Space-Deniable Proofs*

Jesko Dujmovic
jesko.dujmovic@cispa.de
Helmholtz Center for Information Security and Saarland University

Christoph U. Günther
cguenthe@ista.ac.at
ISTA

Krzysztof Pietrzak
pietrzak@ista.ac.at
ISTA

May 6, 2026

Abstract

We introduce and construct a new proof system called *Non-interactive Arguments of Knowledge or Space* (NArKoS), where a space bounded prover can convince a verifier they know a secret, while having access to sufficient space allows one to forge indistinguishable proofs without the secret.

An application of NArKoS are space-deniable proofs, which are proofs of knowledge (say for authentication in access control) that are sound when executed by a lightweight device like a smart-card or an RFID chip that cannot have much storage, but are deniable (in the strong sense of online deniability) as the verifier, like a card reader, can efficiently forge such proofs.

We construct NArKoS in the random oracle model using an OR-proof combining a sigma protocol (for the proof of knowledge of the secret) with a new proof system called *simulatable Proof of Transient Space* (simPoTS). We give two different constructions of simPoTS, one based on labelling graphs with high pebbling complexity, a technique used in the construction of memory-hard functions and proofs of space, and a more practical construction based on the verifiable space-hard functions from TCC'24 where a prover must compute a root of a sparse polynomial. In both cases, the main challenge is making the proofs efficiently simulatable.

1 Introduction

Proofs of Knowledge. Many cryptographic primitives are based on some hard NP-relation $\mathcal{R}$, where given an instance x it's hard to find a witness w such that $(x, w) \in \mathcal{R}$ is in the relation. One example is the discrete logarithm (DL) assumption where $(h, w) \in \mathcal{R}$ if $h = g^w$ where g is some generator of a prime order group. For DL and similar assumptions there's an elegant proof system, due to Schnorr [Sch90], to show knowledge of the witness

*© IACR 2026. This article is a minor revision of the version published by Springer-Verlag available at https://doi.org/10.1007/978-3-032-12290-2_6.

in zero-knowledge. Such proofs can be directly used as an authentication protocol. One can also make them non-interactive via the Fiat-Shamir heuristic [FS87], this way getting a signature scheme.

Proofs of (Sequential) Work. Instead of proving that one knows some secret witness, some cryptographic schemes are based on proof systems that certify that the prover spent some physical resource. The best known example are proofs of work (PoW) due to their usage in the Bitcoin blockchain. The Bitcoin PoW is very simple: the prover, upon receiving a challenge c, must find a nonce π such that the hash $H(c, \pi)$ is below some difficulty, say it starts with d zeros. Assuming H behaves like a random function, an expected 2^d evaluations of H are necessary to find a proof.

Another popular resource for proof systems is *sequential* work, where the prover must perform some sequential computation. Here, having large parallelism should not allow the prover to find a proof any faster. Proofs of sequential work have found numerous applications as they can be used to enforce sufficient time has passed between the prover receiving a challenge, and them finding a proof.

OR-Proofs. The above-mentioned protocol by Schnorr to prove knowledge of a discrete logarithm is a so called Σ-protocol, which is a class of protocols which have particularly nice properties. In particular, the verifier must only send a uniformly random challenge $c \leftarrow\!\!\$\ \mathcal{C}$. Exploiting this structure, Cramer et al. [CDS94] showed how compile Σ protocols into "OR-proofs", where given two statements x_0, x_1, the prover can convince the verifier that they know a witness w such that either (x_0, w) *or* (x_1, w) is in the relation. The main idea is very simple: if the prover knows a witness for x_b, it simulates a transcript for x_{b-1}, which contains some challenge c_{b-1}. Now the prover and verifier basically run the protocol for x_0 and x_1 in parallel, but the verifier sends just one challenge c, and the prover can choose any challenges c_0 and c_1 as long as $c_0 \oplus c_1 = c$. Thus, they can use the simulated c_{b-1} while using $c_b = c \oplus c_{b-1}$ as the actual challenge in the proof of knowledge for x_b.

While originally the goal of or-proofs was to hide which of several witnesses a prover knows, one can use this idea to combine different protocols. In particular, one can combine a standard proof of knowledge with a proof that some resource was wasted. This was first done by Baldimtsi et al. [BKZZ16] who construct *Indistinguishable Proofs of Work or Knowledge* for proofs of work. Arun et al. [ABC22] construct *short lived proofs and signatures*, which combines a proof of knowledge with a proof of sequential work, concretely, a verifiable delay function (VDF).[1]

Short-lived Proofs. A short-lived signature as constructed by Arun et al. [ABC22] certifies that the signer either knew the secret signing key, or that they computed a VDF. By choosing the difficulty for the VDF so that computing it requires, say at least a day, one gets a signature which is sound for a day, but after that becomes deniable as anyone could have computed it by solving the VDF rather than using a secret signing key. While

[1] A verifiable delay function (VDF) is a proof of sequential work with the additional property that the output is unique. The uniqueness property is actually not relevant for their construction, but existing VDFs have nice algebraic structure which makes using them convenient in or-proofs.

one can achieve deniability for signatures in other ways, e.g., by using ephemeral keys that are regularly made public, short-lived signatures are appealing as they achieve deniability automatically after some time, without the signer having to take any action. Unfortunately, short-lived signatures or proofs do not achieve the strong notion of *online* deniability, where the judge (i.e., the party to be convinced by the verifier that an interaction with the prover took place) is in contact with the prover during protocol execution. Even in an offline attack, where the verifier must record some digital evidence that later will convince a judge, deniability of short-lived primitives can be broken if the prover has access to an unpredictable public randomness beacon and a time-stamping service (both are provided by blockchains): The verifier derives the challenge from the randomness beacon and then immediately time-stamps the short-lived proof. This convinces a judge as the prover could not have had enough time to forge the proof.

Space Deniable Proofs. In this work we construct what can be called space-deniable proofs. As short-lived proofs are only sound for some time (assuming they have not been time-stamped), space-deniable proofs are only sound if we assume the prover does not have large space. Concretely, for some space bound S which is a protocol parameter, the protocol is sound assuming the verifier has less than S space, while it's deniable if the judge cannot be convinced that the verifier has less than S space.

Such an assumption is realistic in settings where the proof is executed by some light-weight devices like smart-cards or RFID chips. For example, consider access control using smart-cards, here the smart card stores some secret key sk while the reader knows the public key pk. Smart-cards can only store a few hundred KB, so a reader interacting with the card can safely assume it doesn't have, say 100 MB. At the same time, a judge interacting with the reader cannot assume the reader does not have 100s of MB of memory, and thus could fake the interaction with the smart card without the secret-key, even in an online setting where the judge is communicating with the reader during protocol execution.

As another application consider product tracking using RFID chips. Using a space-deniable protocol will still allow verifying tags, but this tracking would not allow creating any digital evidence about the interaction with a tag.

Our space-deniable proofs are simple challenge response protocols, whether such protocols exist, i.e., "can a two-flow non-interactive protocol achieve online deniability" has been stated as an open problem in [CS20], and we answer this question affirmatively in the setting where the prover is space bounded.

Non-Interactive Arguments of Knowledge or Space (NArKoS). We call the new proof system allowing for space-deniable proofs as outline above *Non-Interactive Arguments of Knowledge or Space*, or NArKoS. The two main algorithms in this proof systems are Prove, which takes as input a public value with the witness $(x, w) \in \mathcal{R}$ (think of a public and secret key) and an auxiliary input aux (think of it as a random challenge), and outputs a proof π. The Verify algorithm, given as input x, aux and π, will accept such proofs.

Instead of using Prove, there's another algorithm Forge that can compute proofs without the witness w, but it requires significant space captured by a parameter S. The

proofs computed by Prove and Forge must be indistinguishable, in our constructions they will be statistically close.

Note that this proof system is zero-knowledge as Forge is basically a simulator as required by the definition of zero-knowledge, though it needs large space. For completeness, we also provide a highly efficient simulator Sim which samples proofs without knowing w and using only low space, but it is given the ability to select the challenge. Looking ahead, our construction relies on the Fiat-Shamir [FS87] heuristic and thus uses a random oracle. We emphasize that Forge does not program the random oracle, while Sim naturally needs to.

Simulatable Proofs of Transient Space. To construct NArKoS we follow the blueprint from [BKZZ16, ABC22], and combine a Σ-protocol with a proof system for the resource of interest, space in our case, through an OR-proof [CDS94]. To be used in an OR-proof, the proof system for space should satisfy the following properties.

Public-coin: The verifier sends just one uniformly random challenge.

Soundness: The honest prover needs S space, while no prover can make the verifier accept with somewhat less than S space.

Simulatable: Transcripts can be simulated using little space.

The last property is used in OR-proofs [CDS94] so one can't distinguish which of the two transcripts was created using the secret, and which one was simulated. In a NArKoS however, one of the two cases doesn't prove knowledge of anything, just access to space. Thus we require that there's a simulator that produces transcripts with little space. Much like the zero-knowledge simulators for Σ protocols, the simulator has the advantage that it does not have to compute the transcript in order.

Another property that is not needed for the OR-composition, but relevant for deniability, is that computing proofs should be highly parallelizable. That is, while computing a proof requires large space, it can be done quickly. This ensures that NArKoS does *not* also become a "short-lived proof" [ABC22], as then it would not be online deniable (i.e., deniable assuming the judge interacts with the verifier during the protocol). Even worse, it would not be offline deniable assuming the verifier can timestamp (as discussed above). Let us remark that it is generically possible to achieve deniability even if proof computation takes a long time: The prover, after receiving the challenge, must simply wait with his reply for the duration it would take the verifier to simulate a proof. If the computation is highly sequential, the resulting delay could make the protocol impractical.

We formally capture the above properties in a proof system *Simulatable Proofs of Transient Space* (simPoTS) in § 4. One caveat is that we must relax the public-coin property. The reason is that our main construction is based on the space-hardness of finding roots of sparse polynomials [DDJ24], which is not public-coin (we'll discuss this in detail later, but informally, the reason is that a random polynomial will not necessarily have a root). This is not an issue, however, as we can use the Fiat-Shamir heuristic with minor tweaks to get a NArKoS.

Interactive Arguments of Knowledge or Space (ArKoS). The applications of space-deniability we outlined above require interaction. For example, smart-card-based authentication relies on the verifier's interactive challenge to ensure that the smart-card must compute the proof at the time of authentication. In general, we assume that any application of space-deniable proofs will want the proofs to be bound to some specific time, interaction, session, etc. Otherwise, it cannot be ruled out that the space-constrained device just always responds with a single, precomputed proof that was forged on a space-unconstrained device.

So far, however, we have only constructed *non-interactive* ArKoS, which are not bound to any interaction by definition. Recall that the non-interactivity stems from our usage of a tweaked Fiat-Shamir heuristic, which we applied to an OR-proof [CDS94] of a sigma protocol and a simPoTS. To get an interactive ArKoS, one might hope to simply forgo the Fiat-Shamir heuristic and use the OR-proof directly. Unfortunately, this does not work:

In the interactive setting, the verifier picks an unpredictable, uniformly random challenge. The simPoTS, however, is not necessarily public-coin, so the challenge might not have a solution. This breaks space-deniability as no transcript can be forged for this challenge.

Fortunately, though, we can modify our NArKoS to get an interactive ArKoS. Recall that Fiat-Shamir derives the challenge using a random oracle. We can bind the resulting Fiat-Shamir challenge to an interaction by simply also inputting the interactive challenge to the random oracle. This results in a two-move protocol where the verifier interactively sends a uniformly random challenge, and the prover responds with a NArKoS that depends on the interactive challenge. Note that the interactive challenge may be replaced by some other unpredictable value that is sufficiently binding for the application (e.g., using the output of a public randomness beacon binds the proof to a certain time).

Summary and outline. What we have described so far certainly sounds a bit strange since we first construct a non-interactive protocol and only then make it interactive. To recap, our construction goes as follows:

1. We construct simulatable proofs of transient space (simPoTS), which may be private-coin (cf. § 4).

2. We then combine simPoTS with any sigma protocol using OR-proofs [CDS94]. The resulting protocol almost looks like a sigma protocol, but does not due to simPoTS having private-coin challenges.

3. To handle these private-coin challenges, we apply a tweaked version of Fiat-Shamir, resulting in a non-interactive protocol (cf. § 7).

4. We then turn the protocol interactive again by using the interactive challenge (or any other unpredictable value) as an extra input to the random oracle used for Fiat-Shamir (cf. § 7.5).

We follow this idiosyncratic route because it allows us to use an efficient simPoTS construction. In this work, we give two simPoTS constructions. The first is based on labelling hard-to-pebble graphs (cf. § 5). While it is public-coin (and would thus directly yield an interactive ArKoS via an OR-proof), it unfortunately has fairly large proofs,

making it impractical. Our second construction is based on the verifiable space-hard functions of Döttling et al. [DDJ24] (cf. § 6). It is much more practical, but it is not public-coin, so we need to follow the above route.

Algebraic attack. We thank Salil Vadhan for pointing out an attack against the space-hard function by Döttling et al. [DDJ24]. A subexponential adversary does not require the conjectured amount of space (cf. § 6 for details). Note that this does not disprove Döttling et al.'s assumption (cf. Ass. 1) as it considers a polynomial-time adversary. Still, the attack puts the assumption on shaky ground, so we do not recommend using our second construction.

2 Technical overview

We will now give a short technical outline of our two simulatable Proof of Transient Space (simPoTS) constructions. While the first (based on graph labeling) is public-coin, the second (based on finding roots of spare polynomials) is private-coin. We then show how a simPoTS is complied into a Non-interactive Argument of Knowledge or Space (NArKoS).

2.1 Pebbling-Based simPoTS

The first natural simPoTS construction builds on the ideas of [ABFG14, RD16], who build proofs of transient space from hard-to-pebble directed acyclic graphs (DAGs). Without going into details, this means the following. Consider assigning a label ℓ_v to every vertex v:

$$\ell_v = \mathcal{H}(v, \ell_{p_1}, \ldots, \ell_{p_d}) \tag{1}$$

where $\mathcal{H}$ is a random oracle and the ℓ_{p_i} are the labels of v's parents. If the DAG is space-hard, then any algorithm computing the labels of sufficiently many sinks must use a lot of space.

Omitting some details, [ABFG14, RD16] turn hard-to-pebble graphs into a PoTS in the following way:

1. The prover labels the graph using this random oracle and sends a Merkle commitment to all labels to the verifier.

2. The verifier challenges the prover to open the labels of some randomly chosen vertices, together with the labels of their parents. The number of requested vertices depends on the specific graph.

3. The prover responds with the openings, and the verifier checks that the labels of these vertices were computed according to Eq. (1).

The check in Item 2 ensures that the committed labels are "mostly correctly computed", and in this case, the space hardness of the graph implies that computing the labels required significant space.

