close ($2^{-\kappa}$) to a value sampled uniformly at random in $\mathbb{Z}_p$.

(d) Rejection sampling, which consists in sampling uniformly at random in the interval $[0, 2^\lambda]$, until a value in $[0, p]$ is found. Unfortunately, since the hash function is used also for hashing the initial (secret) randomness used for computing the commitment, this method is undesirable due to the difficulty of implementing in constant-time (which

[7] We did not examine closely algebraic hashes for this draft, but we stress that their adoption could be appealing for a number of applications requiring recursive proofs. Unfortunately, this requires a much deeper cryptanalysis within a working group.

could lead a secret for the deterministic computation of the commitment).

Despite options Items (b) and (c) are the most fit for computing the challenge, only option Item (c) is suitable for computing deterministically also the initial nonce. in the non-interactive case. Item (a) is a valid option for the interactive version.

Hashing multiple variable-length messages into a single challenge (such is the case when hashing ctx, τ) requires care, and multiple approaches have been used in the past. We see three options available:

(a) Replacing variable-length messages with their hash (i.e. hashing multiple values à la *Merkle-Damgard*) and computing:

$$\mathsf{ctx} := (\mathsf{H}(\mathsf{domsep}), \mathsf{curve}, \mathsf{gens}, \mathsf{H}(\mathsf{id}))$$

this is currently what proposed in [WNR18] for the specific case of the domain separator (in fact, the authors propose to concatenate the same hash twice to fill an entire SHA-256 block and allow for hash pre-processing).

(b) Encoding the messages in a common format, and hashing the encoded message, i.e. computing: $\mathsf{ctx} := \mathsf{encode}(\mathsf{ctx})$, where encode is a serializer such as e.g. JSON encoding, or netstring [Ber98].

(c) Using a protocol framework such as Strobe [Ham17] that builds a sponge construction and can be used for building protocol transcripts. For more information, we direct the reader towards [Ham17, Section 5.3].

Commitment. The first step is the generation of uniformly-distributed random element(s) over $\mathbb{Z}_p$. As mentioned in Section 2, the nonce must be distributed uniformly at random, and even a small bias in the distribution could completely compromise zero-knowledge [HGS01,Ble00,ANT+20].

We propose the construction of a synthetic nonce obtained from hashing statement, context, and secret, together with 8 additional bytes from operating system's entropy:

$$r := \mathsf{H}(\mathsf{ctx}, \tau, w, \mathsf{rnd})$$

where $\mathsf{rnd} := \{0,1\}^{64}$ are 64 bits of entropy. The construction of cryptographically secure source of randomness is a difficult problem, that is particularly challenging on embedded devices such as smart-cards or embedded systems. For those applications for which obtaining a high-quality

entropy is challenging, a stateful counter would be sufficient to achieve random r.

Hashing the statement and the witness to obtain a commitment was already suggested in previous standards [Por13] in the context of deterministic nonce generation. While it is widely recognized that having deterministic nonce helps strengthening the concrete security and mitigates the risks associated to the generation of a commitment, it could on the other hand leak information about the witness used for a proof, when examining consecutive executions of the protocol. This is e.g. the case of a OR-proof that attempts to preserve anonymity. Consider a ring signature for the ring $R = (Y_0, Y_1)$, and two OR-proofs with the same tag τ. In a scenario where the nonce is deterministically generated, then the proofs are identical if the same key is used, and different otherwise.

Computing the challenge. The challenge is computed as:

$$c := \mathsf{H}(\mathbf{Y}, \mathbf{T}, \mathsf{ctx}, \tau)$$

according to the prescriptions of the previous paragraph and section Section 4.3. The implementer can pre-process the hash evaluation of the context ctx and the statement $\mathbf{Y}$.

Response and proof output. The response is computed using standard field addition and multiplication. The implementation should support batched and short form, as wrappers around the proof transcript as generated by P_2.

Verification. The implementation should support two different verification functions, for batched and short verification. The case of batched verification must include a point verification sub-routing that asserts the statement and commitments are in question. Failure to properly check that a commitment is in the group could lead to subgroup attacks [vW96,LL97] or invalid curve attacks [BMM00,BBPV12].

If verification fails, an exception should be raised. If input parsing fails, an exception should be raised. Otherwise, the verifier outputs **true**. Optionally, the implementation can choose to return the parsed statement.

7 Looking Ahead

To keep this proposal short and open the discussion to a wider community, we purposely left open some topics that could be interesting for more tight use cases.

