An abridged version of this paper appears in the proceedings of the 39^{th} International Cryptology Conference—CRYPTO 2019. This is the full version.

Continuous Space-Bounded Non-Malleable Codes from Stronger Proofs-of-Space

Binyi Chen*[1], Yilei Chen[2], Kristina Hostáková†[3], and Pratyay Mukherjee[4]

[1] *University of California, Santa Barbara,* binyichen@cs.ucsb.edu
[2] *Visa Research,* yilchen@visa.com
[3] *TU Darmstadt,* kristina.hostakova@cs.tu-darmstadt.de
[4] *Visa Research,* pratmukh@visa.com

Abstract

Non-malleable codes are encoding schemes that provide protections against various classes of tampering attacks. Recently Faust et al. (CRYPTO 2017) initiated the study of *space-bounded* non-malleable codes that provide such protections against tampering within small-space devices. They put forward a construction based on any *non-interactive proof-of-space (NIPoS)*. However, the scheme only protects against an a priori bounded number of tampering attacks.

We construct non-malleable codes that are resilient to an unbounded polynomial number of space-bounded tamperings. Towards that we introduce a stronger variant of NIPoS called *proof-extractable* NIPoS (PExt-NIPoS), and propose two approaches of constructing such a primitive. Using a new proof strategy we show that the generic encoding scheme of Faust et al. achieves unbounded tamper-resilience when instantiated with a PExt-NIPoS. We show two methods to construct PExt-NIPoS:

1. The first method uses a special family of "memory-hard" graphs, called *challenge-hard graphs* (CHG), a notion we introduce here. We instantiate such family of graphs based on an *extension of* stack of localized expanders (first used by Ren and Devadas in the context of proof-of-space). In addition, we show that the graph construction used as a building block for the proof-of-space by Dziembowski et al. (CRYPTO 2015) satisfies challenge-hardness as well. These two CHG-instantiations lead to continuous space-bounded NMC with different features in the random oracle model.
2. Our second instantiation relies on a new measurable property, called *uniqueness* of NIPoS. We show that standard extractability can be upgraded to proof-extractability if the NIPoS also has uniqueness. We propose a simple heuristic construction of NIPoS, that achieves (partial) uniqueness, based on a candidate memory-hard function in the standard model and a publicly verifiable computation with small-space verification. Instantiating the encoding scheme of Faust et al. with this NIPoS, we obtain a continuous space-bounded NMC that supports the "most practical" parameters, complementing the provably secure but "relatively impractical" CHG-based constructions. Additionally, we revisit the construction of Faust et al. and observe that due to the lack of uniqueness of their NIPoS, the resulting encoding schemes yield "highly impractical" parameters in the continuous setting.

We conclude the paper with a comparative study of all our non-malleable code constructions with an estimation of concrete parameters.

*Research conducted at Visa Research.
†Research conducted at Visa Research.

Contents

1 Introduction

Non-malleable codes and tamper-resilience. The notion of non-malleable codes (NMC) was put forward by Dziembowski, Pietrzak and Wichs [DPW10] as an abstract tool for protecting cryptographic devices against tampering attacks (e.g. [BDL01]). Intuitively, an encoding scheme $(\mathsf{Encode}, \mathsf{Decode})$ is called non-malleable with respect to a class of tampering adversaries (modeled as functions or algorithms) $\mathcal{A}$ if for any adversary $\mathsf{A} \in \mathcal{A}$ and any message x, the output $\mathsf{Decode} \circ \mathsf{A} \circ \mathsf{Encode}(x)$ is independent of x, unless it is equal to x. It is straightforward to see that $\mathcal{A}$ can not contain all efficiently computable functions because in that case it is always possible to just decode a codeword c to x, modify (for example add 1) and re-encode $x + 1$; hence one must consider a restricted class $\mathcal{A}$ which excludes functions able to encode or decode. Therefore, the NMC literature (for example [LL12, FMNV14, AAG+16, DLSZ15, FMNV15, JW15, ADL14, CG14]) focuses on constructing encoding schemes that are non-malleable against a meaningful, broad class of tampering functions; notice that non-malleability against a broader $\mathcal{A}$ translates to protection against stronger tampering attacks.