This protocol, however, is not simulatable because the prover's first message depends only on the random oracle and the graph, and computing it requires large space by

construction. Moreover here a simulator cannot simply output a wrong commitment, as a distinguisher could simply recompute it and check whether it's correct. We fix this by slightly modifying the construction and randomizing the labels of the graph. In particular, we change Eq. (1) to $\ell_v = \mathcal{H}(v, r_v, \ell_{p_1}, \ldots, \ell_{p_d})$ where $r_v \leftarrow\!\!\$\ \{0,1\}^{2\lambda}$.

This turns the Merkle commitment from being perfectly binding to (almost) perfectly hiding. Now the proof is simulatable because opening some labels only reveals "local" information that cannot be checked against the unopened labels. So the simulator, who is given the verifier's challenge vertices in advance, can sample fake labels for the vertices to be opened that are only locally consistent with the graph. This does not pebble the graph, and is possible in low space. The simulator then produces a Merkle commitment where it can only open these fake labels. This is also possible in low space by randomly sampling all labels of intermediary vertices in the Merkle tree that are part of opening proofs for the fake labels.

2.2 First attempt at ArKoS

With the pebbling-based simPoTS we can create an ArKoS for an NP-relation $\mathcal{R}$. We follow prior knowledge-or-resource constructions [BKZZ16, ABC22] and use the technique of Cramer et al. [CDS94] to prove OR-statements. Given a statement x of $\mathcal{R}$, it proceeds as follows:

1. The prover sends first messages a_1 and a_2 of the sigma protocol and the simPoTS, respectively.
2. The verifier responds with uniformly random challenge c.
3. The prover responds with c_1 and c_2 such that $c_1 \oplus c_2 = c$, and third messages z_1 and z_2. The verifier checks that (a_1, c_1, z_1) verifies as a sigma protocol transcript for the statement x and that (a_2, c_2, z_2) is a valid simPoTS transcript.

By construction, the prover has control over either c_1 or c_2 but not both. This means, it can either simulate the sigma protocol or the simPoTS transcript in Item 1 and honestly execute the other one. Hence, in terms of completeness, a prover may succeed if it knows the witness w for the statement x, or if it uses a lot of space.

To see why this is also sound, consider a low-space adversary $\mathcal{A}$ that wins with probability 1 (for simplicity). We will use $\mathcal{A}$ to either extract a witness for x or break the space-hardness of simPoTS. The reduction runs $\mathcal{A}$ twice and uses rewinding. In the first run of $\mathcal{A}$ it gets transcripts (a_1, c_1, z_1) and (a_2, c_2, z_2) for the sigma and simPoTS protocol, respectively. Then, the reduction receives the challenge c^* from the simPoTS game. It then rewinds $\mathcal{A}$ and runs with the challenge $c' = c_1 \oplus c^*$. This results in transcripts (a'_1, c'_1, z'_1) and (a'_2, c'_2, z'_2). If $c_1 \neq c'_1$ we get a witness by the sigma protocol's special soundness. Otherwise, $c_1 = c'_1$, so $c'_2 = c' \oplus c'_1 = c^*$ by construction, which allows us to break simPoTS on challenge c^*.

2.3 Root-finding-based simPoTS

A major downside of the pebbling-based simPoTS is that the proofs are rather large. To get sufficient soundness from the simPoTS we need to open many labels committed to using the Merkle tree. For reasonable parameters, a proof will be several MB in size.

Our second simPoTS is based on the space-hardness of root-finding in sparse polynomials [DDJ24]. Their sparse root finding assumption almost directly implies a PoTS.

They conjecture that finding a zero in a polynomial

$$p(X) = X^d + a_0 + a_1 X + \sum_{i=2}^{\lambda} a_i X^{e_i}$$

over some exponentially-sized finite field $\mathbb{F}$, where $e_1, \ldots, e_\lambda \leftarrow\!\!\$\ [d-1]$, $a_0, \ldots, a_\lambda \leftarrow\!\!\$\ \mathbb{F}$ requires approximately as much space as storing a degree $d-1$ polynomial. The implied simPoTS is of the following form

1. The verifier samples a polynomial $p(X)$ as above and sends it to the prover.
2. The prover computes a root x (i.e., $p(x) = 0$) and sends x to the verifier.
3. The verifier checks whether $p(x) = 0$.

However, this has two issues. First, it is not complete as $p(X)$ may have no roots; we can only prove that $p(X)$ has a root with constant probability. Second, we cannot simulate it without using a lot of space.

To fix these issues, we consider a private-coin challenger. It samples $p'(X)$ as above, computes $y' = p'(x)$ for $x \leftarrow\!\!\$\ \mathbb{F}$, and outputs $p(X) = p'(X) - y'$. This has at least one root, namely x, and while it changes the distribution, it does not affect the space-hardness [DDJ24, Lem. 15]. Furthermore, the simulator can sample the challenge $p(X)$ in the same way. This way they sample a challenge together with a solution.

2.4 Handling private-coin simPoTS

Since simPoTS might be private-coin, we cannot create an ArKoS as above anymore. When forging the proof using a lot of space, the challenge c_2 will be uniformly random (and thus not identically distributed to private-coin challenger outputs). We circumvent this issue by requiring that simPoTS has the following property: Given a list of uniformly-random challenges, there exists an efficient challenge-selection algorithm that outputs a challenge from this list, and the challenge it outputs is identically-distributed to challenges output by the private-coin challenger. When using Fiat-Shamir, this allows us to create a non-interactive ArKoS where the forger can try different first messages until one simPoTS challenge is correctly distributed.

Conceptually, this is similar to *Fiat-Shamir with aborts* [Lyu09], which also tries different first message until the resulting transcript fulfills a certain condition. Since checking the condition in [Lyu09] just depends on the transcript, deciding whether to re-try does not require any state. In our case, however, the condition of whether the challenge is correctly distributed *does* require keeping state.

Intuitively, the challenge-selection algorithm first samples a reference polynomial using the private-coin challenger, counts its roots, and then iterates over the list of uniformly-random challenges until it finds one with the same number of roots. This suffices to ensure that challenges output by private-coin challenger and the challenge-selection algorithm have the same distribution.

2.5 Knowledge Soundness

Prior work [BKZZ16, ABC22] analyzed the following version of knowledge soundness: If the (malicious) prover convinces the verifier with non-negligible probability, then extraction is possible with non-negligible. They prove this by showing that if prover convinces the verifier with probability η, then extraction is possible with probability $\approx \eta^2$, which follows from a forking argument.

We improve upon this analysis by considering knowledge soundness in the sense of Bellare and Goldreich [BG93]: If the prover convinces the verifier on statement x with probability $\eta(x) > \kappa(\lambda)$, then there exists an expected polynomial time extractor that outputs a witness with probability at least $\eta(x) - \kappa(\lambda)$. Here, the extractor has rewinding black-box accesses to the prover and one step counts as one prover call. This definition is harder to show because of its tight connection between the prover's and the extractor's probability. In particular, it does not allow for the quadratic loss incurred above.

We show in § 8 that our construction satisfies such a stronger notion of knowledge soundness. We will use a slight variation that allows us to capture preprocessing [DF02]. Mapped to the smart card example from before, it intuitively states: It is hard to create a smart card whose probability of convincing the reader η is sufficiently high (i.e., $\eta > \kappa$), but the extraction only succeeds with low probability (i.e., less than $\eta - \kappa$).

3 General preliminaries

3.1 Notation

λ is the security parameter, which we usually use implicitly. $\log(\cdot)$ denotes the logarithm in base 2, $|\cdot|$ the size of a set or bit length of an object, $[n] = \{1, \dots, n\}$, and $[a, b] = \{a, a+1, \dots, b\}$.

$x \leftarrow\!\!\$\ \mathcal{X}$ denotes sampling x uniformly at random from the set $\mathcal{X}$. $x \leftarrow \mathsf{A}(\cdots)$ denotes the output of a probabilistic algorithm, $x = \mathsf{A}(\cdots)$ the output from a deterministic one. We write $x \in \mathsf{A}(\cdots)$ to describe all x that are in the support of a probabilistic algorithm $\mathsf{A}(\cdots)$.

For two distributions, we write $\approx_c$ or $\approx_s$ if they are computationally or statistically close, respectively. In definitions, we use $\approx_{s/c}$ if both are acceptable. If the distributions are identical, we use $\equiv$.

We use the usual Landau notation (e.g., $O(\cdot)$) and shorthands such as $\mathrm{poly}(\cdot)$, $\mathrm{negl}(\cdot)$, and $\text{non-negl}(\cdot)$. We typeset vectors in bold $\boldsymbol{v} = (v_1, \dots, v_n)$, sets in calligraphic letters (e.g., $\mathcal{X}$), and algorithms in sans-serif (e.g., A). Exceptions are the adversary $\mathcal{A}$ and the random oracle $\mathcal{H}$.

Commonly used abbreviations are: Polynomial time (PT), probabilistic polynomial time (PPT), random oracle (RO), and random oracle model (ROM).

3.2 Time and space

$\mathrm{Time}(\mathsf{A}(x))$ measures the wall-clock time of an algorithm A on input x. We allow the adversary to have parallelism p, mostly p being polynomial in λ. In this case, $\mathrm{Time}(\mathcal{A}(x))$ still measures the wall-clock time of $\mathcal{A}$ and *not* the total number of computation steps.

Space($\mathsf{A}(x)$) captures the maximum amount of space used by an algorithm A on input x. Here, also the input counts, i.e., Space($\mathsf{A}(x)$) $\geq |x|$. Also note that Space($\mathsf{A}(x)$) $\leq$ Time($\mathsf{A}(x)$) $\cdot\, p$.

We omit the input x when it is clear from context. Time($\mathsf{A}_1, \mathsf{A}_2$) $\leq T$ denotes that Time(A_1) $\leq T$ and Time(A_2) $\leq T$; similarly for Space($\mathsf{A}_1, \mathsf{A}_2$).

3.3 Protocols

In this work, we consider protocols where a prover interacts with a verifier. The verifier sends challenges from some challenge space $\mathcal{C}$. Throughout, $\mathcal{C}$ will be an additive group to allow for disjunctions (in the style of Cramer et al. [CDS94]) later on. We also define when a protocol is public or private coin.

Definition 1 (Challenge space)**.** A challenge space $\mathcal{C}(\lambda)$ is an additive finite Abelian group whose elements $c \in \mathcal{C}(\lambda)$ are of size at most $|c| \leq \text{poly}(\lambda)$.

Definition 2 (Public/private coin)**.** A protocol is *public-coin* if the verifier's randomness may be made public without affecting soundness. Conversely, a protocol is *private-coin* if the verifier's randomness must stay private.

Often, the prover proves knowledge of a witness w for a statement x of an NP-relation $\mathcal{R}$. We also define *hard* NP-relations, which means there is a way to sample statements $(x, w) \leftarrow \mathcal{R}(\lambda)$ such that finding w given x is hard on average. Note that we often omit the security parameter λ and just write $\mathcal{R}$, especially if hardness is not a concern.

Definition 3 (NP-relations)**.** We call a relation $\mathcal{R}(\lambda)$ an NP-relation if $|x| \leq \text{poly}(\lambda)$ and $|w| \leq \text{poly}(\lambda)$, and checking membership $(x, w) \in \mathcal{R}$ takes $\text{poly}(\lambda)$ time.

Overloading notation, let $(x, w) \leftarrow \mathcal{R}(\lambda)$ denote a PPT algorithm sampling statement-witness pairs such that $(x, w) \in \mathcal{R}(\lambda)$. We call $\mathcal{R}(\lambda)$ a *hard* NP-relation if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr[(x, w) \in \mathcal{R}(\lambda) \mid x \leftarrow \mathcal{R}(\lambda); w \leftarrow \mathcal{A}(x)] \leq \text{negl}(\lambda).$$

A sigma protocol for an NP-language $\mathcal{R}$ proves knowledge of a witness w given a statement x in zero-knowledge. It has three rounds (where a round is a message send from one party to the other) and it is public coin, which means the verifier uses public random coins to sample a uniform challenge from $\mathcal{C}$. The *first message* is a, the *challenge* c, and the *third message* z. Together (a, c, z) form a *transcript*.

Definition 4 (Σ-protocol)**.** A Σ-protocol for an NP-relation $\mathcal{R}$ is a three-round public-coin protocol. It is defined by the PPT algorithms $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V}, \mathsf{Sim})$ and a challenge space $\mathcal{C}(\lambda)$:

- $\mathsf{P}_1(x, w) \to a$: On input of a statement x and witness w, outputs a value a.
- $\mathsf{P}_2(x, w, c) \to z$: On input of a statement x, witness w, and challenge c, outputs a value z.
- $\mathsf{V}(x, a, c, z) = d$: On input of a statement x, value a, challenge c, and value z, deterministically outputs a decision $d \in \{\mathsf{acc}, \mathsf{rej}\}$.

- $\mathsf{Sim}(x, c) \to (a, z)$: On input statement x and challenge c, outputs the rest of a transcript (x, a, c, z).

A Σ-protocol fulfills the following properties:

- **Perfect completeness:** For any statement-witness pair $(x, w) \in \mathcal{R}$ it holds that

$$\Pr[\mathsf{acc} = \mathsf{V}(x, a, c, z) \mid a \leftarrow \mathsf{P}_1(x, w); c \leftarrow\!\!\$\ \mathcal{C}(\lambda); z \leftarrow \mathsf{P}_2(x, w, c)] = 1$$

  where the probability is taken over the choice of c, and the coins of $\mathsf{P}_1/\mathsf{P}_2$.

- **Special soundness** There exists a deterministic PT extractor Ext taking a statement x and two transcripts (a, c, z) and (a, c', z') with identical first message a but different challenges $c \neq c'$ that both verify (i.e., $\mathsf{V}(x, a, c, z) = \mathsf{V}(x, a, c', z') = \mathsf{acc}$) as input, and outputs a witness $w = \mathsf{Ext}(x, a, c, z, c', z')$ such that $(x, w) \in \mathcal{R}$.

- **Special honest verifier zero-knowledge (SHVZK)** There exists a PPT simulator $\mathsf{Sim}(x, c)$ such that, for any statement-witness pair $(x, w) \in \mathcal{R}$, it holds that the transcripts of simulated and real executions are computationally/statistically close, i.e.,

$$\left\{(a, z) \;\middle|\; \begin{array}{c} c \leftarrow\!\!\$\ \mathcal{C}(\lambda) \\ (a, z) \leftarrow \mathsf{Sim}(x, c) \end{array}\right\} \approx_{c/s} \left\{(a, z) \;\middle|\; \begin{array}{l} a \leftarrow \mathsf{P}_1(x, w) \\ c \leftarrow\!\!\$\ \mathcal{C}(\lambda) \\ z \leftarrow \mathsf{P}_2(x, w, c) \end{array}\right\}.$$

  We say the Σ-protcol has *computational/statistical SHVZK*.

