SPoCK: Sequential Proofs of Complete Knowledge

Antonio Giulio D'Antona[1][0009-0007-4288-0244], Charlotte Hoffmann*[,2][0000-0003-2027-5549], and Krzysztof Pietrzak[3][0000-0002-9139-1654]

[1] ETH Zurich
gdantona@student.ethz.ch
[2] Zama, Paris, France
charlotte.hoffmann@zama.ai
[3] Institute of Science and Technology Austria
pietrzak@ista.ac.at

Abstract. A proof of knowledge certifies that the prover "knows" a secret. This property is established by the existence of an extractor that, given access to the prover, can extract the secret. Unfortunately, this does not imply that a single entity has access to the secret in the clear, as it may be secret-shared among several parties and the proof computed using multiparty computation (MPC), or the secret may be embedded in a trusted execution environment (TEE) that only allows limited access to it.

Often, the ability to encumber a secret in this way breaks the security of a system; vote selling in e-voting schemes is one example. To address this, two recent papers introduced the notions of "Proofs of Complete Knowledge" (PoCK) [Kelkar et al., CCS'24] and "Individual Cryptography" [Dziembowski et al., CRYPTO'23]. While the goals and constructions in those works differ, both rely on the same key idea to prevent encumbrance. To compute a proof, the prover must evaluate a hash function on a huge number of inputs. One then assumes that only a fraction of those can be encumbered, while for the rest the prover must know the inputs in the clear, and these clear inputs are sufficient to extract the secret. Computing a huge number of hashes is a challenge even for an honest prover (who has the secret in the clear), so those works suggest using outdated Bitcoin mining hardware to make the scheme feasible.

By forcing the prover to evaluate the hashes in a *sequential* manner, we can obtain meaningful security against encumbrance with far fewer hashes, especially against MPC, where round complexity is a major bottleneck.

As a concrete instantiation of this idea, we define and construct **Sequential** Proofs of Complete Knowledge (**S**PoCK). Our construction uses the PoCK of Kelkar et al. (which is based on Fischlin's straight-line

* Most of this work was done while the author was at Institute of Science and Technology Austria.

extractable proofs of knowledge). The computation of this PoCK is then embedded into the computation of the Proof of Sequential Work from [Cohen&Pietrzak, EUROCRYPT'18]. SPoCK thus has the potential to enable complete-knowledge or individual-cryptography primitives where honest parties can use standard hardware. We also propose a variant of this scheme that requires a large amount of memory throughout the evaluation, providing better security also against TEEs.

1 Introduction

Shortly after zero-knowledge proofs [16], which allow one to prove that some NP statement is true, zero-knowledge proofs *of knowledge* (ZK-PoK) [15] were introduced. In these protocols, the proof also certifies that the prover knows an actual witness; that is, for some NP relation $\mathcal{R}_L$ for a language L, given $x \in L$, the proof demonstrates knowledge of a w such that $(x, w) \in \mathcal{R}_L$. To prove "knowledge", one must provide an extractor Ext that can extract a witness from any, potentially malicious, prover who can make the verifier accept with sufficiently high probability.

ZK-PoK are interesting even for languages where membership in L is trivial. For example, consider the discrete logarithm relation

$$((g, h), w) \in \mathcal{R}_{\mathrm{DL}} \iff g^w = h$$

in a prime-order group. Here every pair (g, h) of group elements (with g not the identity) is in the language, but finding w can be hard.

ZK-PoK have found many applications in cryptography, where one typically thinks of (x, w) as a public-key/secret-key pair. Some languages (like discrete log) allow for particularly elegant ZK-PoK called Σ-protocols, which we define in §2. Informally, these are three-move public-coin protocols [13,12] that satisfy the "special honest-verifier zero-knowledge" and "special soundness" properties. The first means that there exists a simulator Sim that, given any second message (the challenge) c, can sample an entire transcript (a, c, z) with the same distribution as a real transcript. The special soundness property states that from any two accepting transcripts (a, c, z) and (a, c', z') with the same first, but different second messages, one can extract a witness.

A Σ-protocol can be turned into a non-interactive ZK-PoK (NI-ZK-PoK) in the random-oracle model using the Fiat–Shamir transform [13]. In this transform, the challenge in the second message is not sampled at random by the verifier, but is instead the output of the random oracle on the instance and the first message (we give an example in §2.2). To argue knowledge soundness of this transformation, one typically assumes that the extractor can program the random oracle and rewind the prover. Such a rewinding extractor is problematic for composition (in particular UC [6] or when recursively composing zero-knowledge proofs [4]) and in the quantum setting [22].

A NI-ZK-PoK allows for *straight-line extraction* if the extractor does not require rewinding. A transform from Σ-protocols into straight-line extractable

NI-ZK-PoK in the random-oracle model was proposed by Pass [20], relying on cut-and-choose techniques. A more efficient transformation using "proofs of work" was later proposed by Fischlin [14]; we review his transform in §2.3. These transformations also do not require the extractor to program the random oracle; the extractor only needs to observe the queries made by the prover to the random oracle $\mathcal{H}$.

Recently, two papers on "proofs of complete knowledge" by Kelkar et al. [17], and "individual cryptography" by Dziembowski, Faust, and Lizurej [10], argue that even straight-line extraction is not sufficient for some applications.

As an example, consider vote selling. A party might hold a secret that is necessary for voting, and also be inclined to sell its voting rights, but not give away the secret, for example because it can be used to retrieve collateral on a blockchain. This party could "encumber" its secret, for example by secret-sharing it among several parties that it believes will not collude to steal its collateral, but are willing to engage in a multiparty computation (MPC) to sign votes. Alternatively, this party could use a trusted execution environment (TEE), such as Intel SGX, that contains the secret and will sign votes, but will not help in retrieving the collateral. We refer to [17,10,11,5,18,3] for more examples.

Both papers [17,10] suggest the same high-level idea to prevent encumbrance in the random-oracle model: construct schemes where the computation that depends on the secret (say, computing a signature) must make a huge number q of queries to the random oracle $\mathcal{H}$. Choose q sufficiently larger than an upper bound s on the number of queries that one believes can be evaluated in an encumbered way, and ensure that the secret can be efficiently extracted from the non-encumbered queries. Formally, knowledge soundness is modeled by not giving the extractor all q queries to $\mathcal{H}$ made by an adversary (as in straight-line extraction), but only $q - s$ of them, where the s omitted queries are chosen by the adversary. In [10], and also in this work, the adversary may additionally split those $q - s$ queries into p sets, and extraction must succeed from at least one of them. This models that in an MPC computation with p parties, at least one party must know the secret in the clear.

The parameter s represents an upper bound on the number of hash queries we believe an adversary could evaluate in an encumbered way, meaning that no single party learns the entire input to the query in the clear. Ideally, a construction should remain secure for as large a value of s as possible, so [17,10] suggest using outdated Bitcoin ASICs, which can evaluate the SHA256 hash function extremely efficiently, for this purpose. Instead of simply making sure the construction is secure against a large number s of encumbered queries, we aim for a construction where it's particularly difficult to achieve even a modest number of such queries. For this we leverage the observation, already discussed in [10], that encumbrance, especially via MPC, becomes particularly difficult when the computation is highly sequential, since this significantly increases the round complexity of practical MPC protocols.

We revisit the construction of zero-knowledge proofs of complete knowledge (ZK-PoCK) from [17]. Their construction essentially applies the Fischlin trans-

form to a Σ-protocol, but uses a much higher difficulty for the underlying proof of work than one would in Fischlin's transform to account for the fact that many queries can be encumbered, and thus will not be available to the extractor.

We adapt their definition to obtain zero-knowledge *sequential* proofs of complete knowledge (ZK-SPoCK), where instead of just ensuring that a malicious prover must make a large number of encumbered $\mathcal{H}$ queries to break complete-knowledge soundness, we require that those encumbered queries must be made *sequentially*. As we can have a much more conservative bound on the *number of rounds* in which an encumbered query is made, compared to the total number of encumbered queries, this leads to more practical constructions that can potentially be executed on standard hardware rather than Bitcoin ASICs.

We construct a ZK-SPoCK by embedding the ZK-PoCK from [17] into the computation of the proof of sequential work (PoSW) from [7]. Our scheme can be made non-interactive and turned into a signature scheme using the Fiat–Shamir transform.

While sequentiality is a powerful tool to prevent encumbrance via MPC, it is less effective against TEEs. To address this, in §7 we consider a variant of our ZK-SPoCK using the proof of sequential work from [19] instead of [7]. The [19] PoSW is based on depth-robust graphs, which need a large amount of memory during evaluation [1]. Thus, in this variant the (honest or malicious) prover must not only perform a long sequential computation, but also use a large amount of memory during the computation, which is something TEEs struggle with.

The PoSW from [7] is parameterized by some parameter $N = 2^n$, and an honest prover makes $N' = 2N - 1$ simple sequential queries, while for some small $\alpha > 0$, a malicious prover making $N'(1-\alpha)$ sequential queries (where each sequential query can contain many inputs to be evaluated in parallel), will almost certainly fail. More precisely, by challenging the prover to open a commitment on M positions, the prover will succeed with probability at most $\approx (1-\alpha)^M$.