Leaky NMC for space-bounded tampering. One such interesting tampering class is space-bounded tampering, in that the only restriction on $\mathcal{A}$ is that any (efficient) tampering algorithm in this class can only use a limited amount of memory. Space-bounded tampering captures the essence of mauling attacks performed by malware that infect small-space devices like mobile phones. However, as noticed by Faust et al. [FHMV17a] (henceforth FHMV), for such tampering class it is unreasonable to assume that a tampering algorithm can not decode. For example, if decoding requires more space than what is available for the attacker, then the encoded secret becomes unusable inside the device. The encoding algorithm, on the other hand, can be reasonably space-intense and performed outside the device. Therefore, it is possible to assume the space-bounded adversary cannot perform encoding, therefore avoiding the aforementioned impossibility.

Moreover, even if $\mathcal{A}$ includes only Decode, "full-fledged" non-malleability is still not achievable. To see this, consider an attacker that decodes c, learns the message x and based on the first bit of x overwrites the memory of the device with a precomputed encoding — leaking the first bit (this can be easily extended to an attack that leaks any $\log(|x|)$ bits by tampering once). However, Faust et al. [FHMV17a] observed that all hope may not be lost if it is possible to guarantee that the leakage is "not too much". Formally FHMV defines a weaker notion called *leaky non-malleability*, which guarantees that an encoding scheme satisfying the notion would leak only a limited amount of information about x. FHMV also showed that this is sufficient for many applications. For example, they showed how one can use such leaky NMC by trading-off tampering with leakage when x comes from a high-entropy distribution (see Section 7 of [FHMV17b] for more details).

Continuous space-bounded tampering. Traditional NMC (as defined in [DPW10]) guarantees non-malleability when the attacker tampers only once. To use such NMC for tamper-resilience (see [DPW10] for more details), one needs to refresh the encoding after each tampering. To combat this issue, in 2014, Faust et al. [FMNV14] proposes the notion of *continuous non-malleable codes* that tolerates an unbounded number of tampering attempts, which consequently removes the necessity of re-encoding in the tampering application. Though FHMV's definition of (leaky) non-malleability allows continuous tampering, their construction (see Theorem 3 of [FHMV17b]) only allows an a priori bounded number of tampering attempts (say θ) because their parameters are related in a way that the leakage (say, ℓ) is directly proportional to θ. Hence, after a few tampering attempts, the leakage becomes as large as $|x|$. Coming up

with a construction that tolerates an unbounded (polynomially large) θ was left open in FHMV (see Remark 2 of [FHMV17b]).

1.1 Our Work

Leaky NMC for continuous space-bounded tampering. In this work we address the open problem by proposing various constructions of non-malleable codes, in all of which the leakage ℓ is proportional to the logarithm of the number of tamperings, i.e. $\log(\theta)$.[1] No prior bound is required for θ in this case. However, we do not claim that our solutions are strictly stronger than that provided in FHMV, because we assume a "self-destruct" mechanism similar to the prior works on continuous non-malleability (e.g. [FMNV14]). Roughly speaking, the "self-destruct" mechanism requires the small-space device to erase its entire state (or make it non-functional) once a tampering is detected. As already shown by FHMV, this is a *necessary* requirement for achieving unbounded continuous space-bounded tampering.

Our approach: Stronger non-interactive proof-of-space. FHMV's encoding scheme relies on any extractable non-interactive proof of space (simply called NIPoS)[2]. In contrast, we introduce a *new* and *stronger* property of NIPoS called *proof-extractability* and prove that when FHMV's encoding scheme is instantiated with a proof-extractable NIPoS (PExt-NIPoS), then we obtain a continuous space-bounded NMC (CSNMC). We take two different approaches to construct PExt-NIPoS — in the following few paragraphs we choose to outline them through the natural flow of our attempts, instead of dividing strictly into two distinct approaches.

Proof-extractability from any NIPoS with uniqueness. Our starting point is the construction of FHMV [FHMV17a] which is based on any NIPoS. We show that any NIPoS can be upgraded to a PExt-NIPoS if it has a special property called *uniqueness*, which we define as a quantitative measure of a NIPoS. We notice that the parameters of the resulting PExt-NIPoS (and consequently the CSNMC scheme yielded via FHMV's generic construction) is directly related to the uniqueness parameter of the starting NIPoS. For example, if a NIPoS has "maximal uniqueness", then the resulting CSNMC incurs "minimal leakage", which is equal to $p - |c|$ bits, where p is the available (persistent) space.[3] Unfortunately, we do not know of a provably secure NIPoS construction with maximal, or even a "reasonably good measure" of uniqueness (later we propose a construction that satisfies partial uniqueness based on heuristic assumptions). In fact, we show that the NIPoS used in FHMV (which is in turn based on the PoS proposed by Ren and Devadas in [RD16]) has poor uniqueness parameters and thus, when adapted to our proof-extractability technique, yields a CSNMC which suffers from a leakage that is as large as $\approx p - |x|$.