4 Simulatable proofs of transient space

A simulatable proof of transient space (simPoTS) is a three-move interactive protocol. It convinces the verifier that the prover used a certain amount of space at some point during the interaction. It is simulatable in a sense similar to a sigma protocol's HVZK: There exists a simulator outputting transcripts that verify and are statistically-close to a transcript from a real interaction.

We will first define the primitive and then give two constructions. The first, dubbed GL (for Graph Labelling), is based on labeling space-hard graphs, like previous approaches to proofs of (transient) space (e.g., [DFKP15]). The second, dubbed SRF (for Sparse Root Finding) relies on the hardness of finding roots of sparse polynomials [DDJ24].

Unfortunately, SRF cannot be implemented in a public-coin manner. Intuitively, we can only guarantee that a uniform polynomial has a root with constant probability. Thus, we allow simPoTS to be private-coin, but require that it is reasonably efficient to pick an identically-distributed challenge from a list of uniformly random ones. Formally, there exists an algorithm with access to a list of uniformly random challenges, and that algorithm outputs a list index in expected PPT. Note that public-coin protocols (like GL) trivially fulfill this property.

4.1 Definition

Definition 5 (Simulatable proof of transient space)**.** A simulatable proof of transient space (simPoTS) is a three-round (potentially private-coin) protocol. It is defined by the algorithms $(\mathsf{Setup}, \mathsf{P}_1, \mathsf{C}, \mathsf{P}_2, \mathsf{V}, \mathsf{Sim}, \mathsf{CSel})$ and a challenge space $\mathcal{C} = \mathcal{C}(\lambda)$:

- $\mathsf{Setup}(S) \to \mathsf{pp}$: On input of a space bound S outputs public parameters pp.
- $\mathsf{P}_1(\mathsf{pp}) \to (\mathsf{st}, a)$: On input of public parameters pp outputs a state st and first message a.
- $\mathsf{C}(\mathsf{pp}) \to c$: On input of public parameters pp outputs a challenge $c \in \mathcal{C}$.
- $\mathsf{P}_2(\mathsf{pp}, \mathsf{st}, a, c) \to z$: On input of public parameters pp, state st, first message a, and challenge c, outputs a third message z.
- $\mathsf{V}(\mathsf{pp}, a, c, z) = d$: On input of public parameters pp, and transcript (a, c, z), deterministically outputs a decision $d \in \{\mathsf{acc}, \mathsf{rej}\}$.
- $\mathsf{Sim}(\mathsf{pp}) \to (a, c, z)$: On input of public parameters pp, outputs a transcript (a, c, z).
- $\mathsf{CSel}(\mathsf{pp}, \mathcal{L}) \to c$: On input of public parameters pp and a list $\mathcal{L}$, outputs a challenge c.

A simPoTS fulfills the following properties:

- **Perfect completeness:** For all polynomials $S(\cdot)$ and all λ, it holds that

$$\Pr\left[\mathsf{acc} = \mathsf{V}(\mathsf{pp}, a, c, z) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ (\mathsf{st}, a) \leftarrow \mathsf{P}_1(\mathsf{pp}) \\ c \leftarrow \mathsf{C}(\mathsf{pp}) \\ z \leftarrow \mathsf{P}_2(\mathsf{pp}, \mathsf{st}, a, c) \end{array}\right] = 1$$

  where the probability is taken over the randomness of C, P_1, and P_2.

- **Efficiency:** There exists a polynomial $\mathrm{poly}(\cdot)$ such that, for all polynomials $S(\cdot)$, it holds that $\mathrm{Time}(\mathsf{Setup}, \mathsf{V}, \mathsf{C}) \leq \mathrm{poly}(\lambda, \log S(\lambda))$ and $\mathrm{Time}(\mathsf{P}_1, \mathsf{P}_2) \leq \mathrm{poly}(\lambda, S(\lambda))$.

- **Space-hardness:** SimPoTS is space-hard with gap $0 < \varepsilon < 1$ if, for a sufficiently large polynomial $S(\cdot)$ and security parameter λ, all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ and $\mathcal{A}_2$ have $\mathrm{poly}(\lambda)$ parallelism and $\mathrm{Space}(\mathcal{A}_1), \mathrm{Space}(\mathcal{A}_2) \leq (1-\varepsilon) \cdot S(\lambda)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{acc} = \mathsf{V}(\mathsf{pp}, a, c, z) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ (\mathsf{st}_0) \leftarrow \mathcal{A}_0(\mathsf{pp}) \\ (\mathsf{st}_1, a) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{st}_0) \\ c \leftarrow\!\!\$\; \mathcal{C}(\lambda) \\ z \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{st}_1, a, c) \end{array}\right] \leq \mathrm{negl}(\lambda) \tag{2}$$

  where the probability is taken over the randomness of Setup, $c \leftarrow\!\!\$\; \mathcal{C}(\lambda)$, and $\mathcal{A}$.

- **Public challenge selection:** Let $\mathcal{L}\colon \mathbb{N} \to \mathcal{C}(\lambda)$ represent an infinite list of uniformly random challenges. There exists a polynomial $\mathrm{poly}(\cdot)$, such that, for every polynomial $S(\cdot)$ and $\mathsf{pp} \in \mathsf{Setup}(S(\lambda))$, it holds that

$$\mathbb{E}[\mathrm{Time}(\mathsf{CSel}(\mathsf{pp}, \mathcal{L}))] \leq \mathrm{poly}(\lambda, S(\lambda))$$

  and CSel outputs an integer $j \in \mathbb{N}$ such that

$$\{c \mid c \leftarrow \mathsf{C}(\mathsf{pp})\} \equiv \{\mathcal{L}(j) \mid j \leftarrow \mathsf{CSel}(\mathsf{pp}, \mathcal{L})\}.$$

- **Statistical honest-verifier simulatability:** There exists a polynomial $\mathrm{poly}(\cdot)$ such that, for all polynomials $S(\cdot)$ and $\mathsf{pp} \in \mathsf{Setup}(S(\lambda))$, it holds that

$$\mathrm{Time}(\mathsf{Sim}(\mathsf{pp})) \leq \mathrm{poly}(\lambda, \log S(\lambda))$$

  and

$$\{(a, c, z) \mid (a, c, z) \leftarrow \mathsf{Sim}(\mathsf{pp})\} \approx_s \left\{ (a, c, z) \;\middle|\; \begin{array}{r} (\mathsf{st}, a) \leftarrow \mathsf{P}_1(\mathsf{pp}) \\ c \leftarrow \mathsf{C}(\mathsf{pp}, a) \\ z \leftarrow \mathsf{P}_2(\mathsf{pp}, \mathsf{st}, c) \end{array} \right\}.$$

Remark 1 (Space-hardness). One may have noticed that we only require space-hardness for uniformly random challenges in Eq. (2). These are not necessarily close to the actual distribution produced by the challenger C. However, since we also require public challenge selection, the challenges produced by C cannot be too far from uniformly random ones to affect the hardness meaningfully.

Remark 2 (Public challenge selection). Public challenge selection may be viewed as a formalization of protocols that abort in a stateful manner. Usually, protocols with aborts are public-coin but have non-perfect completeness. When applying, e.g., Fiat-Shamir, the protocol is re-run until it succeeds. This requires that aborts are independent. Looking ahead, however, SRF (our construction based on polynomial root finding) requires stateful aborts to have the correct distribution.

5 Construction 1: Graph labeling

In this section, we describe a simPoTS based on graph labelling denoted by GL. Our construction essentially follows the construction paradigm for proof of (persistent/transient) space introduced by Dziembowski et al. [DFKP15]. The prover's first message is a Merkle commitment, and the challenge asks the prover to prove that $\mathrm{poly}(\lambda)$-many labels are consistent with the prescribed graph. To make this simulatable, we simply randomize each label.

5.1 Preliminaries

Commitments and Merkle trees. We will implicitly use the following *folklore commitments* in the random oracle model (ROM): $\phi = \mathcal{H}(r, m)$ is a commitment to message $m \in \{0,1\}^{\mathrm{poly}(\lambda)}$ with randomness $r \leftarrow\!\!\$\ \{0,1\}^{2\lambda}$ using the the random oracle (RO) $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$. To open ϕ to m it is sufficient to provide r. In the ROM, this

commitment is *computationally binding* (No $\mathcal{A}$ making a polynomial number of $\mathcal{H}$ queries can find tuples $(m, r) \neq (m', r')$ where $\mathcal{H}(m, r) = \mathcal{H}(m', r')$ except with negligible probability) and *statistically hiding* (ϕ reveals almost no information about m to even to an unbounded $\mathcal{A}$) [CY24, Lem. 17.1.1 & Lem. 17.3.1].

We will also use *Merkle trees* as vector commitments. In short, Merkle trees interpret each vector element as the leaf of a binary tree, and define non-leaf nodes to be the hash of its children. The root of the tree is the commitment ϕ. See, e.g., [CY24, §18] for a detailed description.

Merkle trees use a hash function which we model as a RO with λ bit outputs. In this case, they are *computationally binding* [CY24, Lem. 18.4.1]. Also, commitments are λ and proofs $O(\lambda \log n)$ bits for vectors of length n.

We will use the following syntax for Merkle trees, which allows us to open a set of indices I at once:

- $\mathsf{Merkle.Commit}(\boldsymbol{v}) \to \phi$ commits to a vector $\boldsymbol{v} = (v_1, \ldots, v_n)$.
- $\mathsf{Merkle.BatchOpen}(\boldsymbol{v}, \phi, I) \to \pi$ outputs a proof π attesting that v_i is the opening of ϕ at index i for all $i \in I$.
- $\mathsf{Merkle.BatchVerify}(\phi, \pi, (v_i)_{i \in I}, I) \to \{\mathsf{acc}, \mathsf{rej}\}$ verifies whether v_i is the opening of ϕ at index i for all $i \in I$.

Graphs and graph pebbling. Consider a direct acyclic graph (DAG) G with maximum indegree d. We denote its vertices by $V(G)$ and its sink vertices by $\mathrm{Sinks}(G)$. We denote the parents $(p_1, \ldots, p_i)$, $i \leq d$ of a vertex $v \in V(G)$ by $\mathrm{parents}_G(v)$.

Graph pebbling is an umbrella term comprising various games on a graph G. At each point a vertex is either pebbled or not. The game proceeds in rounds, where in each round one can place and/or remove pebbles from some vertices, but must adhere to the pebbling rules of the game. A sequence of such moves induces a *pebbling sequence* $\boldsymbol{P} = (P_0, \ldots, P_T)$ consisting of *pebbling configurations* $P_i \subseteq V(G)$, where P_i contains the vertices pebbled after round i. We say the pebbling sequence $\boldsymbol{P}$ *pebbles* v if $\{v\} \subseteq P_0 \cup \cdots \cup P_T$, i.e., vertex v is pebbled at some point.

The different pebbling games differ by which moves are allowed. In this paper, we consider the classical black pebbling game: Let $P_0 \subseteq V(G)$ be some initial pebbling configuration, and consider the move from P_i to P_{i+1}. In a move we may remove any number of pebbles, we can also place pebbles, but we may only pebble a node v if all its parents are pebbled in the current configuration, thus $P_{i+1} \subseteq P_i \cup \{v \ : \mathrm{parents}_G(v) \in P_i\}$. We may also bound the parallelism, i.e., the number of pebbles placed in a round, by some parameter p, in which case we add the extra rule $|P_{i+1} \setminus P_i| \leq p$.

Fixing a set of rules, a graph G and a game, it is sometimes possible to characterize pebbling sequences that win the game. For example, lower bounding the time T or the number of pebbles $\max_i |P_i|$ that *any* pebbling sequence must use. This makes graph pebbling useful for the design and analysis of cryptographic schemes, e.g., memory-hard functions (MHFs) [AS15] or proofs of space (PoSpace) [DFKP15].

Graph labeling. Cryptographic schemes should be implementable in practice, but pebbles are only mathematical objects. Thus, *labels* are used instead. Each $v \in$

$V(G)$ vertex has a label $\ell_v \in \mathsf{Labels}$. It is computed by applying a *labeling function* $L\colon V(G) \times \mathsf{Labels}^{\leq d} \to \mathsf{Labels}$ to the labels of v's parents, i.e., $\ell_v = L(v, \ell_{p_1}, \ldots, \ell_{p_i})$ where $(p_1, \ldots, p_i) = \mathrm{parents}_G(v)$ with $i \leq d$ (the maximum indegree of G).

A popular labeling function is $L(v, \ell_1, \ldots, \ell_i) = \mathcal{H}(v, \ell_1, \ldots, \ell_i)$ where $\mathcal{H}$ is a RO and $0 \leq i < d$. For certain games, pebbling lower bounds transfer directly to the ROM. Examples of such games capture the games used for data-independent memory-hard functions [AS15] (but not data-dependent ones [ACK+16]), memory-bound functions [DNW05], bandwidth-hard functions [RD17, BRZ18]. For PoSpace [Pie19, Fis19] pebbling lower bounds translate to the ROM if the metric is parallel time complexity [Pie19].

Remark 3. For simplicity, we will assume that L takes *exactly* d (i.e., the maximum indegree of G) labels as input, and that $\mathrm{parents}_G$ always returns *exactly* d vertices. This is without loss of generality as one can, e.g., define $\mathrm{parents}_G$ to output 0 for the remaining values and define $\ell_0 = 0 \cdots 0$.

Pebbling adversaries. Unfortunately, generically transferring pebbling lower bounds to the ROM is not known for all games relevant for cryptography. For example, data-dependent memory-hard functions [ACP+17] or PoSpace (counting overall cost) [DFKP15, RD16, Rey24]. Sometimes one can prove the desired bound in the ROM directly (e.g., [ACP+17, AGP24]), but often restricting the adversary is the best one can do using known techniques (e.g., [DFKP15, RD16, Rey24]).

The restriction essentially requires that the adversary may only store labels in their entirety instead of, e.g., the XOR of two labels. This means that even though the adversary plays a labeling game in the ROM, it acts as if it were playing the corresponding pebbling game instead. Thus, pebbling lower bounds directly apply to this restricted class of adversaries, and still rule out a large class of natural attack strategies.

Looking ahead, the game capturing simPoTS also has no known way of transferring pebbling lowerbounds to the ROM. Thus, we also restrict the adversary like prior works above.