Delayed input. Despite throughout this work we have assumed that the prover receives as input instance and witness already when computing the first message, in some Σ-protocols (c.f. Protocols 2, 3 and 4) knowledge of instance and witness is required only in the last round. Therefore, the first two messages can be pre-processed obtaining better practical efficiency. Obviously, this feature relevant only for interactive protocols, which might include scenarios requiring deniability or of unconditional soundness.

Unfortunately, the OR composition discussed in Section 4.2 affects the possible delayed-input property of the underlying Σ-protocols. Consider for instance the simple OR composition allowing to prove knowledge of at least one out of two discrete logarithms. Even though the underlying two Σ-protocols satisfy the delayed-property, the compound Σ-protocol obtained through [CDS94] forces knowledge of the instances already before computing the first message. Alternative information-theoretic OR compositions have been proposed in [CPS+16a], for practical classes of Σ-protocols requiring that only one out of two instances is known when the protocol starts and obtaining a compound delayed-input Σ-protocol. Relying on computational (i.e., DDH) assumptions, in [CPS+16b] it is shown how to generically (i.e., not just two and no further restrictions) compose delayed-input Σ-protocols obtaining a compound delayed-input Σ-protocol.

Designated verifier proofs. A designated-verifier zero-knowledge proof of knowledge allows the prover to determine an intended receiver for the proof in a way that guarantees that only the specified receiver can actually obtain any conviction about the validity of the prover's claim. This is important, e.g., in cases where deniability of the communication is important for privacy reasons, and the prover needs to be ensured that the verifier cannot credibly forward the proof to any third party, as he could have simulated proofs with the same distribution himself. Designated-verifier proofs have been studied profoundly in the academic literature [JSI96,CC18]. The most widely known approach is due to Jakobsson et al. [JSI96], which achieves the required functionality basically by proving that "one either knows the secret key, or one is the designated verifier". In a nutshell, the idea is that the designated verifier provides the prover with a public key Y_{pk}, and the verifier then proves that he knows a witness w for the claimed statement Y for the given relation R, or the secret key w_{pk} corresponding to Y_{pk}, using the composition techniques from Section 4.2.

More formally, let R' be such that $(Y_{pk}, w_{pk}) \in \mathsf{R}'$ if and only if w_{pk} is a secret key corresponding to Y_{pk}, e.g., $\mathsf{R}' = \{(Y_{pk}, w_{sk}) : Y = wG\}$. Then, the prover computes a proof for the following relation:

$$\mathsf{R}^{DS} = \{(Y, Y_{pk}), (w, w_{pk}) : (Y, w) \in \mathsf{R} \vee (Y_{pk}, w_{pk}) \in \mathsf{R}'\}\,.$$

It can now be seen that the prover is now unable to forward the proof to a third party, as he could have generated the proof himself using the secret key. It it worth noting that it is of crucial importance that the prover's public key comes together with a proof that the prover indeed knows the secret key: otherwise, by provable sampling the public key at random (e.g., as the hash value of some public string), the prover could credibly claim that he does not know the corresponding key, and thus could not have generated the transcript himself.

Shared proof computation. Splitting the witness across multiple devices is very appealing from an applied security perspective. Recently, we saw a number of protocols based on the same template of of Protocol 1 [KG20,NRS20], especially targeting applications in signatures for cryptocurrencies. Nothing prevents those same techniques to be adopted in the (more general) topic of Σ-protocols, and we believe the community should have a discussion if interactive proof generation should be supported.

R1CS compatibility. It is not clear whether it would be beneficial for an API of Σ-protocols to be aligned with the current zk-proof effort of having a uniform language for expressing relations (c.f. the community reference [ZKP19, section 3], and Drevon's proposal [Dre19]). Generally, statements about discrete logarithms are dealt with so-called Camenisch–Stadler notation, and we believe community should come to agree on the acceptable trade-offs for expressing the relation in R1CS language.

Acknowledgments

The authors thank Jan Bobolz, Mary Maller, and Ivan Visconti for their precious reviews and comments during the early stage of this work. This work has partially been funded by the European Union's Horizon 2020 framework programme under grant agreement no. 830929 (CyberSec4Europe).

References

ABB+10. José Bacelar Almeida, Endre Bangerter, Manuel Barbosa, Stephan Krenn, Ahmad-Reza Sadeghi, and Thomas Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on sigma-protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 151–167. Springer, Heidelberg, September 2010.