Our ZK-SPoCK is based on the [7] PoSW and will have basically the same parameters (as a PoSW), but that's not what we care about. Instead, we want complete-knowledge soundness to hold. In our ZK-SPoCK, the N of the $N' = 2N - 1$ sequential queries (so about half) are "leaf" queries, and those will be special as from the inputs to just a few of those queries we can extract the secret. Thus, we achieve complete-knowledge soundness against any prover who can make somewhat fewer than N rounds that contain encumbered (or "slow") queries. In fact, we can allow any polynomial number of encumbered queries, as long as all those queries are made in somewhat less than N sequential rounds. More precisely, an adversary making $s \approx N(1-\alpha)$ sequential rounds of encumbered queries (and polynomially many non-encumbered queries besides this), must either allow for successful extraction of the secret or fail to make the verifier accept with probability $\approx \left(\frac{s}{N}\right)^M \approx (1-\alpha)^M$, where M is the number of challenges (cf. eq.(10) for the precise bound). Note there's a factor 2 gap between the $N' = 2N - 1$ sequential queries one needs to compute the proof, and the N sequential queries an adversary against complete-knowledge soundness must

encumber. This gap can be narrowed, but we didn't optimize for this and instead aimed for a simple and elegant construction.

2 Σ-protocols

A zero-knowledge proof of knowledge [15] (zkPoK) for an NP language L given by a relation $\mathcal{R}_L$ (i.e., $x \in L \iff \exists w : \mathcal{R}_L(x, w) = 1$) is an interactive protocol between a prover $\mathsf{P}(x, w)$ and a verifier $\mathsf{V}(x)$ where the prover convinces the verifier that they know a witness w' such that $\mathcal{R}_L(x, w') = 1$.

The zero-knowledge property is captured by the existence of an efficient simulator who can sample transcripts that are indistinguishable from a real interaction, while the proof of knowledge property is captured by the existence of an efficient extractor, which, given access to a prover that can make the verifier accept, will extract a witness.

Some languages, like discrete logarithms $\mathcal{R}_{\mathrm{DL}}((g, h), w) = 1 \iff g^w = h$ where g is a generator of a prime order group, allow for particularly elegant zkPoKs called Σ-protocols (or sigma-protocols) [13,12].

Definition 1. A Σ-protocol *(sigma-protocol) for a relation $\mathcal{R}$ is a three-move public-coin protocol defined by four PPT algorithms $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma, \mathsf{Sim}_\Sigma, \mathsf{Ext}_\Sigma)$.*

- $\mathsf{P}_\Sigma(x, w) \to (a, \mathsf{st})$ *on input an instance and witness $(x, w) \in \mathcal{R}$ outputs a value (called commitment) a and state st, a is sent to V.*
- $\mathsf{V}_\Sigma(x, a) \to c$ *then samples a random challenge $c \leftarrow\!\!\$\ \mathcal{C}$ from a challenge space $\mathcal{C}$.*
- $\mathsf{P}_\Sigma(x, w, c, \mathsf{st}) \to z$ *computes the answer z.*
- $\mathsf{V}_\Sigma(x, a, c, z) \to \{\mathsf{acc}, \mathsf{rej}\}$ *decides whether to accept the transcript.*

A Σ-protocol has the following properties:

perfect completeness: $\mathsf{V}_\Sigma(x)$ *always accepts when interacting with an honest prover, i.e., for any $(x, w) \in \mathcal{R}$*

$$\Pr[\mathsf{V}_\Sigma(x, a, c, z) = \mathsf{acc} \mid (a, \mathsf{st}) \leftarrow \mathsf{P}_\Sigma(x, w); c \leftarrow\!\!\$\ \mathcal{C}; z \leftarrow \mathsf{P}_\Sigma(x, w, c, \mathsf{st})] = 1$$

special soundness: *From two accepting transcripts with the same first, but different second messages, one can efficiently extract a witness, i.e.,*

$$\mathsf{Ext}_\Sigma(x, (a, c, z), (a, c', z')) \to w \text{ where } (x, w) \in \mathcal{R}$$

special honest-verifier zero-knowledge: *For any $(x, w) \in \mathcal{R}$ and challenge c, the simulator $\mathsf{Sim}_\Sigma(x, c)$ can sample a transcript that is (computationally or statistically) close to a real interaction conditioned on the challenge being c*

$$\left\{(a, c, z) \;\middle|\; (a, z) \leftarrow \mathsf{Sim}_\Sigma(x, c) \right\} \approx_{c/s} \left\{(a, c, z) \;\middle|\; \begin{array}{r} (a, \mathsf{st}) \leftarrow \mathsf{P}_\Sigma(x, w) \\ z \leftarrow \mathsf{P}_\Sigma(x, w, c, \mathsf{st}) \end{array} \right\}.$$

2.1 Schnorr's Protocol for DL.

Let $\mathbb{G}$ be a prime order cyclic group of order q and g some generator. Schnorr's protocol is a Σ-protocol for the language of discrete logarithms where $((g,h),w) \in \mathcal{R}_{\mathrm{DL}}$ if $h = g^w$.

- $\mathsf{P}((g,h),w) \to (a = g^r, \mathsf{st} = r)$ where $r \leftarrow\!\!\$\ \mathbb{Z}_q$
- $\mathcal{C} = \mathbb{Z}_q$
- $\mathsf{P}((g,h),w,c,r) \to (z = r + c \cdot w)$
- $\mathsf{V}((g,h),a,c,z) = \mathsf{acc} \iff g^z = a \cdot h^c$

2.2 Removing interaction: Fiat-Shamir

As Σ-protocols are public-coin, i.e., the verifier only sends a uniformly random challenge, one can make the protocol non-interactive by deriving the challenge as a hash $\mathcal{H}(x,a)$ of the instance and the first prover message; this is known as the Fiat-Shamir transform [13]. For the discrete log relation $\mathcal{R}_{\mathrm{DL}}$, such a non-interactive zero-knowledge proof of knowledge (NI-ZK-PoK) is computed as

$$\begin{aligned} &\mathsf{P}((g,h),w) \to ((g,h),a,c,z) \ \text{ where} \\ &r \leftarrow\!\!\$\ \mathbb{Z}_q, a = g^r, c := \mathcal{H}((g,h),a), z = r + c \cdot w \end{aligned} \tag{1}$$

The verification is as in the interactive protocol, but one must additionally check that c was correctly derived

$$\mathsf{V}((g,h),a,c,z) = \mathsf{acc} \iff g^z = a \cdot h^c \text{ and } c = \mathcal{H}((g,h),a)$$

Schnorr's signature scheme is basically this proof system, except that for signing a message msg, one would also add msg to the hashed string, i.e., $c := \mathcal{H}(msg,(g,h),a)$.

2.3 Knowledge soundness: rewinding, straight line and complete

In an (interactive) Σ-protocol we are convinced that a successful prover "knows" a witness $w, (x,w) \in \mathcal{R}$, as from any prover who is sufficiently successful, we can extract a witness. Formally, there's a knowledge extractor Ext which, given access to a prover $\widetilde{\mathsf{P}}$ that makes the verifier accept with some sufficiently high probability, can extract a witness. For this, Ext can simply run $\widetilde{\mathsf{P}}$, providing a random challenge c, and this way learn an accepting transcript (a,c,z). Then *rewind* $\widetilde{\mathsf{P}}$ to the point where it received c, and send another random challenge $c' \neq c$ instead. If this gives an accepting transcript (a,c',z'), by special soundness, the extractor can compute a witness.

Fiat-Shamir (Rewinding Extractor) Once we make the protocol non-interactive using Fiat-Shamir, the challenge c is computed as a hash of $\mathcal{H}(x, a)$, and thus rewinding no longer works. We can still show knowledge soundness in this case, but now have to rely on the random-oracle heuristic, where $\mathcal{H}$ is modeled as a random function which all parties can only access through black-box queries. Concretely, Ext can still extract a witness if (1) we allow the extractor to observe all the queries made by $\widetilde{\mathsf{P}}$ (2) to rewind $\widetilde{\mathsf{P}}$ and (3) also to *program* $\mathcal{H}$ (i.e., Ext can choose the outputs for $\mathcal{H}$ on some chosen inputs).

Fischlin's transform (Straight-Line Extractor) Fischlin [14] proposed an alternative way to transform a Σ-protocol into a NI-ZK-PoK. While the NI-ZK-PoK from Fischlin's transformation is less efficient than using Fiat-Shamir, it has a straight-line knowledge extractor (i.e., no rewinding is necessary), moreover the extractor does not need to program the random oracle. The extractor can compute a witness just given the inputs to the random-oracle queries made in one execution of the prover.

The idea is to let the prover hash entire valid transcripts (a, c, z) of the Σ-protocol in such a way that a successful prover must almost certainly hash at least two tuples $(a, c, z), (a, c', z')$ with the same first message, but different challenge. The extractor, who can observe those inputs, can then extract a witness from any such pair.

Consider a Σ-protocol with prover and verifier $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ for a relation $\mathcal{R}$. Applying Fischlin's transform we get a NI-ZK-PoK with prover and verifier (P, V) as outlined below. The transform is parametrized by the number of executions M and a proof-of-work difficulty d. We get roughly $\approx M \cdot d$ bits "knowledge soundness security" (so $M = d = 16$ is a good choice if we aim for 256 bit security). The proof size and verifier complexity scale like M, while the prover complexity scales like $M \cdot 2^d$.