Definition 6 (Pebbling adversary)**.** Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an adversary with access to a random oracle $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$. We say $\mathcal{A}$ is an adversary *in the pebbling model* if

- $\mathcal{A}_0$ adaptively performs Q queries $q_1, \ldots, q_Q$ to $\mathcal{H}$ and outputs a state st_0. It must be of the form $\mathsf{st}_0 = (\tau, h_1, \ldots, h_k)$ where $\tau \in \{0,1\}^*$ is an arbitrary bit string that *does not depend on* $\mathcal{H}$, and the $h_i \in \{0,1\}^\lambda$ are responses to $\mathcal{A}_0$'s queries, i.e., $\{h_1, \ldots, h_k\} \subseteq \{\mathcal{H}(q_1), \ldots, \mathcal{H}(q_Q)\}$.
- $\mathcal{A}_1$ proceeds in rounds making p queries in parallel. $\mathcal{A}_{1,i}(\mathsf{st}_{i-1})$ receives the state of the $i-1$th round and outputs a state $\mathsf{st}_{1,i}$. These states have the above form where the h_i must be query responses from any previous round.

This definition implies that a pebbling adversary $\mathcal{A}$ cannot do better than the corresponding pebbling lower bounds. That is, if any pebbling requires at least space of n pebbles, then $\mathcal{A}$ needs at least $n\lambda$ bits of space.

Space-hard graphs. We will rely on graphs that have high space complexity. Loosely speaking, pebbling a random sink requires a high amount of pebbles at some point with probability at least $1/2$.

Definition 7 (α-space-hard graphs)**.** Let $\{G_{n,\lambda}\}_{n,\lambda\in\mathbb{N}}$ be an efficiently sampleable family of DAGs with constant indegree d, n sources, n sinks, and $\mathrm{poly}(n,\lambda)$ vertices in total. For $\alpha \in [0,1]$, we call such a family *α-space-hard* if, with overwhelming probability over n,

- **α-Space-hardness:** For any pebbling configuration P_0 of size $|P_0| \leq \alpha n$, there exists a subset of sinks $U \subseteq \mathrm{Sinks}(G_{n,\lambda})$ of size $|U| \geq n/2$ such that, for any pebbling sequence $\boldsymbol{P} = (P_0, \ldots, P_T)$ with parallelism p, pebbling a sink $v \in U$, there either exists an $i \in [T]$ such that $|P_i| > \alpha n$, or $\boldsymbol{P}$ requires $T \cdot p \geq 2^\lambda$.[2]
- **Locally-navigable:** Computing the parents of a vertex $v \in V(G_{n,\lambda})$ takes at most $\mathrm{poly}(\lambda, \log n)$ time.

Such graphs have been used for PoPS and PoTS before, e.g., [ABFG14] used stacked superconcentrators [LT82]. A practical option are stacked expander graphs [RD16], which achieve this definition [RD16, Thm. 6]. They consist of $\lambda + 1$ layers with n vertices each where each layer is connected to the next one by a bipartite expander graph. This amounts to $(\lambda+1)n$ vertices total, and the structure allows pebbling one layer after another, so with $2n$ pebbles at most.

Remark 4. Making each layer a depth-robust graph improves constants [Fis19, Rey24] (i.e., the space-hardness guarantee is closer to the honest pebbling cost). These improvements are important for, e.g., proofs of replication [Fis19]. In our applications, better constants do not matter that much. For example, in space-deniable signatures one targets the amount of space S below which soundness should hold, so the signature forging algorithm will need space $(1/\alpha) \cdot S$. Since honest signers only simulate having space, they are not meaningfully affected by differences in α. Only signature forgers must expend more resources.

5.2 Construction

$\mathsf{GL.Setup}$ samples a graph $G_{n,\lambda}$ with $n = S(\lambda)/\lambda$. This graph belongs to α-space-hard family as defined above in Def. 7. For ease of notation, we call this graph G and its vertices $V = V(G)$. It also specifies a protocol constant $\delta < \alpha$. The smaller δ, the higher the space-hardness, but also the message size.

$\mathsf{GL.Setup}(S)$
$n = S(\lambda)/\lambda$ Fix any δ obeying $\delta < \alpha$ Sample a graph $G = G_{n,\lambda}$ (with vertices $V = V(G)$) Output $\mathsf{pp} = (G, \delta)$

The prover $\mathsf{GL.P}_1$ computes a labelling $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_{|V|})$ of this graph according to $\mathcal{H}$. In contrast to prior works (such as, e.g., [DFKP15]), these labels are randomized by including random values r_i. The prover then commits to $\boldsymbol{\ell}$, resulting in the Merkle commitment ϕ.

[2] In principle, we only require that $|U| \geq \beta n$ for some constant β. Since [RD16] achieves $\beta = 0.5$, we do not introduce another parameter to reduce clutter.

$\mathsf{GL.P}_1(\mathsf{pp} = (G, \delta))$

For node $i \in [V]$ in topographical order:
  $r_i \leftarrow\!\!\$\ \{0,1\}^{2\lambda}$
  $(p_1, \ldots, p_d) = \text{parents}_G(i)$
  $\ell_i = \mathcal{H}(i, r_i, \ell_{p_1}, \ldots, \ell_{p_d})$
Set $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_{|V|})$ and $\boldsymbol{r} = (r_1, \ldots, r_{|V|})$
$\phi = \mathsf{Merkle.Commit}(\boldsymbol{\ell})$
Output $\mathsf{st} = (\boldsymbol{\ell}, \boldsymbol{r})$ and $a = \phi$

The prover then receives a uniformly random challenge which consists of $\kappa \in \text{poly}(\lambda)$ many nodes, λ of which are sinks. Formally, the challenge space is $\mathcal{C}(\lambda) = V^\kappa \times \text{Sinks}(G)^\lambda$. Since the challenge is chosen uniformly at random and public-coin, $\mathsf{GL.CSel}(\mathsf{pp}, \mathcal{L})$ is trivial: It outputs the first entry of the list $\mathcal{L}$.

$\mathsf{GL.C}(\mathsf{pp} = (G, \delta))$

$\kappa = (\lambda|V|)/(\delta n)$
Sample and $(c_1, \ldots, c_\kappa) \leftarrow\!\!\$\ V^\kappa$ and $(c_{\kappa+1}, \ldots, c_{\kappa+\lambda}) \leftarrow\!\!\$\ \text{Sinks}(G)^\lambda$
Output $\boldsymbol{c} = (c_1, \ldots, c_{\kappa+\lambda})$

Given $\boldsymbol{c} = (c_1, \ldots, c_{\kappa+\lambda})$, the prover will output all the information showing that the labels ℓ_{c_i} are consistent with the commitment ϕ. That is, it opens ϕ at the position of all parents of c_i.[3]

$\mathsf{GL.P}_2(\mathsf{pp} = (G, \delta), \mathsf{st} = (\boldsymbol{\ell}, \boldsymbol{r}), a = \phi, \boldsymbol{c} = (c_1, \ldots, c_{\kappa+\lambda}))$

For challenge $i \in [\kappa + \lambda]$:
  $(p_1, \ldots, p_d) = \text{parents}_G(c_i)$
  Open $\pi_i = \mathsf{Merkle.BatchOpen}(\phi, \boldsymbol{\ell}, (c_i, p_1, \ldots, p_d))$
  $\boldsymbol{z}_i = (r_{c_i}, \ell_{c_i}, \ell_{p_1}, \ldots, \ell_{p_d}, \pi_i)$
Output $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{\kappa+\lambda})$

The verifier $\mathsf{GL.V}$ uses the information provided by the prover to check the consistency of all labels $\ell_{c_i}, \ldots, \ell_{c_{\kappa+\lambda}}$. It checks whether all openings of ϕ are correct, and whether the label was correctly computed using H.

$\mathsf{GL.V}(\mathsf{pp} = (G, \delta), a = \phi, \boldsymbol{c} = (c_1, \ldots, c_{\kappa+\lambda}), \boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{\kappa+\lambda}))$

For challenge $i \in [\kappa + \lambda]$:
  Parse $\boldsymbol{z}_i = (r_{c_i}, \ell_{c_i}, \ell_{p_1}, \ldots, \ell_{p_d}, \pi_i)$
  If $\mathsf{Merkle.BatchVerify}(\phi, \pi_i, (\ell_{p_1}, \ldots, \ell_{p_d}), \text{parents}_G(c_i)) = \mathsf{rej}$, output rej
  If $\ell_{c_i} \neq \mathcal{H}(c_i, r_{c_i}, \ell_{p_1}, \ldots, \ell_{p_d})$, output rej
Output acc

The simulator $\mathsf{GL.Sim}$ initially samples a list of challenges c_i. This allows it to cheat by only caring about the consistency of the labels corresponding to these challenges. To this

[3] In principle, proving consistency of $c_{\kappa+1}, \ldots, c_{\kappa+\lambda}$ is not necessary; the labels just have to be opened. While this would make the protocol concretely more efficient, we keep the presentation simple.

end, it iterates over the challenge indices in topological order, setting yet-to-be-defined labels to random values. This ensures that all labels that will be opened appear consistent. It then creates a Merkle commitment ϕ such that it can open these labels at the correct position (note that it cannot open any other position).[4] This suffices to simulate a transcript.

$\mathsf{GL.Sim}(G)$

01 $\boldsymbol{c} = c_1, \ldots, c_{\kappa+\lambda} \leftarrow \mathsf{GL.C}(G)$
02 For $c_i \in (c_1, \ldots, c_\kappa)$ in topographical order:
03 $\quad r_i \leftarrow \{0,1\}^{2\lambda}$.
04 $\quad (p_1, \ldots, p_d) = \text{parents}_G(c_i)$
05 $\quad$ For any $j \in [d]$, if ℓ_{p_j} is undefined: Define $\ell_{p_j} \leftarrow\!\!\$\ \{0,1\}^\lambda$
06 $\quad$ Define $\ell_{c_i} = \mathcal{H}(c_i, r_{c_i}, \ell_{p_1}, \ldots, \ell_{p_d})$
07 Compute Merkle commitment ϕ that is consistent with all defined ℓ_{c_i}
08 Compute opening proofs $\boldsymbol{z}_i$ for all defined ℓ_{c_i}
09 Output $(a = \phi, \boldsymbol{c}, z = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{\kappa+\lambda}))$

Theorem 1. GL *is a simPoTS as in Def. 5 when instantiating $G_{S(\lambda),\lambda}$ with an α-space-hard family of DAGs (Def. 7) and modelling $\mathcal{H}$ as a RO. Here, space-hardness holds against pebbling adversaries (cf. § 5.1) with a gap of $\varepsilon = (\alpha - \delta)$ where $\delta < \alpha$ is a constant.*

We prove Thm. 1 using Lems. 1 to 3 below.

Lemma 1. GL *is 1-complete and efficient.*

Proof. Completeness follows because the prover computes consistent labels for all vertices, and the verifier only checks that all the challenge vertices are labelled consistently.

For efficiency, note that G fulfills Def. 7, so it can be sampled efficiently and has $\text{poly}(S(\lambda), \lambda)$ many vertices. It follows that Setup and P_1 (especially the number of loop iterations) are efficient. By choice of $\kappa \in \text{poly}(\lambda)$, for a constant δ and the fact that Merkle commitments are constant size with logarithmic opening proofs, C, P_2 and V are also efficient. □

Lemma 2. GL *is statistically honest-verifier simulatable.*

Proof. First, note V cannot distinguish between P and Sim since all the labels V opens are consistent with ϕ by construction of Sim, and all labels are folklore commitments (§ 5.1), so the real and simulated labels are statistically indistinguishable. Second, we argue that Sim is efficient. Here, observe that computing Lines 2 to 6 only takes $\text{poly}(\lambda)$ time since it iterates $\kappa \in \text{poly}(\lambda)$ times and each parents_G call takes $\text{poly}(\lambda, \log S)$ time since G is locally-navigatable (Def. 7). During this, Sim only stores the labels $\kappa + \lambda$, corresponding randomness and (random) co-path values. In total, this is $\text{poly}(\lambda, \log(S(\lambda))$ since the tree has depth $\log(S(\lambda))$. □

Lemma 3. GL *is space-hard with gap $\varepsilon = (\alpha - \delta)$ for some constant $\delta < \alpha$ under the assumptions of Thm. 1.*

[4] We do not describe this in detail to keep the pseudocode simple.

GL is essentially a proof of persistent space as defined in [DFKP15, RD16] where everyone stores the same file. Thus, space-hardness essentially follows from their proofs. There is only one difference: Their labels are not randomized, so only one label is consistent with a commitment ϕ (assuming the Merkle tree's binding property is not violated). In contrast, GL's labels are randomized, but each label is a folklore commitment (§ 5.1), which is also binding. Thus, our proof essentially follows [DFKP15, RD16].

Proof of Lem. 3. Consider an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ that breaks space-hardness with gap ε with non-negligible probability (over the randomness of $\mathcal{A}$, Setup and C). We will show that this contradicts the space-hardness guarantee of G. By an averaging argument, we may consider $\mathcal{A}$ to be deterministic by fixing the randomness to some good value.

Assume for now that the commitment ϕ output by $\mathcal{A}_1$ is correct (i.e., as computed by P_1). If the $\mathcal{A}$ does not break the Merkle tree's or any of the labels'—recall they are folklore commitments—binding property, $\mathcal{A}$ can open ϕ to (at most) one label at index $i \in [n]$. We will assume that binding is not broken as this may happen with at most negligible probability.

Recall that the family of distributions on DAGs $\{G_{n,\lambda}\}_\lambda$ is α-space-hard. So, except for negligible probability, Setup returns a graph G where more than half of all sinks cannot be pebbled in less than 2^λ pebbling moves using less than αn pebbles where $n = S(\lambda)/\lambda$. In the pebbling model, accounting for the above assumption that ϕ is binding, α-space hardness implies that $\mathcal{A}_2$ pebbles a sink that is expensive to pebble except for probability $2^{-\lambda}$. Pebbling an expensive sink needs at least $\alpha n\lambda = \alpha S(\lambda)$ bits of space or at least 2^λ sequential time. Thus, $\mathcal{A}_2$, a polynomial time algorithm with polynomial parallelism, cannot break space-hardness (in the pebbling model) with more than negligible probability, assuming that the commitments are correct.

Now consider $\mathcal{A}_1$ potentially outputting malformed commitment $\tilde{\phi}$. As above, since folklore commitments and Merkle trees are binding, for every position $i \in [V(G)]$, $\mathcal{A}_1$ may open $\tilde{\phi}$ to (at most) one pair $\tilde{r}_i$ and $\tilde{\ell}_i$ for every $i \in [n]$ (except for negligible probability). We call vertex i *consistent with* $\tilde{\phi}$ if $\tilde{\ell}_i = \mathcal{H}(i, \tilde{r}_i, \tilde{\ell}_{p_1}, \ldots, \tilde{\ell}_{p_d})$ where the p_i are the parents of vertex i in G. Since $\mathcal{A}_2$ has to prove consistency of $\kappa = \lambda|V|/(\delta n)$ vertices chosen uniformly at random, at most δn vertices are not consistent with $\tilde{\phi}$ except with exponentially small probability $e^{-\lambda}$. Since the labels of these vertices might have concise representation, we give $\mathcal{A}_2$ this many pebbles for free [DFKP15, RD16]. As a consequence, $\mathcal{A}_2$ must use $(\alpha - \delta)n$ pebbles or $(\alpha - \delta)S(\lambda)$ bits space except for negligible probability. □

6 Construction 2: Finding roots of sparse polynomials

The following construction, which we denote by SRF, is based on the assumption of Döttling et al. [DDJ24] that finding roots of sparse polynomials is hard. It achieves better concrete parameters than GL.