ABB+12. José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 488–500. ACM Press, October 2012.

AG19. Riham AlTawy and Guang Gong. Mesh: A supply chain solution with locally private blockchain transactions. *PoPETs*, 2019(3):149–169, July 2019.

ANT+20. Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. LadderLeak: Breaking ECDSA with less than one bit of nonce leakage. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 225–242. ACM Press, November 2020.

Ban05. Endre Bangerter. *Efficient Zero-Knowledge Proofs of Knowledgefor Homomorphisms*. PhD thesis, Ruhr-Universität Bochum, Germany, 2005.

BBPV12. Billy Bob Brumley, Manuel Barbosa, Dan Page, and Frederik Vercauteren. Practical realisation and elimination of an ECC-related software bug attack. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 171–186. Springer, Heidelberg, February / March 2012.

BCF19. Daniel Benarroch, Matteo Campanelli, and Dario Fiore. Community Standards Proposal for Commit-and-Prove Zero-Knowledge Proof Systems. ZKProof Community Standard Proposal, available at https://github.com/zkpstandard/zkreference/tree/master/standards-proposals, 2019. accessed on February 22, 2021.

BDL+12. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012.

Ber98. D. J. Bernstein. Netstrings. Internet-Draft draft-bernstein-netstrings-02, Internet Engineering Task Force, August 1998. Work in Progress.

Ber06. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

BG93. Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420. Springer, Heidelberg, August 1993.

BG04. Daniel R. L. Brown and Robert P. Gallant. The static Diffie-Hellman problem. Cryptology ePrint Archive, Report 2004/306, 2004. http://eprint.iacr.org/2004/306.

Ble00. Daniel Bleichenbacher. On the generation of one-time keys in dl signature schemes. In *Presentation at IEEE P1363 working group meeting*, page 81, 2000.

BMM00. Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 131–146. Springer, Heidelberg, August 2000.

Bow17. Sean Bowe. Bls12-381: New zk-snark elliptic curve construction, 2017. Available at: https://electriccoin.co/blog/new-snark-curve/.

BPW12. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, December 2012.

CC18. Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 193–221. Springer, Heidelberg, April / May 2018.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

CGY20. Véronique Cortier, Pierrick Gaudry, and Quentin Yang. How to fake zero-knowledge proofs, again. In *E-Vote-Id 2020-The International Conference for Electronic Voting*, 2020.

Che06. Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 1–11. Springer, Heidelberg, May / June 2006.

CMZ14. Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1205–1216. ACM Press, November 2014.

CPS+16a. Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved OR-composition of sigma-protocols. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 112–141. Springer, Heidelberg, January 2016.

CPS+16b. Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 63–92. Springer, Heidelberg, May 2016.

CPZ20. Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 1445–1459. ACM Press, November 2020.

Cra97. Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI Amsterdam, The Netherlands, 1997.

CZJ+17. Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed E. Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via PVORM. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 701–717. ACM Press, October / November 2017.

Dam00. Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Heidelberg, May 2000.

Dam04. Ivan Damgård. On Σ-Protocols. Lecture on Crptologic Protocol Theory; Faculty of Science, University of Aarhus, 2004.

DED. DEDIS. [n.d.] kyber - dedis advanced crypto library for go. Available at: https://pkg.go.dev/go.dedis.ch/kyber.

DGS+18. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

Dre19. Guillaume Drevon. J-r1cs – a json lines format for r1cs. zkproof Proposal, 2019. Available at: https://docs.zkproof.org/pages/standards/accepted-workshop2/proposal--zk-interop-jr1cs.pdf.

DSW19. Alex Davidson, Nick Sullivan, and Christopher A. Wood. Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups. Internet-Draft draft-sullivan-cfrg-voprf-03, Internet Engineering Task Force, March 2019. Work in Progress.

dV19. Henry de Valence. zkp: a toolkit for schnorr proofs, 2019. Available at: https://medium.com/@hdevalence/zkp-a-toolkit-for-schnorr-proofs-6e381b4f0a31.

dVGT+20. Henry de Valence, Jack Grigg, George Tankersley, Filippo Valsorda, isis lovecruft, and Mike Hamburg. The ristretto255 and decaf448 Groups. Internet-Draft draft-irtf-cfrg-ristretto255-decaf448-00, Internet Engineering Task Force, October 2020. Work in Progress.

EFLL12. Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi: The secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012. https://eprint.iacr.org/2012/629.