Below we outline the NI-ZK-PoK $(\mathsf{P}_F, \mathsf{V}_F, \mathsf{Sim}_F, \mathsf{Ext}_F)$ we get by applying Fischlin's transform to a Σ-protocol $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma, \mathsf{Sim}_\Sigma, \mathsf{Ext}_\Sigma)$. We sketch a somewhat simpler version where the prover runs in *expected* time $M \cdot 2^d$, while the running time in Fischlin [14] is strictly polynomial, but there's an exponentially small completeness error.

Random Oracle $\mathcal{H}$ and Security Parameter λ. All algorithms $(\mathsf{P}_F, \mathsf{V}_F, \mathsf{Sim}_F, \mathsf{Ext}_F)$ get access to a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$, where throughout λ denotes a statistical security parameter. Ext doesn't really need the random oracle (it just gets to see some of the queries made to it), while Sim_F in some protocols is even allowed to program $\mathcal{H}$.

ρ for Salting. The prover $\mathsf{P}_F^{\mathcal{H}}$ will get as input the instance and witness $(x, w) \in \mathcal{R}$, and we additionally allow a value $\rho \in \{0,1\}^\lambda$. This value will then be used as prefix to all $\mathcal{H}$ queries made, where to save on notation we define $\mathcal{H}_\rho(\cdot) \stackrel{\mathsf{def}}{=} \mathcal{H}(\rho, \cdot)$. One function of ρ is domain separation, by sampling a fresh random $\rho \leftarrow\!\!\$\ \{0,1\}^\lambda$ before invoking the prover, we can basically assume that $\mathcal{H}_\rho$ behaves like a

freshly sampled random oracle even if the malicious prover did some polynomially bounded precomputation [8].

Definition of Fischlin's transform. The prover $\mathsf{P}_F^{\mathcal{H}}(\rho,(x,w))$ where $(x,w)\in\mathcal{R}$

- Runs $(a_i,\mathsf{st}_i)\leftarrow\mathsf{P}_\Sigma(x,w)$ M times to get a vector $\mathbf{a}=(a_0,\ldots,a_{M-1})$ of first messages.
- For each $i=0,\ldots,M-1$, $\mathsf{P}_F^{\mathcal{H}}$ computes transcripts $(a_i,j,z_{i,j})$ (where $z_{i,j}=\mathsf{P}_\Sigma(x,w,j,\mathsf{st}_i)$) for challenges $j=0,1,2,\ldots$ until it hits a tuple $\pi_i=(a_i,j,z_{i,j})$ where $\mathcal{H}_\rho(x,i,\mathbf{a},\pi_i)$ starts with d zeros. The proof is $\pi=\{\pi_i\}_{i=0}^{M-1}$.

The verifier $\mathsf{V}_F^{\mathcal{H}}(\rho,x,\pi)$ where $\pi=\{\pi_i\}_{i=0}^{M-1},\pi_i=(a_i,c_i,z_i)$.

- Checks that V_Σ accepts all π_i and that for all $i=0,\ldots,M-1$, $\mathcal{H}_\rho(x,i,\mathbf{a},\pi_i)$ starts with d zeros. $\mathsf{V}_F^{\mathcal{H}}$ accepts iff all checks pass.

The straight line extractor Ext_F gets as input the instance x and the queries $\mathcal{Q}$ made by $\widetilde{\mathsf{P}}_F$. By the following claim, for sufficiently large M,d, Ext_F will almost certainly be able to extract a witness whenever $\widetilde{\mathsf{P}}_F$ outputs a valid proof.

Claim (Straight-line Extractor for Fischlin's transform). Consider any $\widetilde{\mathsf{P}}_F$ and let $\mathcal{Q}$ denote the queries $\widetilde{\mathsf{P}}_F$ makes to its oracle $\mathcal{H}$, then

$$\Pr_\rho[\mathsf{V}_F^{\mathcal{H}}(\rho,x,\pi)=\mathsf{acc}\wedge(x,w)\notin\mathcal{R}\ |\pi\leftarrow\widetilde{\mathsf{P}}_F^{\mathcal{H}}(\rho,(x,w))\ ;\ w\leftarrow\mathsf{Ext}_F(x,\mathcal{Q})]\leq\frac{|\mathcal{Q}|}{2^{M\cdot d}}.$$

Proof. To find a proof, $\widetilde{\mathsf{P}}_F$ must choose some $\mathbf{a}=(a_0,\ldots,a_{M-1})$, and then, for each $i\in[0,M-1]$, find a valid proof $\pi_i=(a_i,j,z_{i,j})$ where $\mathcal{H}_\rho(x,i,\mathbf{a},\pi_i)$ starts with d zeros. If for any i it queries two different $(a_i,j,z_{i,j}),(a_i,k,z_{i,k}),k\neq j$, we can extract a witness. To avoid this, $\widetilde{\mathsf{P}}_F$ must be "lucky" with all M queries and hit a tuple where the hash starts with d zeros the first try. For a fixed $\mathbf{a}$ this happens with probability $(2^{-d})^M$. The bound follows by taking the union bound over the at most $|\mathcal{Q}|$ different $\mathbf{a}$ vectors $\widetilde{\mathsf{P}}_F$ tries out. □

Proofs of Complete Knowledge Above, we let Ext_F observe all the $\mathcal{H}$ queries made by $\widetilde{\mathsf{P}}_F^{\mathcal{H}}$. In a proof of complete knowledge, we want Ext_F to succeed even if $\widetilde{\mathsf{P}}_F^{\mathcal{H}}$ can make a large number s of "slow" queries that are *not* given to Ext_F. This way we capture "encumbered queries" (done using an MPC protocol, or a TEE), the remaining queries are called "fast". Moreover we want to capture a distributed prover made up of up to p parties.

To model this, we allow the malicious prover to split all the fast queries into p parts, and require that Ext_F can extract the witness from at least one of those parts, thus capturing that at least one of those distributed adversaries must know the secret.

Concretely, we model our adversary $\widetilde{\mathsf{P}}^{\mathcal{H}}_{F,s,p}$ as a monolithic adversary but we let it choose which s queries to remove from $\mathcal{Q}$, and how to split the remaining queries into (potentially empty) sets $\mathcal{Q}_1, \ldots, \mathcal{Q}_p$. This adversary breaks complete-knowledge soundness if we can't extract a witness from any of $\mathcal{Q}_i$'s. To compute the complete-knowledge error for Fischlin's transform we must, for any malicious prover $\widetilde{\mathsf{P}}_{F,s,p}$, bound

$$\Pr_{\rho}\left[\begin{array}{c} \mathsf{V}^{\mathcal{H}}_F(\rho, x, \pi) = \mathsf{acc} \\ \text{and } \forall i \in [p] \ : \ (x, w_i) \notin \mathcal{R} \end{array} \middle| \begin{array}{r} (\pi, \mathcal{Q}_1, \ldots, \mathcal{Q}_p) \leftarrow \widetilde{\mathsf{P}}^{\mathcal{H}}_{F,s,p}(\rho, (x, w)) \\ \text{where } \forall i \in [p] \ : \ \mathcal{Q}_i \subseteq \mathcal{Q} \\ \text{and } |\cup_{i=1}^{p} \mathcal{Q}_i| \geq |\mathcal{Q}| - s \\ w_i \leftarrow \mathsf{Ext}(x, \mathcal{Q}_i) \end{array}\right]$$

This is basically the probability that we solve the M proofs of work, each of difficulty 2^{-d}, using our s slow queries (we get an additional $p \cdot M$ queries as the adversary is distributed, but as in practice $s \gg p \cdot M$, we can ignore this). More formally, this is the probability that, when sampling s variables ($s + p \cdot M$ to be precise), each being 1 with probability 2^{-d}, we see at least M 1's. To choose parameters, we need to make an assumption on s, say we believe no adversary can make $s = 2^{30}$ (around a billion) *encumbered* queries in the time from receiving the beacon ρ to outputting the proof. Now, we could set $M = 16 = 2^4$ and $d = 34$, and the bound becomes roughly 2^{-105} (by a Chernoff bound).

Looking ahead, for the *sequential* proof of complete knowledge we construct in this work, we'll get similar complete-knowledge soundness, but not in terms of the total number of encumbered queries s, but the number s' of sequential rounds in which the adversary can perform encumbered queries. When considering encumbrance using MPC, we can assume a bound on s' which is much smaller than a meaningful bound on s. As a consequence, we can choose much smaller parameters (especially d), and thus get a more practical construction.

3 Proofs of Sequential Work

Proofs of sequential work (PoSW) were introduced by Mahmoody, Moran and Vadhan [19]. Below, we consider the more efficient construction of Cohen and Pietrzak [7] (but we'll also consider the [19] PoSW in §7 when discussing TEEs). Informally, a PoSW is a proof system by which a prover P convinces a verifier V that they did some inherently sequential computation. The construction from [7] is in the random oracle model, and based on a DAG as illustrated in Fig.1. The prover, on input a challenge ρ, uses ρ to salt a random oracle $\mathcal{H}_\rho(\cdot) \stackrel{\mathsf{def}}{=} \mathcal{H}(\rho, \cdot)$, and then sequentially computes a labeling of the DAG, where the label of a node is $\mathcal{H}_\rho$ applied to the labels of its parents. P then sends the label of the root (one can think of this as a Merkle commitment to all the labels) to V, who then chooses M challenges $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{M-1})$ and sends $\boldsymbol{\gamma}$ to P. The prover P must reply with an opening to the label of the leaf nodes $\gamma_i \in \boldsymbol{\gamma}$.