Modeling space-bounded adversary with bounded description. The lack of a NIPoS with "good uniqueness" drives us to revisit the adversarial model of FHMV, in particular, how

[1] In the rest of the paper whenever we say that an encoding scheme satisfies continuous space-bounded non-malleability or is a CSNMC, we mean that the encoding scheme is a leaky NMC for space-bounded tampering with $\ell \propto \log(\theta)$.

[2] In this section by NIPoS we informally refer to a (non-interactive) proof-of-space with extractability. Later while treating formally we also say phrases like "extractable NIPoS" or "NIPoS has extractability" explicitly and sometimes also use the abbreviation like Ext-NIPoS for that — but in general a NIPoS is assumed to have implicit extractability.

[3] We assume that the entire space in the device is split into two parts, the persistent space which is reusable, and the transient space which is refreshed after each tampering. An impossibility shown in FHMV (see Theorem 1 of [FHMV17b]) restricts the persistent space to remain relatively small when θ is unbounded.

they formalize the notion of space. In FHMV, which in turn follows the notion introduced by Dziembowski et al. [DKW11b], the adversary is separated into two parts: a "big adversary" which is a PPT adversary with no space-bound, and a "small adversary" that is a space-bounded poly-time adversary. In a security game, the big adversary starts interacting with the challenger, and then outputs small adversaries which will then have access to the target codeword (or the proof, in case of NIPoS) and execute tampering in a space-bounded manner.

We notice that FHMV assumes that the small adversary can have arbitrary amount of auxiliary information hardcoded in its description (see Page-5 of [FHMV17b]). In reality this seems to be an overkill, because if the small adversary (e.g. malware) has a huge description, it might not even fit into a small-space device (e.g. a mobile device), let alone executing tampering. So, it is reasonable to assume that such adversary has a bounded size description. In particular, we define a class of space-bounded adversaries as $\mathcal{A}^{s,f}_{\mathsf{space}}$ containing all poly-time adversaries that have a description of size at most f-bit and which require at most s-bit to execute.

PExt-NIPoS from Challenge-hard Graphs (CHG). We define a new family of "memory-hard graphs" called *challenge-hard-graphs* and construct PExt-NIPoS for the class of space-bounded adversaries $\mathcal{A}^{s,f}_{\mathsf{space}}$ from that. We provide two instantiations of CHG: (i) The first one extends the stack of local expanders (SoLEG), used by Ren and Devadas [RD16] in the context of proof-of-space. We uses a novel technique to connect a gadget with a standard SoLEG in order to amplify crucial challenge-hardness parameters. This technique may be of independent interest. (ii) The second one uses the graph designed by Paul et al. [PTC76] and used by Dziembowski et al. [DFKP15], who use the notion of challenge-hardness *implicitly* to construct proof-of-space. Both of the constructions use standard graph-pebbling techniques to ensure memory-hardness (and challenge-hardness) and work in the random oracle model. Plugging-in these PExt-NIPoS constructions into FHMV's encoding scheme, we obtain CSNMC schemes with "almost minimal leakage" $\ell \approx p - |c|$.

A NIPoS with partial uniqueness based on heuristics. The constructions mentioned above all come with rigorous security proofs (in the random oracle model). However, it turns out that in order to achieve reasonable security, the concrete parameters of these constructions are fairly impractical (see Section 9 for a detail analysis). For example, for a message of size 1 MB, the size of a codeword is almost 800 MB for the CHG-based NMC constructions (see Table 2). To complement this, we take a step back on our initial idea of constructing NIPoS with "good uniqueness", and propose a simple and practical instantiation of NIPoS based on heuristic assumptions. The construction uses a concrete instantiation of a *memory-hard-function* (MHF), and applies a (non-interactive) publicly verifiable computation where the verification requires small space. When the MHF is instantiated with the SoLEG-based construction of Ren and Devadas [RD16], the resulting NIPoS has extractability and a "good measure of uniqueness". This yields a PExt-NIPoS with very good parameters and, consequently, plugging-in that to FHMV's encoding scheme we obtain a CSNMC with very small proof size (in killobytes), that also allows a leakage, as small as $p - 0.99|c|$, in certain settings.