6.1 Preliminaries

The assumption below follows from [DDJ24, Ass. 1 and Rem. 2].

Assumption 1 (Sparse Root-Finding [DDJ24, adapted])**.** We define the space-hardness of finding a root in a polynomial from distribution $\mathcal{D}_{\lambda,S}$ as follows: Root-Finding is hard with a gap $0 < \varepsilon < 1$ if, for all sufficiently large polynomials $S(\cdot)$ and PPT adversaries $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ with $\mathrm{Space}(\mathcal{A}_\lambda) \leq (1-\varepsilon)\cdot S(\lambda)$, there exists a negligible function negl such that

$$\Pr\left[p(x^*) = 0 \;\middle|\; \begin{array}{l} p(X) \leftarrow \mathcal{D}_{\lambda,S(\lambda)} \\ x^* \leftarrow \mathcal{A}_\lambda(p(X)) \end{array}\right] = \mathrm{negl}(\lambda)$$

where the probability is taken over the randomness of $\mathcal{D}$ and $\mathcal{A}$.

Remark 5. The definition of Döttling et al. [DDJ24] considers non-uniform adversaries. In the rest of the paper, we only consider uniform ones.

Candidate distribution. To make our assumption more concrete we state a candidate distribution $\mathcal{D}$ as suggested in [DDJ24, Def. 3]. Let $\mathbb{F}$ be a finite field of prime order $\Omega(2^\lambda)$ over which the polynomials are defined, and $d = \lceil S/\log|\mathbb{F}|\rceil$.

Distribution $\mathcal{D}_{\lambda,S}$

Set $d = \lceil S/\log|\mathbb{F}|\rceil$
Sample $e_2, \ldots, e_\lambda \leftarrow\!\!\$\ [d-1]^{\lambda-2}$ uniformly at random
Sample $a_0, \ldots a_\lambda \leftarrow\!\!\$\ \mathbb{F}^\lambda$ uniformly at random
Output the polynomial $X^d + a_0 + a_1 X + \sum_{i=2}^{\lambda} a_i X^{e_i}$

Subexponential algebraic attack. Root-finding of polynomials over finite fields $\mathbb{F}_q$ can be computed with an arithmetic formula of size $\mathrm{poly}(d, \log q)$, where d is the degree of the polynomial. In our scenario $\log q < d$. By Theorem 2.6 in [SY10] root finding can be computed with a formula of depth $O(\log d)$. A Turing machine can compute a formula of depth $O(\log d)$ in space $O(\log d)$ and time $O(2^{\log^2 d})$. Therefore there is a $O(2^{\log^2 d})$-time attack against the space-hardness of root finding.

6.2 Construction

Let $\mathcal{D} = \mathcal{D}_{\lambda,S}$ be a distribution over polynomials for which Ass. 1 holds. Note that $\mathsf{SRF.Setup}$ and $\mathsf{SRF.P}_1$ do nothing. This proof of transient space is a two message protocol. We keep the same notation for consistency's sake.

$\mathsf{SRF.C}$ samples a random point in the field and evaluates the polynomial $p(X)$ on it. This effectively defines the challenge polynomial $p(X) - y$.

$c \leftarrow \mathsf{SRF.C}()$

Sample $p(x) \leftarrow \mathcal{D}$
Sample $x \leftarrow\!\!\$\ \mathbb{F}$ uniformly at random
Let $y \leftarrow p(x)$
Output $(p(X), y)$

$\mathsf{SRF.CSel}$ initially runs $\mathsf{SRF.C}$ and computes the number of preimages. It then iterates over the uniformly random tuples $(p(X), y)$ in the list and outputs it if y has the same number of preimages relative of $p(X)$.

$c \leftarrow \mathsf{SRF.CSel}(\mathcal{L})$
01 $(p'(X), y') \leftarrow \mathsf{SRF.C}()$ 02 Compute R', the set of preimages of y' with respect to p', i.e., the roots of $p'(X) - y'$ 03 Initialize $i = 1$ 04 Loop: 05 Parse $(p(X), y) = \mathcal{L}(i)$ 06 Compute R, the set of preimages of y with respect to p, i.e., the roots of $p(X) - y$ 07 If $\lvert R' \rvert = \lvert R \rvert$ output i

The prover $\mathsf{SRF.P}_2$ outputs a root of the challenge polynomial $p(X) - y$. Note that such a root exists by construction of SRF.C.

$\mathsf{SRF.P}_2(c = (p(X), y))$
01 Compute R, the set of preimages of y with respect to p. I.e. $R = p^{-1}(y)$ 02 If $\lvert R \rvert = 0$ output $\perp$ 03 Sample $z \leftarrow\!\!\$\ R$ uniformly at random 04 Output z

The verifier SRF.V checks whether the prover's output is actually a root of the challenge polynomial.

$\mathsf{SRF.V}(p(X), y, x)$
01 Ensure $p(x) \stackrel{?}{=} y$, else output rej 02 Output acc

SRF.Sim the simulator picks a random point and evaluates the polynomial $p(X)$ on it. This procedure is almost the same as SRF.C.

$\mathsf{SRF.Sim}()$
01 Sample $p(x) \leftarrow \mathcal{D}$ 02 Sample $x \leftarrow\!\!\$\ \mathbb{F}$ uniformly at random 03 Let $y = p(x)$ 04 Output $(a = \perp, c = y, z = x)$

Theorem 2. SRF *is a complete simPoTS as in Def. 5 when instantiating* $G_{S(\lambda),\lambda}$.

Lemma 4. SRF *is space hard under the condition that sparse root finding is hard for the distribution* $\mathcal{D}$ *(Ass. 1).*

Proof. From [DDJ24, Lem. 15] follows that if an adversary compute a root for $p(X) - p(x)$ over random choice of x with probability ε then it can compute a root for $p(X) - y$ with probability $\geq \varepsilon/d$ over the random choice of y. Hardness of computing the root of $p(X) - y$ follows directly from Ass. 1. □

Lemma 5. SRF *is complete.*

Proof. Completeness follows because by definition of $\mathsf{SRF.C}$ there exists an $x \in \mathbb{F}$ such that $p(x) = y$ for the challenge y. □

Lemma 6. SRF *has public challenge selection.*

Proof. This follows from Clms. 1 and 2 below. □

Claim 1. $\mathsf{SRF.CSel}$ *the expected runtime in polynomial in* d *and* λ.

Proof. Because $p(X)$ is a degree d polynomial it can have at most d roots. Therefore, the expected runtime of $\mathsf{SRF.CSel}$ is

$$\begin{aligned}&\mathbb{E}[\mathrm{Time}(\mathsf{CSel})]\\ =&\sum_{s\in[d]} \Pr_{x'\leftarrow\$\mathbb{F},p'\leftarrow\mathcal{D}}[|p'^{-1}(p'(x'))| = s] \cdot 1/\Pr_{y\leftarrow\$\mathbb{F},p\leftarrow\mathcal{D}}[|p^{-1}(y)| = s] \cdot \mathrm{Time}(\mathcal{R})\end{aligned}$$

where $\mathcal{R}$ is one run of the loop in Lines 4 to 7.

From $p(X)$ being a degree d polynomial and the polynomial identity lemma follows that for every $y \in \mathbb{F}$ there are $m \leq d$ different values $x_1, \ldots, x_m$ such that $p(x_1) = \cdots = p(x_m) = y$. Therefore, for any $s \in [d]$ we have

$$\Pr_{x'\leftarrow\$\mathbb{F}}[|p^{-1}(p(x'))| = s] \leq d \Pr_{y\leftarrow\$\mathbb{F}}[|p^{-1}(y)| = s],$$

and thus

$$\begin{aligned}&\mathbb{E}[\mathrm{Time}(\mathsf{CSel})]\\ \leq&\sum_{s\in[d]} \Pr_{x'\leftarrow\$\mathbb{F},p'\leftarrow\mathcal{D}}[|p'^{-1}(p'(x'))| = s] \cdot d/\Pr_{x\leftarrow\$\mathbb{F},p\leftarrow\mathcal{D}}[|p^{-1}(p(x))| = s] \cdot \mathrm{Time}(\mathcal{R})\\ =&d^2 \cdot \mathrm{Time}(\mathcal{R}).\end{aligned}$$

Because $\mathrm{Time}(\mathcal{R})$ is polytime in d and λ we have $\mathbb{E}[\mathrm{Time}(\mathsf{CSel})]$ is polynomial. □

Claim 2. $\mathsf{SRF.CSel}$*'s output is identically distributed to* $\mathsf{SRF.C}$*'s output.*

Proof. Fix any $p^*(X)$ in the support of $\mathcal{D}$, any $s \in [d]$, and any $y^* \in \mathbb{F}$ such that $|p^{*-1}(y^*)| = s$. We want to show that

$$\begin{aligned}&\Pr[p(X) = p^*(X) \wedge y = y^* \mid (p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L})]\\ =&\Pr[p(X) = p^*(X) \wedge y = y^* \mid (p(X), y) \leftarrow \mathsf{C}].\end{aligned}$$

Let $\mathcal{P} = \{y \mid y \in \mathbb{F} \wedge |p^{*-1}(y)| = s\}$. By construction of C, it follows that

$$\begin{aligned}&\Pr[p(X) = p^*(X) \wedge y = y^* \mid (p(X), y) \leftarrow \mathsf{C}]\\ =&\Pr\left[y = y^* \,\middle|\, \begin{array}{l}(p(X), y) \leftarrow \mathsf{C}\\ p(X) = p^*(X)\\ |p^{-1}(y)| = s\end{array}\right] \cdot \Pr\left[|p^{-1}(y)| = s \,\middle|\, \begin{array}{l}(p(X), y) \leftarrow \mathsf{C}\\ p(X) = p^*(X)\end{array}\right]\\ &\cdot \Pr[p(X) = p^*(X) \mid (p(X), y) \leftarrow \mathsf{C}]\\ =&\frac{1}{|\mathcal{P}|} \cdot \Pr\left[|p^{-1}(y)| = s \,\middle|\, \begin{array}{l}(p(X), y) \leftarrow \mathsf{C}\\ p(X) = p^*(X)\end{array}\right] \cdot \Pr[p(X) = p^*(X) \mid (p(X), y) \leftarrow \mathsf{C}]\\ =&\frac{1}{|\mathcal{P}|} \cdot \Pr\Big[|p^{-1}(y)| = s \mid (p(X), y) \leftarrow \mathsf{C}\Big] \qquad (3)\end{aligned}$$

where the second equality follows from the fact that there are $s \cdot |\mathcal{P}|$ equally-likely choices for x in Line 2 of C obeying $p^{*-1}(p(x)) = s$, and s of them map to y^*.

Similarly, by construction of CSel, it holds that

$$\begin{aligned}
&\Pr[p(X) = p^*(X) \wedge y = y^* \mid (p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L})] \\
=&\Pr\left[y = y^* \;\middle|\; \begin{array}{l}(p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L}) \\ p(X) = p^*(X) \\ |p^{-1}(y)| = s\end{array}\right] \cdot \Pr\left[|p^{-1}(y)| = s \;\middle|\; \begin{array}{l}(p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L}) \\ p(X) = p^*(X)\end{array}\right] \\
&\cdot \Pr[p(X) = p^*(X) \mid (p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L})] \\
=&\frac{1}{|\mathcal{P}|} \cdot \Pr\left[|p^{-1}(y)| = s \;\middle|\; \begin{array}{l}(p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L}) \\ p(X) = p^*(X)\end{array}\right] \\
&\cdot \Pr[p(X) = p^*(X) \mid (p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L})] \\
=&\frac{1}{|\mathcal{P}|} \cdot \Pr\Big[|p^{-1}(y)| = s \mid (p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L})\Big] \qquad (4)
\end{aligned}$$

where the second equality follows from fact that there are $|\mathcal{P}|$ many choices for y obeying $p^{*-1}(y) = s$, and each y is equally likely as $\mathcal{L}$ given to CSel contains uniformly random ys.

The claim now follows from Eqs. (3) and (4), and the fact that

$$\Pr\Big[|p^{-1}(y)| = s \mid (p(X), y) \leftarrow \mathsf{C}\Big] = \Pr\Big[|p^{-1}(y)| = s \mid (p(X), y) \leftarrow \mathsf{CSel}(\mathcal{L})\Big]$$

for any $s \in [d]$ by construction of CSel. □

Lemma 7. SRF *is simulatable.*

Proof. For $(a, c, z) \leftarrow \mathsf{Sim}()$ and $c' \leftarrow \mathsf{C}()$, $z' \leftarrow \mathsf{P}_2(c')$ we have (a, c, z) and $(\perp, c', z')$ follow the same distribution. $c = (p(X), y)$ and $c' = (p'(X), y')$ are sampled identically and z and z' are uniform under the condition that $p(z) = c$ and $p'(z') = c'$ respectively. □

7 Space-deniable non-interactive arguments

We will define **N**on-interactive **Ar**guments of **K**nowledge or **S**pace (NArKoS). They are zero-knowledge argument for some relation $\mathcal{R}$ that can be forged given sufficient space.

Our construction is essentially a disjunction (in the style of Cramer et al. [CDS94]) between a Σ-protocol and a simulatable proof of transient space made non-interactive using Fiat-Shamir [FS87]. This almost directly yields a space-deniable signature scheme when used with a hard NP-relation [FS87]. Previous resource-deniable designs used the same approach (work [BKZZ16] and sequential time [ABC22]), and our definition is inspired by theirs.

Note that we also include auxiliary data aux in our definition. We will later use this for signatures and interactive ArKoS.

7.1 Definition

Definition 8 (Space-deniable non-interactive argument)**.** A non-inter-active argument of knowledge or space (NArKoS) for an NP-relation $\mathcal{R}$ is defined by the algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Forge}, \mathsf{Verify}, \mathsf{Sim})$.

- $\mathsf{Setup}(S) \to \mathsf{pp}$: On input of a space bound S outputs public parameters pp.
- $\mathsf{Prove}(\mathsf{pp}, \mathsf{aux}, x, w) \to \pi$: On input public parameters pp, auxiliary data aux, statement x, and witness w, outputs a proof π.
- $\mathsf{Forge}(\mathsf{pp}, \mathsf{aux}, x) \to \pi$: On input public parameters pp, auxiliary data aux, and statement x, outputs a proof π.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{aux}, x, \pi) = d$: On input of public parameters pp, auxiliary data aux, statement x, and proof π, deterministically outputs a decision $d \in \{\mathsf{acc}, \mathsf{rej}\}$.
- $\mathsf{Sim}(\mathsf{pp}, \mathsf{aux}, x) \to \pi$: On input of public parameters pp, auxiliary data aux, and statement x, outputs a proof π.