The security proof in [7] goes along the following lines. Let N' be the size of the DAG. If, for some $1 > \alpha > 0$, a malicious prover $\widetilde{\mathsf{P}}$ only makes $(1 - \alpha)N'$

instead of N' sequential queries (where each query can contain polynomially many inputs to be evaluated in parallel), then the prover will not be able to open an α fraction of the leaf nodes (i.e., for a node $v \in \{0,1\}^n$, provide the labels of the nodes $\mathsf{open}(v)$ as defined below) and thus fail to convince the verifier with probability $1-(1-\alpha)^M$. Thus, while an honest prover makes N' simple sequential queries, a malicious prover can't win even with $N'(1-\alpha)$ sequential queries, where with each sequential query, they can evaluate many inputs in parallel.

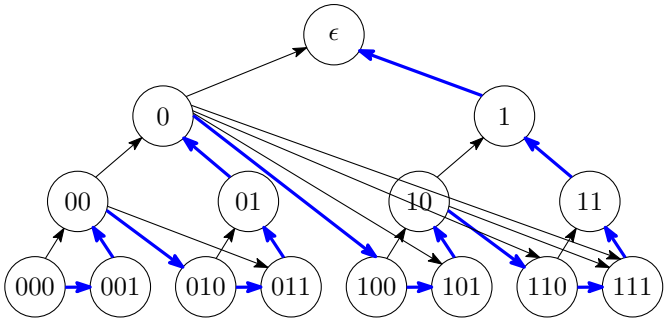

Fig. 1. The DAG G_n underlying the PoSW from [7]. Here shown for $n = 3$, so we have $N = 2^n$ leaf nodes indexed by $\{0,1\}^n$ and $N' = 2N-1$ indexed with $\{0,1\}^{\leq n}$ in total. The directed path from 0^n to ϵ, indicating the order in which the labels must be computed, is shown in blue.

For an index $u \in \{0,1\}^k$, we'll denote with $u[1\ldots k] \in \{0,1\}^i$ its i-bit prefix with the convention $u[1\ldots -1]$ being the index ε of the root. For a leaf node $v = v[1\ldots n] \in \{0,1\}^n$, we denote with $\mathsf{open}(v)$ the nodes whose labels one needs to open the label ℓ_v, which are exactly the nodes on the co-path from the root to v.

$$\mathsf{open}(v) \stackrel{\mathsf{def}}{=} \{v[1\ldots k-1] \| (1-v[k])\}_{k=1}^{n} \tag{2}$$

Note that given the labels for the nodes in $\mathsf{open}(v)$, we can recompute all the labels on the path from v to the root ε. This opening is a valid opening (of the recomputed label of v) if the recomputed label of the root matches the commitment.

4 SPoCK definition

In this paper we construct a public-coin ZK-SPoCK as defined in §4.1 below. For some applications, for example when we want to construct signatures, we want a *non-interactive* zero-knowledge sequential proof of complete knowledge (NI-ZK-SPoCK). We define it in §4.2, and observe that we can construct a NI-ZK-SPoCK from a public-coin ZK-SPoCK as defined below using the Fiat-Shamir heuristic.

As all our constructions rely on a hash function $\mathcal{H}$, which for the analysis is modeled as a random oracle, we already explicitly model the oracle access to $\mathcal{H}$ in the definition. We don't consider precomputation attacks. This is justified if we assume that the salt ρ is some fresh random value sampled at the beginning of the security game (in the interactive setting, the verifier can choose ρ, in the non-interactive setting it must be some unpredictable beacon, e.g., the hash of a recent Bitcoin block).

4.1 Public-coin ZK-SPoCK Definition

A three-move public-coin ZK-SPoCK for a relation $\mathcal{R}$ is given by four algorithms $(\mathsf{P}^{\mathcal{H}} = (\mathsf{P}_1^{\mathcal{H}}, \mathsf{P}_2^{\mathcal{H}}), \mathsf{V}^{\mathcal{H}} = (\mathsf{V}_1^{\mathcal{H}}, \mathsf{V}_2^{\mathcal{H}}), \mathsf{Ext}, \mathsf{Sim}^{\mathcal{H}})$. With $\mathsf{par} = \{\lambda, \Gamma, N, \ldots\}$ we denote the public parameters used in the scheme, which must at least include the security parameter λ, a sequentiality parameter $N \in \mathbf{poly}(\lambda)$ and a challenge space Γ.

- $\mathsf{P}_1^{\mathcal{H}}(\mathsf{par}, \rho, (x, w)) \to (\Phi, \mathsf{st})$ on input a salt ρ and an instance with witness $(x, w) \in \mathcal{R}$ outputs a message Φ (that is sent to the verifier, think of it as a type of commitment) and a state st (to be passed to the second stage).
- $\mathsf{V}_1^{\mathcal{H}}(\mathsf{par}, \rho, x, \Phi) \to \gamma$ outputs a random challenge $\gamma \leftarrow\!\!\$\ \Gamma$.
- $\mathsf{P}_2^{\mathcal{H}}(\mathsf{par}, \rho, (x, w), \mathsf{st}, \gamma) \to \pi$ computes the proof from the state and a challenge $\gamma \in \Gamma$.
- $\mathsf{V}_2^{\mathcal{H}}(\mathsf{par}, \rho, x, \Phi, \gamma, \pi) \to \mathsf{acc}/\mathsf{rej}$ decides whether to accept or reject the proof.

Ideally, for a given N (which is part of par), we want $\mathsf{P}_1^{\mathcal{H}}(\mathsf{par}, \rho, (x, w))$ to make not much more than N sequential queries to $\mathcal{H}$, while $\mathsf{P}_2^{\mathcal{H}}$ should ideally make $o(N)$ queries and $\mathsf{V}_2^{\mathcal{H}}$ should be highly efficient and run in $\mathbf{poly}(\log(N), \lambda)$ time ($\mathsf{V}_1^{\mathcal{H}}$ is efficient by definition).

Below $\mathcal{Q}$ will denote the $\mathcal{H}$ queries (i.e., input/outputs) made by P_1. Recall that P_1 can query $\mathcal{H}$ in rounds, in each round giving a batch $x_1, x_2, \ldots$ of inputs, and receiving the outputs $y_1 := \mathcal{H}(x_1), y_2 := \mathcal{H}(x_2), \ldots$. We will consider subsets $\mathcal{Q}' \subseteq \mathcal{Q}$, and as for us it will be relevant in which round a particular query was made, we assume queries are stored as triples (i, x, y), where i indicates in which round the input x was queried. We'll denote with $|\mathcal{Q}|$ the total number of queries in $\mathcal{Q}$ and with

$$\mathsf{rounds}(\mathcal{Q}') = |\{i \ : \ (i, x, y) \in \mathcal{Q}'\}|$$

The number of distinct rounds in which the queries in $\mathcal{Q}'$ were made. Note that for $\mathcal{Q}' \subseteq \mathcal{Q}$, $\mathsf{rounds}(\mathcal{Q} \setminus \mathcal{Q}') \leq s$ means that $\mathcal{Q}'$ covers all of $\mathcal{Q}$ except for some queries that all fall into at most s different rounds.

A ZK-SPoCK satisfies the following properties

Completeness. An honestly generated proof should be accepted. For any $(x, w) \in \mathcal{R}$

$$\Pr\left[\mathsf{V}_2^{\mathcal{H}}(\mathsf{par}, \rho, x, \Phi, \boldsymbol{\gamma}, \pi) = \mathsf{acc} \;\middle|\; \begin{array}{c}(\Phi, \mathsf{st}) \leftarrow \mathsf{P}_1^{\mathcal{H}}(\mathsf{par}, \rho, (x, w)) \\ \boldsymbol{\gamma} \leftarrow \mathsf{V}_1^{\mathcal{H}}(\mathsf{par}, \rho, x, \Phi) \\ \pi \leftarrow \mathsf{P}_2^{\mathcal{H}}(\mathsf{par}, \rho, (x, w), \mathsf{st}, \boldsymbol{\gamma})\end{array}\right] = 1. \quad (3)$$

Soundness (Complete-Knowledge) For some parameter s (denoting the number of slow rounds of queries), and p (denoting the number of subadversaries as discussed in §2.3), consider a two part malicious prover $(\widetilde{\mathsf{P}}_{1,s,p}, \widetilde{\mathsf{P}}_2)$. $\widetilde{\mathsf{P}}_{1,s,p}$, apart from the commitment Φ and a state st to be passed to $\widetilde{\mathsf{P}}_2$, can also choose subsets $\mathcal{Q}_1, \ldots, \mathcal{Q}_p$ of the $\mathcal{H}$ queries $\mathcal{Q}$ made by $\widetilde{\mathsf{P}}_{1,s,p}$, deciding which subadversary gets to see which queries. Those sets must cover all of $\mathcal{Q}$, except some queries that must all be contained in at most s distinct rounds. We want the probability

$$\Pr_{\rho}\left[\begin{array}{l|r} \mathsf{V}_2^{\mathcal{H}}(\mathsf{par},\rho,x,\Phi,\boldsymbol{\gamma},\pi)=\mathsf{acc} & (\Phi,\mathsf{st},\mathcal{Q}_1..\mathcal{Q}_p)\leftarrow \widetilde{\mathsf{P}}_{1,s,p}^{\mathcal{H}}(\mathsf{par},\rho,(x,w)) \\ \text{and } \forall i\in[p]\ :\ (x,w_i)\notin\mathcal{R} & \boldsymbol{\gamma}\leftarrow \mathsf{V}_1^{\mathcal{H}}(\mathsf{par},\rho,x,\Phi) \\ & \pi\leftarrow \widetilde{\mathsf{P}}_2^{\mathcal{H}}(\mathsf{par},\rho,(x,w),\mathsf{st},\boldsymbol{\gamma}) \\ & \text{where } \forall i\in[p]\ :\ \mathcal{Q}_i\subseteq\mathcal{Q} \\ & \text{and } \mathsf{rounds}\left(\mathcal{Q}\setminus\cup_{i=1}^{p}\mathcal{Q}_i\right)\le s \\ & \forall i\in[p]: w_i\leftarrow \mathsf{Ext}(\mathsf{par},x,\mathcal{Q}_i) \end{array}\right]$$

that V_2 accepts and at the same time the extractor fails to extract from all the $\mathcal{Q}_i$'s to be negligible.