While the above scheme is practical, it is not provably secure, since we can not assume that the hash-functions within the MHF are random oracles, as the prover needs to access the circuit of the MHF to produce a proof of computation.[4] Note that any MHF, while used in practice with concrete hash functions (for example SHA3) for important practical applications [Tar], provides provable guarantees only in the random oracle model (see, e.g. [ACP+17]). Instead, we

[4]This is similar in spirit to the works (e.g., [CGM16b, DFKP16]) that use zero-knowledge proofs of full-domain-hash signatures — security of such signatures only holds in the random oracle model, while one needs to have a concrete instantiation before applying zero-knowledge proofs.

Approach	PExt-NIPoS type	Assumptions	Leakage	Size of A
CHG	SoLEG-based	RO	$\approx p - \|c\|$	Bounded
	PTC-based	RO	$\approx p - \|c\|$	Bounded
Uniqueness	FHMV-based	RO	$\approx p - \|x\|$	Unbounded poly
	MHF-based	Heuristic	$\approx p - 0.99\|c\|$	Unbounded poly

Table 1: Among the above constructions, the MHF-based one is the most practical one whereas the SoLEG-based one has the best concrete parameters among the provably-secure constructions. For a detail comparison of the concrete parameters please see Table 2 in Section 9.

rely on heuristic assumptions that intuitively state that the MHF remains memory-hard when the random oracle is instantiated with a standard hash function like SHA3.

Roadmap. We summarize our contributions below in Section 1.2. In Section 1.3 we provide an elaborative technical overview. Then, after providing preliminaries in Section 3 and basic definitions of Continuous Space-bounded Tampering in Section 4, we define the new NIPoS properties (uniqueness and proof-extractability) in Section 5 where we also discuss their relations. In Section 6, we show that the FHMV's encoding scheme satisfies continuous space-bounded non-malleability when instantiated with PExt-NIPoS. Section 7 introduces the notion of challenge-hard-graphs and shows how to use them to construct PExt-NIPoS. We provide a heuristic construction of NIPoS with (partial) uniqueness relying on memory-hard functions in Section 8 and finally in Section 9, we conclude with a instantiations and comparison of the important concrete parameters of different encoding schemes we constructed.

1.2 Summary of our Contributions

Our overall contributions can be summarized as follows:

- We propose the first constructions of continuous space-bounded (leaky) non-malleable codes (with a necessary "self-destruct" mechanism) and thus resolve an open problem posed by FHMV [FHMV17a]. Overall we propose *four* different constructions of different merits; we provide a comparison in Table 1:
- We introduce various abstract notions of NIPoS, like *proof-extractability* and *uniqueness*, and show relations among them. The abstractions are targeted towards constructing CSNMC as the main end goal, but may be of independent interests. We prove that the FHMV encoding scheme is a CSNMC when instantiated with any PExt-NIPoS.
- We propose different techniques to construct a PExt-NIPoS. We introduce the notion of *challenge-hard graphs* and show how to build PExt-NIPoS from that. We propose a novel technique to bootstrap the important *challenge-hardness parameters* of a CHG by carefully connecting a gadget to a special type of memory-hard graphs (SoLEG). Furthermore, we provide a simple construction of *partially unique* NIPoS that yields "reasonably practical" parameters for the resulting PExt-NIPoS and CSNMC. It is based on heuristic assumptions on memory-hard functions and complements the provably secure but "relatively impractical" CHG-based constructions.
- Finally we provide a comparative study of the most important parameters of all our CSNMC constructions with respect to concrete instantiations. This helps us to understand the practical impacts of different techniques and constructions proposed in this work.

1.3 Technical Overview

Revisiting FHMV's construction. We start by briefly revisiting the construction of FHMV [FHMV17a]. Recall that FHMV's generic encoding scheme is based on any extractable (non-interactive) proof-of-space (NIPoS).

First let us briefly recall the notion of proof-of-space introduced in [ABFG14, DFKP15]. In an interactive proof-of-space (PoS) protocol, a prover P interactively proves that she has "sufficient amount of space/memory" to a space-bounded verifier V. One can use Fiat-Shamir transformation [FS87] to make it non-interactive, in that the entire proof can be represented as one single string, say π_{id}, with respect to an identity id. The verifier is able to verify the pair (id, π_{id}) within bounded space. Extractability of NIPoS guarantees that: given an honestly generated pair (id, π_{id}), if a space-bounded "small adversary" A is able to compute another valid (i.e. correctly verified) pair $(id', \pi_{id'})$ such that $id \neq id'$, then id' can be *efficiently* extracted from the RO queries made by the "big adversary" B (that has no space-restriction and may be represented by a PPT algorithm) given a "small hint"[5].