FFS87. Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In Alfred Aho, editor, *19th ACM STOC*, pages 210–217. ACM Press, May 1987.

FHSS+20. Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-10, Internet Engineering Task Force, October 2020. Work in Progress.

Fis01. Marc Fischlin. *Trapdoor Commitment Schemes and Their Applications*. PhD thesis, Johann Wolfgang Goethe-Universität, Germany, 2001.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

Ham15. Mike Hamburg. Decaf: Eliminating cofactors through point compression. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 705–723. Springer, Heidelberg, August 2015.

Ham17. Mike Hamburg. The STROBE protocol framework. Cryptology ePrint Archive, Report 2017/003, 2017. http://eprint.iacr.org/2017/003.

Hao17. Feng Hao. Schnorr Non-interactive Zero-Knowledge Proof. RFC 8235, September 2017.

HGS01. Nick A Howgrave-Graham and Nigel P. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.

HLPT20. Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy*, pages 644–660. IEEE Computer Society Press, May 2020.

HR11. Feng Hao and Peter Y. A. Ryan. Password authenticated key exchange by juggling. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols XVI*, pages 159–171, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

JKK14. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Heidelberg, December 2014.

JL17. Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, January 2017.

JSI96. Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 143–154. Springer, Heidelberg, May 1996.

KG20. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852, 2020. https://eprint.iacr.org/2020/852.

KKR19. Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. Cryptography for #MeToo. *PoPETs*, 2019(3):409–429, July 2019.

KLI+18. Bogdan Kulynych, Wouter Lueks, Marios Isaakidis, George Danezis, and Carmela Troncoso. Claimchain. *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, Jan 2018.

Kre12. Stephan Krenn. *Bringing Zero-Knolwedge Proofs of Knowledge to Practice*. PhD thesis, University of Fribourg, Switzerland, 2012.

LKF+19. Wouter Lueks, Bogdan Kulynych, Jules Fasquelle, Simon Le Bail-Collet, and Carmela Troncoso. zksk. *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society - WPES'19*, 2019.

LL97. Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 249–263. Springer, Heidelberg, August 1997.

Mau09. Ueli M. Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 272–286. Springer, Heidelberg, June 2009.

Mau15. Ueli Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *Des. Codes Cryptogr.*, 77(2-3):663–676, 2015.

MEK+10. Sarah Meiklejohn, C. Christopher Erway, Alptekin Küpçü, Theodora Hinkle, and Anna Lysyanskaya. ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security 2010*, pages 193–206. USENIX Association, August 2010.

MP15. Gregory Maxwell and Andrew Poelstra. Borromean Signatures, 2015. Available at https://raw.githubusercontent.com/Blockstream/borromean_paper/master/borromean_draft_0.01_34241bb.pdfx.

NIS00. NIST. Digital signature standard. FIPS 186-2, 2000. Available at: https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf.

Noe15. Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

NRS20. Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round schnorr multi-signatures. Cryptology ePrint Archive, Report 2020/1261, 2020. https://eprint.iacr.org/2020/1261.

NRSW20. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 1717–1731. ACM Press, November 2020.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

Por13. Thomas Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979, August 2013.

PS97. David Pointcheval and Jacques Stern. New blind signatures equivalent to factorization (extended abstract). In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *ACM CCS 97*, pages 92–99. ACM Press, April 1997.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

SEC00. Certicom research, standards for efficient cryptography group (SECG) — sec 1: Elliptic curve cryptography. http://www.secg.org/secg_docs.htm, September 20, 2000. Version 1.0.

tt17. `luigi1111` and `fluffypony`. Disclosure of a major bug in cryptonote based currencies, 2017. Available at: https://www.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html.

Unr17. Dominique Unruh. Post-quantum security of Fiat-Shamir. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 65–95. Springer, Heidelberg, December 2017.

vW96. Paul C. van Oorschot and Michael J. Wiener. On Diffie-Hellman key agreement with short exponents. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 332–343. Springer, Heidelberg, May 1996.

WNR18. Pieter Wuille, Jonasonas Nick, and Tim Ruffing. Bip 0340, 2018. Available at: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-July/016203.html.

XLA. XLAB. [n.d.] emmy - library for zero-knowledge proofs. Available at: https://github.com/xlab-si/emmy. Last accessed: July 9, 2019.

ZKP19. ZKProof. ZKProof Community Reference v0.2. Technical report, 2019. accessed on February 8, 2021.

Zuc00. Robert Zuccherato. Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME. RFC 2785, March 2000.