Honest-Verifier Zero-Knowledge. We require the existence of an efficient simulator Sim where for any ρ and $(x,w)\in\mathcal{R}$, $\mathsf{Sim}^{\mathcal{H}}(\mathsf{par},\rho,x)$ outputs a transcript that is indistinguishable from a transcript $(\Phi,\boldsymbol{\gamma},\pi)$ as generated in eq.(3).

4.2 NI-ZK-SPoCK Definition

In this section we define a non-interactive ZK-SPoCK, which can be constructed by using the Fiat-Shamir heuristic to the public-coin ZK-SPoCK from the previous section. Concretely, instead of letting V_1 sample a random challenge $\boldsymbol{\gamma}$, this challenge is computed as a hash $\boldsymbol{\gamma}\leftarrow\mathcal{H}(\rho,x,\Phi)$ of the instance and first message.

A NI-ZK-SPoCK for a relation $\mathcal{R}$ is given by four algorithms $(\mathsf{P}^{\mathcal{H}},\mathsf{V}^{\mathcal{H}},\mathsf{Ext},\mathsf{Sim}^{\mathcal{H}})$.

$\mathsf{P}^{\mathcal{H}}(\mathsf{par},\rho,(x,w))\to\pi$ on input public parameters par (as in the previous section), a salt ρ, an instance with witness $(x,w)\in\mathcal{R}$ outputs a proof π.
$\mathsf{V}^{\mathcal{H}}(\mathsf{par},\rho,x,\pi)\to\mathsf{acc}/\mathsf{rej}$ decides whether to accept or reject.

We require the following properties

Completeness. An honestly generated proof should be accepted. For any $(x,w)\in\mathcal{R}$

$$\Pr\left[\mathsf{V}^{\mathcal{H}}(\mathsf{par},\rho,x,\pi)=\mathsf{acc}\mid\pi\leftarrow\mathsf{P}^{\mathcal{H}}(\mathsf{par},\rho,(x,w))\right]=1. \tag{4}$$

Soundness (Complete-Knowledge). For some parameter s denoting the number of slow query rounds, and p denoting the number of subadversaries (as discussed in §2.3), consider a malicious prover $\widetilde{\mathsf{P}}$, which apart from the proof

π must also choose subsets $\mathcal{Q}_1, \ldots, \mathcal{Q}_p$ that cover all of its $\mathcal{H}$ queries, except for a set that must all fall into at most s rounds.

$$\Pr_{\rho}\left[\begin{array}{c|l} \mathsf{V}^{\mathcal{H}}(\mathsf{par}, \rho, x, \pi) = \mathsf{acc} & (\pi, \mathcal{Q}_1..\mathcal{Q}_p) \leftarrow \widetilde{\mathsf{P}}^{\mathcal{H}}_{s,p}(\mathsf{par}, \rho, (x, w)) \\ \text{and } \forall i \in [p] \ : \ (x, w_i) \notin \mathcal{R} & \quad \text{where } \forall i \in [p] \ : \ \mathcal{Q}_i \subseteq \mathcal{Q} \\ & \quad\quad \text{and } \mathsf{rounds}\left(\mathcal{Q} \setminus \cup_{i=1}^{p} \mathcal{Q}_i\right) \leq s \\ & \quad\quad\quad \forall i \in [p] : w_i \leftarrow \mathsf{Ext}(\mathsf{par}, x, \mathcal{Q}_i) \end{array}\right]$$

that the extractor cannot extract a witness from any of the $\mathcal{Q}_i$'s, assuming V accepts, to be negligible.

Zero-Knowledge. We require the existence of an efficient simulator Sim where for any ρ and $(x, w) \in \mathcal{R}$, $\mathsf{Sim}^{\mathcal{H}}(\mathsf{par}, \rho, x)$ outputs a proof π that is indistinguishable from a real proof $\pi \leftarrow \mathsf{P}^{\mathcal{H}}(\mathsf{par}, \rho, (x, w))$. The simulator is not given a witness, but instead is allowed to program the random oracle $\mathcal{H}$.

5 Construction of a public-coin ZK-SPoCK

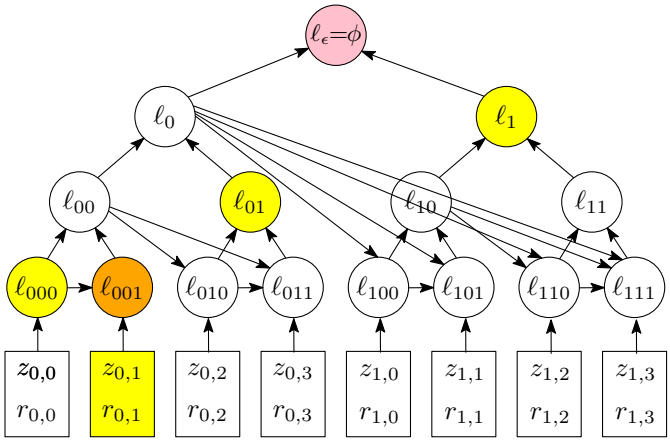

Fig. 2. Illustration of our ZK-SPoCK for a toy example with $M = 2$ and $N = 8$. The prover will sample $\mathbf{a} = (a_0, a_1)$ and $z_{i,j}$ that satisfy $\mathsf{V}_\Sigma(x, a_i, j, z_{i,j}) \to \mathsf{acc}$ and (pseudo)random $r_{i,j}$ values. It will then compute all labels sequentially, where a label is the hash of the labels of its parents (and also $\mathbf{a}$ and a beacon ρ). The commitment $\phi = \ell_\epsilon$ and $\mathbf{a}$ are sent to the verifier, who sends a challenge $\boldsymbol{\gamma} = (\gamma_0, \gamma_1) \in [0, N/M - 1]^M$, say $(1, 2)$, and the prover must now open ℓ_{001} and ℓ_{110} (the labels above $z_{i,\gamma_i}, i \in [0, M-1]$). To open the (orange) label ℓ_{001}, we must provide all the (yellow) labels on the co-path from node 001 to ϵ (i.e., $\ell_{000}, \ell_{01}, \ell_1$), together with the $z_{0,1}$ and $r_{0,1}$ values.

In this section we present our sequential proof of complete knowledge. Figure 2 illustrates this construction which is based on the DAG G_n from the [7] PoSW as shown in Figure 1.

The construction relies on the following building blocks and parameters.

Σ A Σ-protocol $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma, \mathsf{Ext}_\Sigma, \mathsf{Sim}_\Sigma)$ for a relation $\mathcal{R}$.

$\mathcal{H}, \lambda, \rho$ A hash function (modeled as random oracle in the analysis) $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$. Here λ is a statistical security parameter, say $\lambda = 256$. The salt ρ is used as a prefix to $\mathcal{H}$, we use the notation $\mathcal{H}_\rho(\cdot) = \mathcal{H}(\rho, \cdot)$.

$G_n, N = 2^n, N' = 2N - 1$ The DAG from [7] (cf. Fig 1) on $N' = 2N - 1 = 2^{n+1} - 1$ vertices with N leaves.

$M = 2^m$ The challenge space in our construction is $\Gamma \stackrel{\mathsf{def}}{=} [0, N/M - 1]^M$. [4] On challenge $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{M-1}) \leftarrow \Gamma$, the prover must open M leaf labels. We will assume that M is a power of two as this simplifies exposition, and set $m = \log(M)$.

o The parameter $o \in [0, n-1]$ is only used for efficiency during the opening phase: by storing the labels of the top o levels in the tree (which requires $\lambda \cdot 2^{o+1}$ bits), one can recompute the opening of a leaf label making 2^{n-o+1} $\mathcal{H}$ queries.