A NArKoS fulfills the following properties:

- **Perfect completeness:** For all polynomials $S(\cdot)$, auxiliary data aux, and statement-witness pairs $(x, w) \in \mathcal{R}$ it holds that

$$\Pr\left[\mathsf{acc} = \mathsf{Verify}(\mathsf{pp}, \mathsf{aux}, x, \pi) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{pp}, \mathsf{aux}, x, w) \end{array}\right] = \mathsf{acc}$$

  where the probability is taken over the randomness of Setup and Prove.

- **Efficiency:** There exists a polynomial $\mathrm{poly}(\cdot)$ such that, for all polynomials $S(\cdot)$ and statement-witness pairs $(x, w) \in \mathcal{R}$ it holds that $\mathrm{Time}(\mathsf{Setup}, \mathsf{Verify}) \leq \mathrm{poly}(\lambda, \log S(\lambda))$ and $\mathrm{Time}(\mathsf{Prove}) \leq \mathrm{poly}(\lambda, \log S(\lambda))$.

- **Space-hard soundness:** NArKoS has space-hard soundness with space gap $0 < \varepsilon < 1$ if, for all sufficiently large polynomials $S(\cdot)$, auxiliary data aux, statement-witness pairs $(x, w) \in \mathcal{R}$, and all PPT adversaries $\mathcal{A}$ with $\mathrm{poly}(\lambda)$ parallelism, and $\mathrm{Space}(\mathcal{A}) \leq (1 - \varepsilon) \cdot S(\lambda)$, there exists a PPT extractor Ext with rewinding oracle access to $\mathcal{A}$ such that the following holds: If

$$\Pr\left[\mathsf{acc} = \mathsf{Verify}(\mathsf{pp}, \mathsf{aux}, x, \pi) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \pi \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{aux}, x) \end{array}\right] \geq \text{non-negl}(\lambda)$$

  for some non-negligible function $\text{non-negl}(\cdot)$, then $\mathsf{Ext}^{\mathcal{A}}(\mathsf{pp}, x)$ and outputs a witness w with non-negligible probability, i.e.,

$$\Pr\Big[(x, w) \in \mathcal{R} \mid w \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{pp}, x)\Big] \geq \text{non-negl}(\lambda)$$

  for some non-negligible function $\text{non-negl}(\cdot)$.

- **Honest-verifier zero-knowledge (HVZK):** There exists a polynomial $\mathrm{poly}(\cdot)$ such that, for all polynomials $S(\cdot)$, auxiliary data aux, and statements $(x, \cdot) \in \mathcal{R}$, it holds that Sim uses little space, i.e.,

$$\mathrm{Time}(\mathsf{Sim}(\mathsf{pp}, \mathsf{aux}, x)) \leq \mathrm{poly}(\lambda, \log S(\lambda)),$$

and it holds that the transcripts of simulated and real executions are computationally/statistically close, i.e.,

$$\left\{\pi \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \pi \leftarrow\!\!\$\ \mathsf{Sim}(\mathsf{pp}, \mathsf{aux}, x) \end{array}\right\} \approx_{c/s} \left\{(a, c, z) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \pi \leftarrow\!\!\$\ \mathsf{Prove}(\mathsf{pp}, \mathsf{aux}, x, w) \end{array}\right\}.$$

We say NArKoS has *computational/statistical HVZK*.

- **Forgeability:** For all polynomials $S(\cdot)$, auxiliary data aux, statement-witness pairs $(x, w) \in \mathcal{R}$, and $\mathsf{pp} \in \mathsf{Setup}(S(\lambda))$ it holds that

$$\mathbb{E}[\mathrm{Time}(\mathsf{Forge}(\mathsf{pp}, \mathsf{aux}, x))] \leq \mathrm{poly}(\lambda, S(\lambda)),$$

and the distributions produced by Prove and Forge are computationally/statis-tically close, i.e.,

$$\left\{\pi \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \pi \leftarrow\!\!\$\ \mathsf{Prove}(\mathsf{pp}, \mathsf{aux}, x, w) \end{array}\right\} \approx_{c/s} \left\{\pi \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \pi \leftarrow\!\!\$\ \mathsf{Forge}(\mathsf{pp}, \mathsf{aux}, x) \end{array}\right\}.$$

We say NArKoS has *computational/statistical space-deniability*.

7.2 Construction

Our construction Σ-OR is a disjunction of a sigma protocol Σ and a simPoTS $\mathsf{simPoTS}$, made non-interactive using Fiat-Shamir. Due to the disjunction, we require that Σ and $\mathsf{simPoTS}$ have the same challenge space $\mathcal{C}$ (we discuss the compatibility of our simPoTS constructions in § 7.4).

Setup generates $\mathsf{simPoTS}$ parameters.

Σ-OR.$\mathsf{Setup}(S)$
01 Output $\mathsf{pp} \leftarrow \mathsf{simPoTS.Setup}(S)$

Prove generates the first message a_1 of Σ and simulates a transcript (a_2, c_2, z_2) for $\mathsf{simPoTS}$. It derives a challenge c using $\mathcal{H}(\cdots, a_1, a_2)$, and defines the challenge $c_1 = c + c_2$. It resumes Σ on c_1 and outputs both transcripts.

Σ-OR.$\mathsf{Prove}(\mathsf{pp}, \mathsf{aux}, x, w)$
01 $a_1 \leftarrow \Sigma.\mathsf{P}_1(x, w)$ 02 $(a_2, c_2, z_2) \leftarrow \mathsf{simPoTS.Sim}(\mathsf{pp})$ 03 $c = \mathcal{H}(\mathsf{pp}, \mathsf{aux}, x, a_1, a_2)$ 04 $c_1 = c + c_2$ 05 $z_1 \leftarrow \Sigma.\mathsf{P}_2(x, w, c_1)$ 06 Output $\pi = (a_1, a_2, c_1, z_1, z_2)$

Forge is similar to Prove but instead executes $\mathsf{simPoTS}$ honestly, while simulating Σ. Since $\mathsf{simPoTS}$ might be private-coin, we have to rely on $\mathsf{simPoTS.CSel}$ to pick an appropriate challenge. To this end, we have to provide a list $\mathcal{L}$ that contains uniformly-random challenges. We lazily sample each element of $\mathcal{L}$ by simulating Σ on a new challenge $c_1 \leftarrow\!\!\$\ \mathcal{C}$, deriving the challenge $c = \mathcal{H}(\cdots)$, and defining $c_2 = c + c_1$. After CSel outputs a list index, we resume $\mathsf{simPoTS}$ with that challenge and output the transcripts.

Σ-OR.Forge$(\mathsf{pp}, \mathsf{aux}, x)$

$(\mathsf{st}, a_2) \leftarrow \mathsf{simPoTS.P}_1(\mathsf{pp})$

Lazily sample list $\mathcal{L}(i)$ defined as
  $c_1^{(i)} \leftarrow\!\!\$\ \mathcal{C}(\lambda)$
  $(a_1^{(i)}, z_1^{(i)}) \leftarrow \Sigma.\mathsf{Sim}(\mathsf{pp}, x, c_1^{(i)})$
  $c^{(i)} = \mathcal{H}(\mathsf{pp}, \mathsf{aux}, x, a_1^{(i)}, a_2)$
  Return $c_2^{(i)} = c^{(i)} + c_1^{(i)}$

$j \leftarrow \mathsf{simPoTS.CSel}(\mathsf{pp}, \mathcal{L})$
$z_1 \leftarrow \mathsf{simPoTS.P}_2(\mathsf{pp}, \mathsf{st}, c_2^{(j)})$
Output $\pi = (a_1^{(j)}, a_2, c_1^{(j)}, z_1^{(j)}, z_2)$

Verify simply checks that both transcripts verify.

Σ-OR.Verify$(\mathsf{pp}, \mathsf{aux}, x, \pi = (a_1, a_2, c_1, z_1, z_2))$

$c = \mathcal{H}(\mathsf{pp}, \mathsf{aux}, x, a_1, a_2)$
$c_2 = c + c_1$
If $\Sigma.\mathsf{V}(x, a_1, c_1, z_1) = \mathsf{simPoTS.V}(\mathsf{pp}, a_2, c_2, z_2) = \mathsf{acc}$ output acc, else rej

Sim picks a challenge $c \leftarrow\!\!\$\ \mathcal{C}$ and runs the simulators of simPoTS to get a challenge c_2. It then sets $c_1 = c + c_2$ and runs $\Sigma.\mathsf{Sim}(x, c_1)$. After programming the RO $\mathcal{H}(\cdots) = c$ it outputs the transcript.

Σ-OR.Sim$(\mathsf{pp}, \mathsf{aux}, x)$

$c \leftarrow\!\!\$\ \mathcal{C}(\lambda)$
$(a_2, c_2, z_2) \leftarrow \mathsf{simPoTS.Sim}(\mathsf{pp})$
$c_1 = c + c_2$
$(a_1, z_1) \leftarrow \Sigma.\mathsf{Sim}(\mathsf{pp}, x, c_1)$
Program $\mathcal{H}(\mathsf{pp}, \mathsf{aux}, x, a_1, a_2) = c$
Output $\pi = (a_1, a_2, c_1, z_1, z_2)$

Theorem 3. *If Σ is a Σ-protocol for an NP-relation $\mathcal{R}$ with challenge space $\mathcal{C}$ and* simPoTS *is a simulatable PoTS with the same challenge space $\mathcal{C}$, then Σ-OR is a space-deniable non-interactive argument in the random oracle model.*

We prove Thm. 3 using Lems. 8 to 11 below.

Lemma 8. *Under the assumptions of Thm. 3, Σ-OR is perfectly complete and efficient.*

Proof. Perfect completeness follows from Σ's perfect completeness and simPoTS's statistical honest-verifier simulatability. Efficiency follows from the simPoTS's efficiency and the fact that all of Σ's algorithms are in PPT. □

Lemma 9. *Under the assumptions of Thm. 3 and assuming that Σ-OR.Sim can program the random oracle $\mathcal{H}$, Σ-OR is computationally or statistically honest-verifier zero-knowledge depending on whether Σ is computationally or statistically SHVZK.*

Proof. Note that Σ-OR.Sim programs the RO $\mathcal{H}$ correctly to a uniformly random value. Its polynomial runtime and the computational (resp. statistical) indistinguishability of the distributions follows from Σ's SHVZK and simPoTS statistical honest-verifier simulatability. □

Lemma 10. *Under the assumptions of Thm. 3, Σ-OR is computationally or statistically space-deniable depending on whether Σ is computationally or statistically SHVZK.*

Proof. First, note that Σ-OR.Forge runs in expected PPT since Σ.Sim is a PPT algorithm and simPoTS.CSel runs in expected PPT.

Second, computational (resp. statistical) indistinguishability of the distributions follows from a series of hybrids: First, on the left-hand side, replace simPoTS.Sim by a non-simulated execution of the protocol. This is statistically close by simPoTS's statistical honest-verifier simulatability. Second, replace the challenger simPoTS.C by simPoTS.CSel, and fix $c_1 \leftarrow\!\!\$\ \mathcal{C}(\lambda)$ first instead of c_2. This can be achieved by defining a list $\mathcal{L}'$ similar to $\mathcal{L}$ (Lines 3 to 7 in Σ-OR.Forge) but where Σ is still executed correctly (and not simulated). Since c_2 is distributed uniformly, simPoTS.CSel will output a $c_2^{(j)}$ that is distributed identically to the outputs of simPoTS.C by simPoTS's public challenge selection. Last, simulate Σ which is precisely the distribution on the right-hand side. This is computationally (resp. statistically) close by Σ's SHVZK. □

Lemma 11. *For sufficiently large polynomials $S(\lambda)$, if simPoTS is space-hard with gap ε, then there exists an $\varepsilon \lesssim \varepsilon' < 1$ such that Σ-OR has space-hard soundness with gap ε'.*

Proof. Suppose $\mathcal{A}$ produces an accepting proof π with non-negl. probability $\eta(\lambda)$ and makes at most $Q \in \mathrm{poly}(\lambda)$ queries to the random oracle $\mathcal{H}$. We construct an adversary $\mathcal{B}$ with black-box rewinding access to $\mathcal{A}$ that either extracts a witness w given only the statement x or breaks the space-hardness of simPoTS. Without loss of generality, we omit the auxiliary data aux in the remainder of the proof.

$\mathcal{B}(\mathsf{pp}, x)$ receives as input pp from the simPoTS game. It first fixes some randomness for $\mathcal{A}$, which it will use for both rewindings. To do this space efficiently, it picks $r \leftarrow\!\!\$\ \{0,1\}^\lambda$ and queries $\mathcal{H}(r, i)$ on-demand. In total, this requires $2\lambda + O(\log \lambda)$ bits (the latter to store the counter i).

Then, $\mathcal{B}$ runs $\mathcal{A}(\mathsf{pp}, x)$ once, proxying queries to the random oracle $\mathcal{H}$. It receives π which defines two transcripts (a_1, c_1, z_1) and (a_2, c_2, z_2). For now, assume that π verifies.

$\mathcal{B}$ uses a_2 to receive the simPoTS game challenge c_{simPoTS}, computes $\hat{c} = c_{\mathsf{simPoTS}} + c_1$ and discards all values except $\hat{c}$, a_1, a_2. This adds an overhead of $\mathrm{poly}(\lambda)$ bits.

It now rewinds and runs $\mathcal{A}(\mathsf{pp}, x)$ again with the same randomness. It also proxies queries to $\mathcal{H}$, except for programming the query $\mathcal{H}(\mathsf{pp}, \mathsf{aux}, x, a_1, a_2) = \hat{c}$. In the end, $\mathcal{A}$ outputs a new proof π'; again assume that π' verifies.

For now, assume that the transcripts defined by π' have the shape (a_1, c_1', z_1') and (a_2, c_2', z_2'), i.e., a_1 and a_2 are as in the previous transcript. We distinguish two cases: If $c_1 = c_1'$, then $c_2 = c_{\mathsf{simPoTS}}$ by our choice of $\hat{c}$. Since π' and thus z_2' verify $\mathcal{B}$ breaks space-hardness of simPoTS. If $c_1 \neq c_1'$, then $\mathcal{B}$ can extract a witness w for x by Σ's special soundness.