Our three-move public-coin SPoCK $(\mathsf{P}_1^\mathcal{H}, \mathsf{P}_2^\mathcal{H}, \mathsf{V}_1^\mathcal{H}, \mathsf{V}_2^\mathcal{H}, \mathsf{Ext}, \mathsf{Sim}^\mathcal{H})$ is defined below with public parameters $\mathsf{par} = (\lambda, n, m, o)$ as outlined above. It will be convenient to define a mapping $\Lambda : \{0,1\}^n \to [0, M-1] \times [0, N/M - 1]$ from leaf labels $\{0,1\}^n$ to indices, think of the first index as choosing one out of M bins, and the second as the position in the bin of size N/M (below think of $v \in \{0,1\}^n$ as a number in $[0, 2^n - 1]$).

$$\Lambda(v) \stackrel{\mathsf{def}}{=} (\lfloor v/2^{n-m} \rfloor \ , \ v \mod 2^{n-m}) \tag{5}$$

$\mathsf{P}_1^\mathcal{H}(\mathsf{par}, \rho, (x, w)) \to ((\mathbf{a}, \phi), \mathsf{st})$:

- Sample M first messages $\mathbf{a} = (a_0, \ldots, a_{M-1})$, $(a_i, \mathsf{st}_i) \leftarrow \mathsf{P}_\Sigma(x, w)$ of the Σ protocol.
- For each $i \in [0, 2^m - 1], j \in [0, 2^{n-m} - 1]$ compute the third message of a Σ transcript $(a_i, j, z_{i,j})$

$$z_{i,j} \leftarrow \mathsf{P}_\Sigma(x, w, j, \mathsf{st}_i). \tag{6}$$

- Sample a random $r \leftarrow\!\!\$\ \{0,1\}^\lambda$ and define

$$r_{i,j} := \mathcal{H}(r, i, j). \tag{7}$$

  Think of the $r_{i,j}$ as (pseudo)random values which can be derived from the short seed r.
- Compute a labelling $\{\ell_v\}_{v \in \{0,1\}^{\le n}}$ of G_n (making N' sequential queries to $\mathcal{H}_\rho$) as

$$\ell_v := \mathcal{H}_{\rho,\mathbf{a}}(v, \epsilon_v, \ell_{p_1}, \ldots, \ell_{p_d}), \tag{8}$$

  where $(p_1, \ldots, p_d) = \mathrm{parents}(v)$ and

[4] Here we assume $[0, N/M - 1]$ is a subset of the challenge set of the underlying Σ protocol, or at least that there's an efficient mapping of $[0, N/M - 1]$ to distinct challenges.

1. If v is not a leaf, i.e., $v \in \{0,1\}^{<n}$, then $\epsilon_v = \emptyset$.
2. If v is a leaf, let $(i,j) := \Lambda(v)$ (cf. eq.(5)) and set $\epsilon_v = (z_{i,j}, r_{i,j})$.

Let $\phi = \ell_\epsilon$ be the label of the root (which we think of as a Merkle commitment to all labels). To speed up openings in the second phase, we'll also store the labels in the o highest layers. $\mathsf{st} = (\{\ell_v\}_{v\in\{0,1\}^{\le o}}, r)$.

$\mathsf{V}_1^{\mathcal{H}}(\mathsf{par}, \rho, x, \Phi) \to \boldsymbol{\gamma}$: Outputs $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{M-1}) \leftarrow\!\!\$\ \Gamma \stackrel{\mathsf{def}}{=} [0, N/M-1]^M$.

$\mathsf{P}_2^{\mathcal{H}}(\mathsf{par}, \rho, (x,w), \mathsf{st}, \boldsymbol{\gamma}) \to \boldsymbol{\tau}$: Parse $\mathsf{st} = \{\ell_v\}_{v\in\{0,1\}^{\le o}}$ and $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{M-1})$. Recompute and output the openings of the labels of the nodes $v_i \stackrel{\mathsf{def}}{=} \Lambda^{-1}(i, \gamma_i)$ for $i \in [0, M-1]$, that is (with open as in eq.(2), $z_{i,j}$ as in eq.(6) and $r_{i,j}$ as in eq.(7))

$$\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{M-1}) \text{ where } \tau_i = (\{\ell_u\}_{u\in\mathsf{open}(v_i)}, z_{i,\gamma_i}, r_{i,\gamma_i}), \tag{9}$$

i.e., τ_i is the opening of the commitment $\phi = \ell_\epsilon$ on the positions of the committed labels of $v_i \stackrel{\mathsf{def}}{=} \Lambda^{-1}(i,\gamma_i)$ for $i \in [0, M-1]$.

$\mathsf{V}_2^{\mathcal{H}}(\mathsf{par}, \rho, x, (\mathbf{a}, \phi), \boldsymbol{\gamma}, \boldsymbol{\tau}) \to \mathsf{acc}/\mathsf{rej}$: parse the input into the public parameters par, the beacon ρ, the statement x, the commitment $(\mathbf{a} = (a_0, \ldots, a_{M-1}), \phi)$, the challenge $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{M-1})$ and opening $\tau = (\tau_0, \ldots, \tau_{M-1})$ where $\tau_i = (\{\ell_u\}_{u\in\mathsf{open}(v_i)}, z_{i,\gamma_i}, r_{i,\gamma_i})$ with $v_i = \Lambda^{-1}(i,\gamma_i)$. Now check for all $i \in [0, M-1]$ that

1. $\mathsf{V}_\Sigma(x, a_i, \gamma_i, z_{i,\gamma_i}) \stackrel{?}{=} \mathsf{acc}$ is an accepting transcript of the Σ protocol using the committed a_i as the first, and challenge γ_i as the second message.
2. τ_i is a correct opening (to the label ℓ_{v_i} and transcript $(a_i, \gamma_i, z_{i,\gamma_i})$) by recomputing the labels as in eq.(8).

6 Analysis of the ZK-SPoCK

In this section we analyze the efficiency and prove the soundness and zero-knowledge properties of the public-coin ZK-SPoCK from §5.

6.1 Efficiency

Our scheme is basically as efficient as the underlying PoSW from [7]. The additional computation and communication one needs to additionally embed and check Σ protocol tuples is small in comparison as we'll outline below.

Compared to [7], where we use only ρ as a prefix for hashing, here (cf. eq.(8)) we additionally use $\mathbf{a}$ in the prefix. This is not an issue even though $\mathbf{a}$ is somewhat long: as we use the same prefix throughout the entire labelling, we can first hash the prefix $(\rho, \mathbf{a})$ into as short (say, 256 bits) prefix ρ', and then use $\mathcal{H}_{\rho'}(\cdot) = \mathcal{H}(\rho', \cdot)$ to compute labels.

Another difference is that we additionally hash the random $r_{i,j}$ with the leaf labels. Those are fairly short, moreover we derive them from a single seed r, so there's no extra storage requirement on the prover, it just needs to store the short seed r.

Finally, the prover needs to compute all the $z_{i,j}$. While computing the third message in a Σ protocol can be expensive, we note that we only need to compute tuples for consecutive challenges, i.e., $(a_i, 0, z_{i,0}), (a_i, 1, z_{i,1}), (a_i, 2, z_{i,2}), \ldots$. At least in Schnorr's protocol (cf. eq.(1)) for discrete logarithms, going from $z_{i,j}$ to $z_{i,j+1}$ is very efficient and it just requires a single addition, i.e., $z_{i,j+1} = z_{i,j} + w$.

6.2 Soundness (Complete-Knowledge)

Before we prove complete-knowledge soundness of our protocol, we prove Lemma 1 below, which (informally) states that with overwhelming probability, once the prover sent the commitment ϕ, it will only be able to open leaf labels that were computed in strict total order of random oracle rounds. Let us mention that a stronger version of this lemma already appears in [7], they additionally show that also the labels on the path from a leaf label (that can be opened) to the root must have been computed in order. We prove the weaker lemma to be self-contained, and because the lemma we need has a much simpler proof than the stronger version in [7].

We'll use the following property of G_n.

A useful property of G_n. For any two leaves $v < v'$ in lexicographic order, there is a directed path from v to v' in G_n: write the two leaves at their first differing bit as

$$v = a0b \qquad \text{and} \qquad v' = a1b'.$$

Following the binary-tree edges from v leads to the node $a0$, and by the definition of the additional edges of G_n, the node $a0$ is a parent of v'. Thus any lexicographically ordered subset of leaves lies on a directed path of G_n (as illustrated in Fig. 1).

Lemma 1 (Sequentiality of consistently openable leaves). *Consider a malicious prover $\widetilde{P}_{1,s,p}, \widetilde{P}_2$ (as in §4.1), and let q denote the total number of random-oracle queries made during the experiment. Then, except with probability at most $\frac{\log(N)\, q^2}{2^\lambda}$, the following holds.*

Let $Z \subseteq \{0,1\}^n$ denote the set of leaves that can be opened. For every $v \in Z$, the random-oracle query recomputing the leaf label that will be checked during opening

$$\ell_v = \mathcal{H}_{\rho,\mathbf{a}}(v, z_{i,j} \ldots)$$

was made by $\widetilde{P}_{1,s,p}$ before it output its commitment ϕ. Moreover, if $v_1 < v_2 < \cdots < v_t$ are the leaves in Z, ordered lexicographically, then the corresponding leaf-label queries were made in strictly increasing random-oracle rounds.

Proof. We condition on the following bad event not occurring: either there is a collision among random-oracle outputs, or an oracle output coincides with a label that had already been used as an input to an earlier oracle query. Since every query contains at most $\log(N)$ labels as inputs, a union bound shows that this event occurs with probability at most $\frac{\log(N)\, q^2}{2^\lambda}$.

Conditioned on this event not occurring, a successful opening of a leaf determines unique labels for all nodes on its path and co-path to the root. In particular, these labels must agree whenever two successful openings contain the same node. Moreover, whenever a label ℓ_u is used as input in the query defining another label ℓ_v, the query producing ℓ_u must have occurred in an earlier random-oracle round: otherwise ℓ_u would have had to be guessed before its oracle query was made.

We now consider any two consistently openable leaves $v < v'$ in lexicographic order. Write them at their first differing bit as

$$v = a0b \qquad \text{and} \qquad v' = a1b'.$$

Let $u = a0$. Since u is an ancestor of v in the underlying binary tree, verification of the opening of v recomputes ℓ_u from ℓ_v by successively applying $\mathcal{H}_{\rho,\mathbf{a}}$ along the path from v to u. Hence the query defining ℓ_v occurs before the query defining ℓ_u (unless $u = v$, in which case $\ell_u = \ell_v$).

On the other hand, by the definition of the additional (non-tree) edges of G_n, $u = a0$ is a parent of the leaf $v' = a1b'$. Consequently, the query defining $\ell_{v'}$ takes ℓ_u as one of its inputs, and therefore must occur strictly after the query defining ℓ_u.

This argument holds for any pair $v, v' \in Z$, thus the leaf-label queries for all nodes in Z were made in sequential order as claimed. □

Theorem 1 (Complete-Knowledge Soundness). *Consider a malicious prover $\widetilde{\mathsf{P}}_{1,s,p}, \widetilde{\mathsf{P}}_2$ as in §4.1, and let q denote the total number of random-oracle queries made during the experiment. Then the complete-knowledge soundness error of the construction from §5, which is defined as*

$$\Pr\left[\begin{array}{l|r} \mathsf{V}_2^{\mathcal{H}}(\mathsf{par}, \rho, x, (\mathbf{a}, \phi), \boldsymbol{\gamma}, \boldsymbol{\tau}) = \mathsf{acc} & ((\mathbf{a}, \phi), \mathsf{st}, \mathcal{Q}_1..\mathcal{Q}_p) \leftarrow \widetilde{\mathsf{P}}_{1,s,p}^{\mathcal{H}}(\mathsf{par}, \rho, (x, w)) \\ \textit{and } \forall k \in [p] : (x, w_k) \notin \mathcal{R} & \boldsymbol{\gamma} \leftarrow\!\!\$\ [0, N/M - 1]^M \\ & \boldsymbol{\tau} \leftarrow \widetilde{\mathsf{P}}_2^{\mathcal{H}}(\mathsf{par}, \rho, (x, w), \mathsf{st}, \boldsymbol{\gamma}) \\ & \forall k \in [p] : \mathcal{Q}_k \subseteq \mathcal{Q} \\ & \mathsf{rounds}\left(\mathcal{Q} \setminus \bigcup_{k=1}^{p} \mathcal{Q}_k\right) \leq s \\ & \forall k \in [p] : w_k \leftarrow \mathsf{Ext}(\mathsf{par}, x, \mathcal{Q}_k) \end{array}\right]$$

is at most

$$\leq \left(\frac{s + M \cdot p}{N}\right)^M + \frac{\log(N) \cdot q^2}{2^\lambda}. \tag{10}$$

Proof. Let $\mathcal{Q}_{\mathsf{slow}} := \mathcal{Q} \setminus \bigcup_{k=1}^{p} \mathcal{Q}_k$ denote the slow queries, by assumption they were made in at most s rounds, i.e., $\mathsf{rounds}(\mathcal{Q}_{\mathsf{slow}}) \leq s$.

Below we condition on the bad event from Lemma 1 not happening, which holds except with probability at most $\frac{\log(N) \cdot q^2}{2^\lambda}$.

Fix the commitment $(\mathbf{a}, \phi)$ produced by $\widetilde{\mathsf{P}}_{1,s,p}$. Let Z be the set of leaf positions (i, j) that are consistently openable with respect to ϕ and for which

$$(a_i, j, z_{i,j})$$

is an accepting Σ-protocol transcript.

By Lemma 1, the random-oracle queries defining the leaf labels corresponding to the elements of Z occur in pairwise distinct oracle rounds. Since $\mathcal{Q}_{\mathsf{slow}}$ contains queries from at most s rounds, at most s elements of Z can have their leaf-label query in $\mathcal{Q}_{\mathsf{slow}}$. Consequently, at least

$$|Z| - s$$

of the corresponding leaf-label queries belong to $\bigcup_{k=1}^{p} \mathcal{Q}_k$.

Suppose first that

$$|Z| > s + M \cdot p.$$

Then more than $M \cdot p$ good leaf-label queries belong to $\bigcup_{k=1}^{p} \mathcal{Q}_k$. Associate with every such query the pair (k, i), where $k \in [p]$ is such that the query belongs to $\mathcal{Q}_k$, and $i \in [0, M-1]$ is the bin containing the corresponding leaf. There are only $M \cdot p$ possible pairs (k, i). Hence, by the pigeonhole principle, there exist some k and i such that $\mathcal{Q}_k$ contains two good leaf queries corresponding to (i, j) and (i, j') with $j \neq j'$.

Thus $\mathcal{Q}_k$ contains two accepting Σ-protocol transcripts

$$(a_i, j, z_{i,j}) \qquad \text{and} \qquad (a_i, j', z_{i,j'})$$

with the same first message and different challenges. By special soundness of the underlying Σ protocol, we thus can define an extractor Ext

$$w_k \leftarrow \mathsf{Ext}(\mathsf{par}, x, \mathcal{Q}_k)$$

which extracts a valid witness. Therefore complete-knowledge failure is impossible when $|Z| > s + M \cdot p$.

It remains to consider the case

$$|Z| \leq s + M \cdot p.$$

For $i \in [0, M-1]$, let

$$Z_i := \{j : (i, j) \in Z\} \qquad \text{and} \qquad b_i := |Z_i|.$$

Thus

$$\sum_{i=0}^{M-1} b_i = |Z|.$$

If V_2 accepts, then for every i the challenged leaf (i, γ_i) must belong to Z. Since the γ_i are sampled independently and uniformly from $[0, N/M - 1]$, we obtain

$$\Pr[\mathsf{V}_2 \text{ accepts}] \leq \prod_{i=0}^{M-1} \frac{b_i}{N/M}.$$

Using the AM–GM inequality (which states that for any $x_1, \ldots, x_t \in \mathbb{R}^{\geq 0}$ we have $\left(\prod_{i=1}^{t} x_i\right) \leq \left(\frac{\sum_{i=1}^{t} x_i}{t}\right)^t$)

$$\begin{aligned}\prod_{i=0}^{M-1} \frac{b_i}{N/M} &\leq \left(\frac{\sum_{i=0}^{M-1} b_i/(N/M)}{M}\right)^M \\ &= \left(\frac{|Z|}{N}\right)^M \\ &\leq \left(\frac{s+M\cdot p}{N}\right)^M . \end{aligned} \tag{11}$$

Finally, adding the probability of the bad event excluded by Lemma 1 gives

$$\left(\frac{s+M\cdot p}{N}\right)^M + \frac{\log(N)\cdot q^2}{2^\lambda},$$

as claimed. □

Numerical example. Let $s' = s + M \cdot p$, as in practice $M \cdot p \ll s$, think of s' as the effective bad-leaf budget appearing in the soundness bound. Consider parameters $s' = 2^{30}, M = 2^4 = 16, N = 2^{37}$, which roughly correspond to the parameters we chose in the numerical example for the Fischlin PoCK (note that $N' = 2N-1 = 2^{38}-1$ is the number of hash queries required by the prover, while in §2.3 this was $M \cdot 2^d = 2^{38}$). The bound in eq.(11) becomes $(2^{30}/2^{37})^{16} = 2^{-112}$, thus similar to the 2^{-105} we got for the PoCK in §2.3.

But as discussed in §2.3, the real advantage comes from the fact that in practice we can choose a much smaller s for a SPoCK than for a PoCK, as in the former s counts rounds containing encumbered queries, so overall we expect this SPoCK to be much more efficient for a similar security level.

6.3 Zero-Knowledge

In this section we will prove the zero-knowledge property of the ZK-SPoCK from §5.

Theorem 2. *The* ZK-SPoCK *from §5 satisfies the honest-verifier zero-knowledge property in the random oracle model (as defined in §4.1).*

Proof. We must construct a simulator $\mathsf{Sim}^{\mathcal{H}}(\mathsf{par}, \rho, x)$ that outputs a transcript $((\phi, \mathbf{a}), \boldsymbol{\gamma}, \boldsymbol{\tau})$ that is indistinguishable from a real transcript; this is done as follows:

1. Sample a random challenge $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{M-1}) \leftarrow [0, N/M - 1]^M$
2. Using the simulator for the Σ protocol, for $i \in [0, M-1]$ sample tuples $(a_i, \gamma_i, z_{i,\gamma_i})$ and set $\mathbf{a} := (a_0, \ldots, a_{M-1})$.
3. For all i and $j \neq \gamma_i$, pick $z_{i,j}$ at random.
4. Pick a random $r \leftarrow\!\!\$\ \{0,1\}^\lambda$ and for all i, j let $r_{i,j} := \mathcal{H}(r, i, j)$.