It is clear that $\mathcal{B}$ runs in PPT. Its space overhead over $\mathcal{A}$ is $2\lambda + O(\log \lambda) + \mathrm{poly}(\lambda)$ where the last term depends on the first message sizes and the challenge size of the constituent protocols. So if $S(\lambda)$ is sufficiently large, namely $(1 - \varepsilon) \cdot S(\lambda) > 2\lambda + O(\log \lambda) + \mathrm{poly}(\lambda)$,

then there exists an $\varepsilon' < 1$ such that is $\mathcal{B}$ uses at most $(1-\varepsilon') \cdot S(\lambda)$ space in total. That is, $\mathcal{B}$ is an adversary against $\mathsf{simPoTS}$ with space gap ε'.

Let us analyze the probability that $\mathcal{B}$ extracts a witness. The probability that π and π' verify, and that a_1 and a_2 stay the same is at least $\eta(\lambda)^2/Q$. This follows from the local forking lemma [BDL19]. Since breaking space-hardness of $\mathsf{simPoTS}$ happens with at most negligible probability, $\mathcal{B}$ extracts a witness with probability at least $\eta(\lambda)^2/Q - \mathrm{negl}(\lambda)$ which is non-negligible by assumption on $\eta(\cdot)$. □

7.3 Space-deniable signatures

The Σ-OR construction almost directly gives a space-deniable signature scheme using a common design pattern [FS87]: A public-private key pair is a statement-witness pair of a hard NP-relation, i.e., $(\mathsf{pk}, \mathsf{sk}) = (x, w) \leftarrow \mathcal{R}(\lambda)$. To sign a message m, run $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, \mathsf{aux} = m, x = \mathsf{pk}, w = \mathsf{sk})$ where $\mathsf{aux} = m$ importantly. Space-hard soundness of Σ-OR now implies that any algorithm that forges signatures with non-negligible probability allows us to recover the secret key, but this is not possible by the hardness of $\mathcal{R}$.

7.4 Challenge space compatibility

We assumed that the challenge spaces of Σ and $\mathsf{simPoTS}$ are equal, but the challenge spaces of GL and $\mathsf{simPoTS}$ are rather non-standard. There exist generic composition methods, e.g., repeating Σ in parallel to increase $\Sigma.\mathcal{C}$'s size, or using primitives like partially-binding vector commitments [GGHAK22]. However, our schemes GL and SRF may be modified to be directly compatible with a wide range of Σ protocols.

$\mathsf{GL.C}$ samples a list of vertices, but these can be derived using a PRG. This has no impact $\mathsf{GL.Sim}$ as it can produce messages a and z given any challenge c.

Using a PRG for $\mathsf{SRF.C}$ does not work since $\mathsf{SRF.Sim}$ chooses the challenge itself. However, we can use two observations to do better. First, $\mathsf{SRF.Sim}$ does not need control over the full challenge, but only a part of it, namely $y \in \mathbb{F}$.[5] Second, $\mathsf{SRF.P}_1$ always outputs a fixed first message $a_1 = \bot$. Therefore, we can modify Σ-OR to derive two challenges $c \in \mathbb{F}$ and $c_p \in \mathcal{D}$ where only the first challenge c needs to be modified. This can be achieved using domain-separated ROs, e.g., $c = \mathcal{H}(0, \mathsf{pp}, \mathsf{aux}, x, a_1, \bot)$ and $c_p = \mathcal{H}(1, \mathsf{pp}, \mathsf{aux}, x, a_1, \bot)$. Thus, SRF is compatible with any sigma protocol with challenge space $\Sigma.\mathcal{C} = \mathbb{F}$. For example, $\mathbb{F} = \mathbb{Z}_q$ works for the discrete-logarithm relation (e.g., space-deniable Schnorr signatures).

7.5 Interactive arguments

A NArKoS may be turned into an interactive ArKoS generically, similarly to how we implemented signatures. The first message is empty, i.e., $a = \bot$, the verifier's challenge is $c \leftarrow\!\!\$\ \{0,1\}^\lambda$, and the third message is $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, \mathsf{aux} = c, x, w)$ or $\pi \leftarrow \mathsf{Forge}(\mathsf{pp}, \mathsf{aux} = c, x)$. In contrast to the NArKoS definition, this construction allows for space-unbounded PPT preprocessing (like the definition of $\mathsf{simPoTS}$, Def. 5). This is because c acts as a salt [DGK17].

[5] Recall that the challenge is a polynomial $p(X) \leftarrow \mathcal{D}$ over the field $\mathbb{F}$ and a coefficient $y \in \mathbb{F}$. The goal is to find a root of $p(X) - y$. Thus, $\mathsf{SRF.Sim}$ works for *any* polynomial $p(X)$ as long as it can control y.

One might wonder whether it is possible to also use the interactive version of the disjunction directly (i.e., without Fiat-Shamir). In general, the answer is "no" because it breaks deniability as we cannot implement Forge anymore. The reason is that simPoTS may be private-coin and only offer public challenge selection (i.e., we can select a correctly distributed challenge from a list of uniformly random challenges). While we can lazily sample such a list using Fiat-Shamir (cf. Lines 3 to 7 in Σ-OR.Forge), we only get a single fixed challenge in the interactive setting.

Let us note that if simPoTS is public-coin (e.g., the graph-labeling construction GL), then we can use the interactive disjunction directly. The properties and proofs are mostly unchanged except for two details:

First, the simulator now does not need to program the random oracle. It is not hard to see that Σ-OR.Sim achieves *special* HVZK as it works for any given challenge.

Second, while our only public-coin construction GL is in the ROM, this need not be the case in general. In the plain model, the proof of Lem. 11 (space-hard soundness) needs to be adapted. The reason is that it relies on the RO to keep the space overhead of the reduction low. This is not an issue, however, as we can replace the random oracle by a PRF. Auerbach et al. [ACFK17] used this technique in the context of memory-tight reductions (to analyze, e.g., RSA-FDH).

8 Knowledge Soundness

The space-hard soundness definition in Def. 8 follows prior works [BKZZ16, ABC22]. If a low-space (malicious) prover convinces the verifier of any statement with non-negligible probability, then it is possible to extract a witness with non-negligible probability. This implies standard soundness (i.e., a low-space prover cannot convince the verifier of false statements), and is also sufficient for signatures based on hard NP-relations. It is weaker, however, than the usual notion of knowledge soundness due to Bellare and Goldreich [BG93].[6]

Definition 9 (Knowledge soundness [BG93])**.** For some statement x, if a (malicious) prover $\tilde{\mathsf{P}}$ convinces the verifier V with probability $\eta(x) > \kappa(\lambda)$, then there exists an extractor Ext with rewinding access to $\tilde{\mathsf{P}}$ that outputs a witness w with $(x, w) \in \mathcal{R}$ in an expected number of steps $\mathrm{poly}(\lambda)/(\eta(x) - \kappa(\lambda))$. Here, an oracle call to $\tilde{\mathsf{P}}$ counts as one step.

Towards this definition, we could try rewinding as in the proof of Σ-OR's space-hard soundness. If the prover convinces the verifier with probability η, then this intuitively results in an extraction probability of $\approx \eta^2$. This is not sufficient, however, for the above definition, which requires an extraction probability of $\approx \eta$. To fix this, we need more elaborate rewinding techniques, and to use them, we need a stronger version of space-hardness, which we introduce next. Furthermore, for reasons we elaborate on later, we will use the knowledge soundness definition of Damgård and Fujisaki [DF02], which is closely related to that of Bellare and Goldreich [BG93].

In the remainder of this section we will consider interactive ArKoS (cf. § 7.5). In particular, the "direct" disjunctive composition of a Σ protocol with a simPoTS. As we

[6]Note that we stated their definition in the introduction in a different, yet equivalent form.

outlined in § 7.5, this only works with public-coin simPoTS (i.e., it works with GL but not SRF). These restrictions are without loss of generality. We only use them because the intuition behind the following definitions maps more nicely to the interactive setting, and because it allows us to keep the presentation simple (e.g., no RO programming).

8.1 Stronger space-hardness definition

The following definition differs from the previous space-hardness definition (cf. Def. 5) in two ways: First, it allows multiple interactions, and only requires that $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_\ell)$ wins one. Second, it concretely tracks the time and parallelism of $\mathcal{A}$. Like before, space-hardness is parameterized by the space-gap ε.[7]

In the context of the smart card example given in the introduction, this means the following: The adversary is allowed some space-unbounded precomputation to create a malicious smart card. Then, in the online phase, the adversary has multiple attempts to convince the reader using this malicious smart card.

Notation. Let us fix some notation that we will use throughout this section.

- $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \ldots)$ is the adversary where the number of adversaries will be clear from context. $\mathcal{A}_0$ performs *preprocessing*, which is not space-bounded, and all $\mathcal{A}_{i>0}$ are *online* adversaries, which are space-bounded.
- $t_{\mathsf{pre}} = t_{\mathsf{pre}}(\lambda)$ and $t_{\mathsf{on}} = t_{\mathsf{on}}(\lambda)$ denote the maximum time taken by $\mathcal{A}_0$ and $\mathcal{A}_{i>0}$, respectively. That is, $\mathrm{Time}(\mathcal{A}_0) \le t_{\mathsf{pre}}$ and $\mathrm{Time}(\mathcal{A}_1, \ldots) \le t_{\mathsf{on}}$.
- $p = p(\lambda)$ denotes the maximum parallelism of any $\mathcal{A}_i$, including $\mathcal{A}_0$.

Definition 10 (Strong space-hardness)**.** SimPoTS is strongly space-hard with gap $0 < \varepsilon < 1$ if, for a sufficiently large polynomial $S(\cdot)$ and all probabilistic adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{2\ell})$ with $\mathrm{Space}(\mathcal{A}_1, \ldots) \le (1-\varepsilon) \cdot S(\lambda)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that

$$\Pr\left[\exists i \in [\ell].\, \mathsf{acc} = \mathsf{V}(\mathsf{pp}, a_i, c_i, z_i) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)) \\ \mathsf{st}_0 \leftarrow \mathcal{A}_0(\mathsf{pp}) \\ (\mathsf{st}_1, a_1) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{st}_0) \\ c_1 \leftarrow\!\!\$\ \mathcal{C}(\lambda) \\ (\mathsf{st}_2, z_1) \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{st}_1, a_1, c_1) \\ \vdots \\ (\mathsf{st}_{\ell-1}, a_\ell) \leftarrow \mathcal{A}_{2\ell-1}(\mathsf{pp}, \mathsf{st}_{\ell-2}) \\ c_\ell \leftarrow\!\!\$\ \mathcal{C}(\lambda) \\ z_\ell \leftarrow \mathcal{A}_{2\ell}(\mathsf{pp}, \mathsf{st}_{2\ell-1}, a_\ell, c_\ell) \end{array}\right] \le \mathrm{negl}(\lambda)$$

where ℓ denotes the number of $\mathcal{A}$'s online interactions.

[7] We do not concretely track the success probability, but leave it negligible. This is to avoid clutter as we already have many parameters.

Since our construction SRF (based on finding roots of polynomials) has no pp and no first message a_1, the following lemma is a consequence of Ass. 1.

Lemma 12. SRF *fulfills strong space-hardness for any* t_{on} *and* $\ell \in \mathrm{poly}(\lambda)$*, and arbitrary* t_{pre}*.*

GL also enjoys strong space-hardness for large parameters, especially ℓ.

Lemma 13. *Let* t_{pre}*,* t_{on} *and* ℓ *be such that* $(p \cdot t_{\mathsf{pre}} + 2 \cdot p \cdot \ell \cdot t_{\mathsf{on}})^2/2^\lambda$ *is negligible in* λ*. Then* GL *has strong space-hardness with gap* $\varepsilon = (\alpha - \delta)$ *under the assumptions of Thm. 1.*

Proof (Sketch). This proof is similar to the regular space-hardness Lem. 3. There are four events that cause space-hardness to fail. In the following, let the number of RO queries be $Q \leq p \cdot t_{\mathsf{pre}} + 2 \cdot p \cdot \ell \cdot t_{\mathsf{on}}$ by assumption.

- Setup does not return an α-space-hard graph. By assumption, this only happens with negl(λ) probability.
- The Merkle tree and the folklore's commitment binding properties are broken. This happens with probability at most $1/2 \cdot Q^2/2^\lambda$ [CY24, Lem. 18.4.1], which is negligible by assumption.
- For any attempt $i \in [\ell]$ more than δn committed values are not consistent. Assuming binding holds, over the choice of c_i, this happens with probability at most $e^{-\lambda}$. Since $\ell < Q$, by a union bound, this is also negligible.
- For any attempt $i \in [\ell]$ all λ sinks are cheap to pebble. Over the choice of c_i, this only happens with probability $2^{-\lambda}$; again, negligible by a union bound.

If none of these events occurs, space-hardness holds. □

8.2 Knowledge-soundness of interactive Σ-OR

We first define knowledge soundness for ArKoS against an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$. Intuitively, using the smart card example from the introduction, one can think of $\mathcal{A}_0$ preparing a smart card, and $\mathcal{A}_1$ and $\mathcal{A}_2$ represent the computation of the smart card when inserted into the reader.

$$\left.\begin{aligned} \mathsf{pp} &\leftarrow \mathsf{Setup}(S(\lambda)) \\ \mathsf{st}_0 &\leftarrow \mathcal{A}_0(\mathsf{pp}, x) \end{aligned}\right\} \text{Smart card preparation}$$

$$\left.\begin{aligned} (\mathsf{st}_1, a) &\leftarrow \mathcal{A}_1(\mathsf{pp}, x) \\ c &\leftarrow\!\!\$\ \mathcal{C}(\lambda) \\ z &\leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{st}_2, a, c) \end{aligned}\right\} \text{Interaction within reader}$$

Knowledge-soundness should now tell us that we can extract a witness from any space-bounded smart card that convinces the reader with sufficiently high probability. This intuition mostly holds apart from one issue. For example, when using GL, consider that the graph sampled by GL.Setup is easy to pebble, which happens with at most negligible probability. Then even if the smart card convinces the reader with high probability, we cannot extract anything as we have no space-hardness guarantees.

To fix this issue, we will use the notion of knowledge soundness used by Damgård and Fujisaki [DF02]. It is similar to Bellare and Goldreich [BG93] but handles computational assumptions. To see how it does so, we introduce some definitions.

We will first define what a "view" is. It is a partial transcript up to st_0 (i.e., the prepared smart card). We then consider the remaining interaction (i.e., when the smart card is inserted into the reader) and define the winning probability of this view.