5. Using the values above, compute all labels as in eq.(8).
6. Using the values computed so far, define $\boldsymbol{\tau}$ as in eq.(9).
7. Output $((\phi, \mathbf{a}), \boldsymbol{\gamma}, \boldsymbol{\tau})$

This gives a valid transcript by construction, i.e., $\mathsf{V}_2^{\mathcal{H}}(\mathsf{par}, \rho, x, (\mathbf{a}, \phi), \boldsymbol{\gamma}, \boldsymbol{\tau}) = \mathsf{acc}$. But this transcript is of course not correctly computed, as for all $(i, j), j \neq \gamma_i$, the labels of the leaf nodes are computed using a wrong $z_{i,j}$, i.e., $(a_i, j, z_{i,j})$ will be rejected by V_Σ. But even a distinguisher that knows the witness (but makes $\ll 2^\lambda$ queries to $\mathcal{H}$) will not be able to distinguish this simulated transcript from a real transcript. The reason is that, while such a distinguisher can compute the correct $z_{i,j}$, it will not be able to verify if the leaf label that used $z_{i,j}$ as input was computed correctly, as this input also includes the random value $r_{i,j} \in \{0,1\}^\lambda$. □

7 TEEs

The SPoCK constructed in this paper requires a long sequential computation, which makes it hard to encumber when using MPC. Unfortunately, it provides less security when considering the other main mechanism to encumber a computation, namely using trusted execution environments (TEE) like Intel's SGX enclaves, as those contain fairly fast cores.

On the other hand, trusted execution environments (TEEs) typically provide only a limited amount of protected memory, and accessing data beyond this region requires secure paging to untrusted memory, incurring substantial overhead. For instance, Intel SGX enclaves are restricted to a relatively small protected memory region (typically 64 to 256MB), and when the working set exceeds this size, encrypted paging leads to significant performance degradation. Prior work reports slowdowns of several times for memory-intensive workloads once paging is triggered [21]. Consequently, functions that require large working memory together with frequent accesses to pseudorandom locations are particularly inefficient to evaluate inside TEEs, as such access patterns defeat caching and cause repeated secure page swaps.

Our SPoCK is based on the PoSW from [7], which is based on the DAG as illustrated in Figure 1. This DAG has the property that its labelling can be computed using little memory: just $\log(N)$ labels must be stored at any point. This was a massive improvement of the original construction of a PoSW from [19], which was based on depth-robust graphs, and where the prover had to store most of the up to N' labels when computing the labelling. Interestingly, while this was a major disadvantage of their PoSW, depth-robust graphs later turned out to be the right tool to construct cryptographic primitives that provably require a lot of space or memory, like proofs of space [9] or memory-hard functions [1].

We can easily adapt our SPoCK to use the [19] PoSW instead of [7]. For this, we would start with a depth-robust DAG on N nodes $\{0,1\}^n$, and then put a complete binary tree (as a DAG) on top of it as illustrated in Fig.3. To compute the labelling of this DAG one first (sequentially) computes the N labels of the depth-robust DAG, and then (potentially in parallel) computes the labels of the

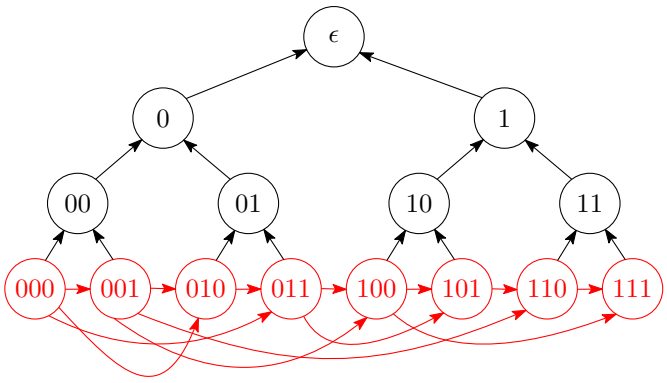

Fig. 3. The graph underlying the [19] PoSW. A depth-robust DAG (in red) with a full binary tree on top.

tree above (as in our construction, the label of the root now commits all the labels below).

The "space-hard" SPoCK constructed this way will inherit all the security properties of our construction, but it will now put a higher burden on (honest or malicious) provers, who must now use a large amount $\Theta(N \cdot \lambda)$ of memory (it's actually not known if depth-robust graphs really need that much space, or if they allow for some time-space trade off. To ensure we need that much space one should use a graph with high sustained-space complexity [2]). If this is too much for the honest prover, one can, e.g., concatenate T depth-robust graphs on N/T nodes each instead of using one big DAG on N nodes, this will then require in the order of $N \cdot \lambda/T$ space.

8 Conclusions

In this work we showed how proofs of complete knowledge, which prevent encumbrance of the secret by forcing the prover to make a huge number of oracle queries, can be combined with techniques from timed cryptography, and (in §7) also space-hard cryptography, to significantly harden the scheme against encumbrance, as now the queries must be done sequentially (which is really hard to do in MPC), and can additionally require a large memory with lots of frequent random accesses (which is hard for TEEs).

These ideas can likely also be applied to other primitives from individual cryptography. Our work opens the possibility that such primitives can be implemented on standard hardware, rather than relying on special hardware like Bitcoin ASICs in the existing constructions.

9 Acknowledgements

This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/F85. We used ChatGPT for grammar, spell checking, and basic language editing.

References

1. Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017, Part III. Lecture Notes in Computer Science, vol. 10212, pp. 3–32. Springer, Cham, Switzerland, Paris, France (Apr 30 – May 4, 2017) 4, 20
2. Alwen, J., Blocki, J., Pietrzak, K.: Sustained space complexity. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 99–130. Springer, Cham, Switzerland, Tel Aviv, Israel (Apr 29 – May 3, 2018) 21
3. Austgen, J., Fábrega, A., Kelkar, M., Vilardell, D., Allen, S., Babel, K., Yu, J., Juels, A.: Liquefaction: Privately liquefying blockchain assets. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025. pp. 1493–1511. IEEE (2025), https://doi.org/10.1109/SP61157.2025.00156 3
4. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing. pp. 111–120. ACM Press, Palo Alto, CA, USA (Jun 1–4, 2013) 2
5. Bormet, J., Dziembowski, S., Faust, S., Lizurej, T., Mielniczuk, M.: Strong secret sharing with snitching. In: Kalai, Y.T., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025, Part IV. Lecture Notes in Computer Science, vol. 16003, pp. 295–327. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2025) 3
6. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001) 2
7. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 451–467. Springer, Cham, Switzerland, Tel Aviv, Israel (Apr 29 – May 3, 2018) 4, 9, 10, 13, 14, 15, 16, 20
8. Dodis, Y., Guo, S., Katz, J.: Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 473–495. Springer, Cham, Switzerland, Paris, France (Apr 30 – May 4, 2017) 8
9. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 585–605. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015) 20
10. Dziembowski, S., Faust, S., Lizurej, T.: Individual cryptography. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 547–579. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023) 3

11. Dziembowski, S., Faust, S., Lizurej, T., Mielniczuk, M.: Secret sharing with snitching. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024: 31st Conference on Computer and Communications Security. pp. 840–853. ACM Press, Salt Lake City, UT, USA (Oct 14–18, 2024) 3
12. Feige, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing. pp. 210–217. ACM Press, New York City, NY, USA (May 25–27, 1987) 2, 5
13. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology – CRYPTO'86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 1987) 2, 5, 6
14. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 152–168. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005) 3, 7
15. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science. pp. 174–187. IEEE Computer Society Press, Toronto, Ontario, Canada (Oct 27–29, 1986) 2, 5
16. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th Annual ACM Symposium on Theory of Computing. pp. 291–304. ACM Press, Providence, RI, USA (May 6–8, 1985) 2
17. Kelkar, M., Babel, K., Daian, P., Austgen, J., Buterin, V., Juels, A.: Complete knowledge: Preventing encumbrance of cryptographic secrets. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024: 31st Conference on Computer and Communications Security. pp. 2415–2429. ACM Press, Salt Lake City, UT, USA (Oct 14–18, 2024) 3, 4
18. Kelkar, M., Ganesh, A., Partap, A., Bonneau, J., Weinberg, S.M.: Breaking omertà: On threshold cryptography, smart collusion, and whistleblowing. In: Huang, C., Chen, J., Shieh, S., Lie, D., Cortier, V. (eds.) Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025. pp. 3505–3519. ACM (2025), https://doi.org/10.1145/3719027.3765087 3
19. Mahmoody, M., Moran, T., Vadhan, S.P.: Publicly verifiable proofs of sequential work. In: Kleinberg, R.D. (ed.) ITCS 2013: 4th Innovations in Theoretical Computer Science. pp. 373–388. Association for Computing Machinery, Berkeley, CA, USA (Jan 9–12, 2013) 4, 9, 20, 21
20. Pass, R.: On deniability in the common reference string and random oracle model. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 316–337. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003) 3
21. Taassori, M., Shafiee, A., Balasubramonian, R.: VAULT: reducing paging overheads in SGX with efficient integrity verification structures. In: Shen, X., Tuck, J., Bianchini, R., Sarkar, V. (eds.) Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018. pp. 665–678. ACM (2018), https://doi.org/10.1145/3173162.3177155 20
22. Unruh, D.: Quantum proofs of knowledge. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 135–152. Springer Berlin Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012) 2