Definition 11 (View of $\mathcal{A}$)**.** For any polynomial $S(\cdot)$, statement x, public parameters $\mathsf{pp} \in \mathsf{Setup}(S(\lambda))$, and state $\mathsf{st}_0 \in \mathcal{A}_0(\mathsf{pp})$, we call $(x, \mathsf{pp}, \mathsf{st}_0) = \mathsf{view}$ a "view" of $\mathcal{A}$. Its winning probability $\eta(\mathsf{view})$ is defined as

$$\Pr\left[\mathsf{acc} = \mathsf{V}(\mathsf{pp}, a, c, z) \;\middle|\; \begin{array}{l}(\mathsf{st}_1, a) \leftarrow \mathcal{A}_1(\mathsf{pp}, x) \\ \quad c \leftarrow\!\!\$\; \mathcal{C}(\lambda) \\ \quad z \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{st}_2, a, c)\end{array}\right] = \eta(\mathsf{view}).$$

We call a view "bad" if the adversary is sufficiently successful, but extraction takes longer than one would expect. Intuitively, this can only happen if it breaks space-hardness of $\mathsf{simPoTS}$ with higher probability than it should. This is essentially what we will prove in the remainder of this section.

Definition 12 (Bad view)**.** Let view be a view of $\mathcal{A}$. For any extractor Ext, polynomial $\mathrm{poly}(\cdot)$, and knowledge error $\kappa(\cdot)$, we call a view "bad" if $\eta(\mathsf{view}) > \kappa(\lambda)$, but the expected number of steps (one oracle call counts as one step) of $\mathsf{Ext}^{\mathcal{A}}$ is greater than $\frac{\mathrm{poly}(\lambda)}{\eta(\mathsf{view})-\kappa(\lambda)}$.

With these definitions in place, we call a protocol "knowledge sound" if finding bad views is computationally hard.

Definition 13 (Space-hard knowledge soundness)**.** ArKoS has knowledge soundness with *space gap* $0 \leq \varepsilon < 1$ and *knowledge error* $\kappa(\cdot)$ if the following holds: There exists a polynomial $\mathrm{poly}(\cdot)$ and extractor Ext such that for a sufficiently large polynomial $S(\cdot)$, statement x, and $\mathcal{A}$ with space bound $\mathrm{Space}(\mathcal{A}_1, \mathcal{A}_2) \leq (1-\varepsilon) \cdot S(\lambda)$ it holds that

$$\Pr[(x, \mathsf{pp}, \mathsf{st}_0) = \mathsf{view} \text{ is bad} \mid \mathsf{pp} \leftarrow \mathsf{Setup}(S(\lambda)); \mathsf{st}_0 \leftarrow \mathcal{A}_0(\mathsf{st}, x)] \leq \mathrm{negl}(\lambda).$$

We now show that if $\mathsf{simPoTS}$ is strongly space-hard, then finding bad views is hard. Intuitively, we run a truncated version of the extractor that tries to find two transcripts with different challenges. During execution, we continually embed strong-space hardness challenges. If Σ's special soundness does not allow us to extract, then we can instead break one strong space-hardness challenge.

Let us point out two parameters of the theorem statement. First, the reduction needs to store roughly $3\times$ as much space as the adversary, so the space gap degrades accordingly. We will discuss in Rem. 7 how we can reduce the space overhead at the cost of a longer runtime. Second, the knowledge error κ needs to be large enough relative to the number of attempts in the strong space-hardness game, namely $\kappa = \Omega(1/\ell)$.

Remark 6. Note that this also implies sigma protocols Σ's well-known knowledge error bound of $\kappa \geq 1/|C|$. Indeed, recall that Σ and $\mathsf{simPoTS}$ share the challenge space $\mathcal{C}$ and suppose that $\ell \geq |\mathcal{C}|$ for contradiction. Then there exists an adversary that breaks $\mathsf{simPoTS}$ strong-space hardness with constant probability (the adversary simply guesses and solves one challenge during precomputation and then hopes to get that challenge in one of its ℓ online attempts).

Theorem 4. *If* simPoTS *is strongly space-hard against adversaries parameterized by* t_{pre}, t_{on}, p, *and* ℓ *with a space-gap* ε', *then* Σ-OR *has knowledge soundness with space gap* $\varepsilon \geq (\varepsilon' + 2)/3$ *and knowledge error* $\kappa(\lambda) \geq 480/\ell$ *against an adversary also parameterized by* t_{pre}, t_{on}, *and* p.

Proof. This proof mostly follows Damgård and Fujisaki [DF02, § 5], but adapted to our setting.

Consider the Σ-OR adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$. We will first describe Ext. Then we show that if $\mathcal{A}_0$ outputs bad views too often, we can create an adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1, \ldots)$ breaking strong space-hardness of simPoTS.

Description of Ext**:** Our goal is to find two Σ-OR transcripts of the form $(a_1, a_2, c_1, c_2, z_1, z_2)$ and $(a_1, a_2, c_1', c_2', z_1', z_2')$ that verify and where $c_1 \neq c_1'$. Given such transcripts, we can apply Σ's special soundness to extract a witness w.

Towards this goal, we take the procedure Rewind(view) from [DF02, App. A] with minor modifications (we will explain it momentarily). Assume for now that Rewind needs $120/\eta(\mathsf{view})$ steps in expectation. It produces two such transcripts that verify, but does not guarantee $c_1 \neq c_1'$. We call an output of Rewind "useful" if it also holds that $c_1 \neq c_1'$ and "useless" if not. Useful outputs are useful for extracting a witness. Useless outputs are actually also useful—but to break strong space-hardness.

Ext now calls Rewind(view) repeatedly until it gets a useful output and uses this output to extract a witness w. Since Rewind(view) takes expected steps $120/\eta(\mathsf{view})$, Ext runs in expected steps $\mathrm{poly}(\lambda)/\eta(\mathsf{view})$ unless

$$\Pr[\mathsf{Rewind}(\mathsf{view}) \text{ output is useless}] \geq 1/2$$

(otherwise, we expect to run Rewind at most twice). Thus, if view is bad, Rewind(view) outputs a useless value with probability at least $1/2$.

Adversary $\mathcal{B}$ **against** simPoTS**:** We now design an adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1, \ldots)$ against the strong space-hardness of simPoTS. For preprocessing, we set $\mathcal{B}_0 = \mathcal{A}_0$. For the online algorithms, $\mathcal{B}_{2i-1}$ and $\mathcal{B}_{2i}$ will run $\mathcal{A}_1$ and $\mathcal{A}_2$ as called by Rewind. It follows that $\mathcal{A}$ and $\mathcal{B}$ have the same parameters t_{pre}, t_{on}, and p (ignoring minor time overheads orchestrating the Rewind calls).

Next, we show that $\mathcal{B}$ performs at most ℓ online interactions for sufficiently large $\eta(\mathsf{view})$. As said, Rewind runs in an expected number of steps $120/\eta(\mathsf{view})$ where a step is an oracle call to $(\mathcal{A}_1, \mathcal{A}_2)$. Running Rewind for a strict number of steps $(4 \cdot 120)/\eta(\mathsf{view})$, Markov's inequality implies that Rewind terminates and produces an output with probability $\leq 1/4$. As long as $\eta(\mathsf{view}) \geq 480/\ell$, which holds by assumption on κ, $\mathcal{B}$ performs at most ℓ online interactions in the strong space-hardness game.

Last, we consider $\mathcal{B}$'s success probability. If view is bad, this output will be useless except for probability $< 1/2$. Now assume that a useless Rewind output will allow us to break strong space-hardness. This means that $\mathcal{B}$ breaks strong space-hardness with $\Pr[\mathsf{view} \text{ is bad}] \cdot (1 - 1/4 - 1/2)$. By the space usage of Rewind stated below, it breaks strong space-hardness with space gap $\varepsilon' \leq 3\varepsilon - 2$, which is non-negative for $\varepsilon > 2/3$. Thus, $\Pr[\mathsf{view} \text{ is bad}]$ must be negligible.

The procedure Rewind**:** First consider an idealized game on a binary matrix $H \in \{0,1\}^{R \times C}$ which has an η fraction of 1 entries, the coordinates of which are unknown to us. Our goal is to find two 1 entries in the same row. The procedure first randomly samples entries in H until it finds a 1. Then, it runs two subroutines in an alternating fashion (note that [DF02, App. A] runs them in parallel):

1. Randomly sample an entry in the same row where the previous 1 entry was. If we find a 1 entry again, terminate and output the pair. Note that compared to [DF02, App. A] we allow the 1 entry to be the same one.

2. Flip a biased coin that comes up 1 with probability $\eta/7$. Since η is unknown, this is done by flipping a coin that comes up 1 with probability $1/7$ and sampling a random entry of H. If both are 1, abort and restart from the beginning.

Next, we argue that Rewind runs in expected time $120/\eta$. Finding the first 1 takes $1/\eta$ in expectation, and each subroutine runs for at most $7/\eta$ steps in expectation before Subroutine 2 aborts. This results in $15/\eta$ steps in expectation before an abort happens, and we will now show that it aborts with probability of $1/8$, leading to $120/\eta$ steps in expectation. This closely follows [DF02, App. A]'s argument, but we account for the fact that we allow the same 1 entry in Subroutine 1. For Subroutine 1 note that a row will have at least $\eta/2$ 1s with probability at least $1/2$ by an averaging argument. In such a row, Subroutine 1 produces an output in at most $4/\eta$ steps except for probability of at most $1/2$ by Markov's inequality. Subroutine 2 takes $7/\eta$ steps in expectation, and more than $4/\eta$ steps with probability at least $1/2$ by [DF02, App. A]. Thus, the overall probability that Subroutine 1 produces an output before Subroutine 2 aborts is at least $1/8$.

In our setting the rows of H represent the first message of $\mathcal{A}_1$ and the columns the challenges of the verifier. An entry is 1 if and only if $\mathcal{A}$ produces a transcript that verifies. Thus, $C = |\mathcal{C}(\lambda)|$ and $\eta = \eta(\mathsf{view})$. Sampling an entry is in general implemented by running $\mathcal{A}$, but the details depend on which subroutine is being run. When trying to find the first 1, we run $\mathcal{A}$ repeatedly with a uniform challenge from $\mathcal{C}$ (this takes one $\mathcal{B}$ interaction in the strong space-hardness game per attempt). Once it finds the first 1, we keep its intermediate state st_1 and transcript $(a_1, a_2, c_1, c_2, z_1, z_2)$ in memory. Then, we sample an entry in the aforementioned subroutines in the following way:

1. We request a new challenge c^* from the strong space-hardness game of simPoTS. We then rewind $\mathcal{A}$ by running $\mathcal{A}_2(\mathsf{st}_1, c^* + c_1)$ (again, one $\mathcal{B}$ interaction).

2. We use st_0 and run $\mathcal{A}_1$ and $\mathcal{A}_2$ without rewinding on a uniform challenge from $\mathcal{C}$ (again, one $\mathcal{B}$ interaction).

This ensures that Rewind outputs two transcripts with the same first message and different challenge. If the output is useless, then the programming of the challenge ensures that $c_2 = c^*$ in Subroutine 1, so it can be used to break strong space-hardness.

In terms of space overhead, Rewind must store st_0 constantly, and must keep track of st_1 (we ignore other smaller values needed for bookkeeping). Since each state may at most have size $(1-\varepsilon) \cdot S$ by definition, Rewind takes at most $3 \cdot (1-\varepsilon) \cdot S$ space. □

Remark 7. Like in the proof of Lem. 11 we can use a random oracle (or PRFs [ACFK17]) to reduce the space overhead. Instead of storing st_0 and st_1, $\mathcal{B}$ always recomputes them

by re-running $\mathcal{A}$. To make $\mathcal{A}$ deterministic, $\mathcal{B}$ provides $\mathcal{A}$ with fixed randomness derived using RO, which allows for a concise description. The downside of this approach is that it increases the runtime of $\mathcal{B}$ (in particular, the online time of $\mathcal{B}$ is larger than the precomputation time of $\mathcal{A}$).

8.3 Knowledge soundness of other resources

The above proof should extend to the resources work and sequential time. One just needs to define a "strong work/time-hardness" capturing ℓ interactions (analogous to Def. 10). In this case, work/sequential time are measured per interaction.

Note that in Thm. 4 the knowledge error $\kappa = \Omega(1/\ell)$, so ideally ℓ is large. This is mostly relevant to short-lived proofs [ABC22] constructions relying on hidden-order groups. With sufficient computation, an adversary can figure out the order, so a safe choice is $\ell \in \text{poly}(\lambda)$

Acknowledgements. Jesko Dujmovic: Funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Christoph U. Günther and Krzysztof Pietrzak: This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/F85. For open access purposes, the author has applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

References

[ABC22] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 487–516. Springer, Cham, December 2022.

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 538–557. Springer, Cham, September 2014.

[ACFK17] Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 101–132. Springer, Cham, August 2017.

[ACK+16] Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 358–387. Springer, Berlin, Heidelberg, May 2016.

[ACP+17] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 33–62. Springer, Cham, April / May 2017.

[AGP24] Benedikt Auerbach, Christoph U. Günther, and Krzysztof Pietrzak. Trapdoor memory-hard functions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 315–344. Springer, Cham, May 2024.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 595–603. ACM Press, June 2015.

[BDL19] Mihir Bellare, Wei Dai, and Lucy Li. The local forking lemma and its application to deterministic encryption. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 607–636. Springer, Cham, December 2019.

[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, Berlin, Heidelberg, August 1993.

[BKZZ16] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 902–933. Springer, Berlin, Heidelberg, December 2016.

[BRZ18] Jeremiah Blocki, Ling Ren, and Samson Zhou. Bandwidth-hard functions: Reductions and lower bounds. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1820–1836. ACM Press, October 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, Berlin, Heidelberg, August 1994.

[CS20] Sofía Celi and Iraklis Symeonidis. The current state of denial. In *Privacy Enhancing Technologies Symposium*, 2020.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024.

[DDJ24] Nico Döttling, Jesko Dujmovic, and Antoine Joux. Space-lock puzzles and verifiable space-hard functions from root-finding in sparse polynomials. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 431–459. Springer, Cham, December 2024.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 125–142. Springer, Berlin, Heidelberg, December 2002.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Berlin, Heidelberg, August 2015.

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 473–495. Springer, Cham, April / May 2017.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54. Springer, Berlin, Heidelberg, August 2005.

[Fis19] Ben Fisch. Tight proofs of space and replication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 324–348. Springer, Cham, May 2019.

[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, Berlin, Heidelberg, August 1987.

[GGHAK22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 458–487. Springer, Cham, May / June 2022.

[LT82] Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *J. ACM*, 29(4):1087–1130, October 1982.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009.

[Pie19] Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 59:1–59:25. LIPIcs, January 2019.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 262–285. Springer, Berlin, Heidelberg, October / November 2016.

[RD17] Ling Ren and Srinivas Devadas. Bandwidth hard functions for ASIC resistance. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 466–492. Springer, Cham, November 2017.

[Rey24] Leonid Reyzin. Proofs of space with maximal hardness. In *65th FOCS*, pages 1159–1177. IEEE Computer Society Press, October 2024.

[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, New York, August 1990.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3-4):207–388, 2010.