Given a NIPoS, FHMV's encoding scheme works as follows. On input a message x, the space-intense encoding algorithm runs the prover of NIPoS on an identity x to generate a proof π_x. The codeword c is simply the pair (x, π_x). The space-bounded decoding algorithm, on receiving $c = (x, \pi_x)$, runs the (space-bounded) verifier. If the verification passes, it returns x, otherwise it returns $\perp$ denoting the invalidity of c. Intuitively, non-malleability follows from the guarantee provided by NIPoS; namely, whenever the small adversary tampers to a valid codeword $(x', \pi_{x'})$, the new message x' must be independent of the input message x.

To be slightly more formal, to show that this encoding scheme is non-malleable against space-bounded attacker, one needs to simulate the tampering experiment with "a small leakage" on x. Given the extractability, the simulator can be constructed as follows: the leakage is obtained using the "small hint". As guaranteed by the extractability of NIPoS, since the "small hint" (of length η, say) is sufficient to extract id', each tampering can be simulated by first obtaining the hint as a leakage and then running the NIPoS-extractor to obtain id'. Clearly, this strategy runs into problem for unbounded continuous tampering as the overall leakage ℓ becomes proportional to $\theta \cdot \eta$ (where θ denotes the number of tampering queries).

Proof-extractability to the recovery. The above discussion shows that we possibly need a stronger guarantee from the underlying NIPoS to make FHMV's encoding scheme a CSNMC. Towards that, we introduce a stronger property of a NIPoS called *proof-extractability* (PExt-NIPoS). It guarantees that, given a "small hint" (of length η', say), it is possible to construct a stronger extractor that extracts not only the changed identity, but also the changed proof: $(id', \pi_{id'})$. Intuitively, this means that if a small adversary computes a valid pair $(id', \pi_{id'})$, then the "big adversary" must have computed the *entire* proof $\pi_{id'}$ (as opposed to a part of the proof as for NIPoS) outside the small-space device; hence, enabling extracting the entire proof from the RO queries made by B only.

Given the proof-extractor, the new NMC simulator works as follows: it uses the hint to get a "small leakage" and then runs the proof-extractor to obtain $(id', \pi_{id'})$. Furthermore, the simulator also needs an extra leakage, which consists of the "extra persistent space" (of size $p - n$)[6] — now the simulator reconstructs the entire persistent tampered state and can continue the rest of

[5] Note that we made some syntactical change to FHMV's definition of extractability by introducing an explicit hint-producing function. We introduce the length of the hint as a new extractability parameter which must be small for making the definition meaningful. For example, if the leakage function leaks the entire pair (id', π'_{id}), then the definition would be trivially satisfied. Looking ahead, in the proof of CSNMC this hint will be used by the NMC simulator as a leakage to simulate the tampering experiment. For more details we refer to Section 5.

[6] As discussed in Remark 3, this leakage is necessary in most of the natural settings.

the tampering experiment without having to make any further leakage query. However, to avoid any leakage before the first tampering takes place (for example, if the first 100 tampering functions are identities), the simulator needs to know the index when the target codeword changes for the first time in the sequence of tampering and for that the leakage becomes proportional to $\log(\theta)$. Overall, the simulator only needs to make a constant number of leakage queries (two, to be precise) to simulate any (polynomial) number of tampering, as opposed to making one leakage query for each tampering. The overall leakage becomes $\ell \propto \log(\theta) + \eta' + (p - n)$ thereby achieving CSNMC. Therefore, the main question that remains is how to construct PExt-NIPoS, which will be described in the next few paragraphs.

Uniqueness and Proof-extractability. We observe that, if a NIPoS has a special property, called *uniqueness*, then it satisfies *proof-extractability*. Intuitively, uniqueness means for a fixed identity id, there exists exactly one string π_{id} such that π_{id} verifies correctly with respect to id. Unfortunately, we do not know how to construct a NIPoS with such property (even under heuristic assumptions). Therefore, to have a more relaxed and fine-grained notion, we define uniqueness as a *quantitative measure*: a NIPoS has u_{pos}-uniqueness means that, for any identity id, the first u_{pos} bits of any valid π_{id} are fixed and can be computed efficiently with overwhelming probability.

We then show (in Lemma 2) that any u_{pos}-unique NIPoS satisfies proof-extractability, where the size η' of the hint required for PExt-NIPoS depends on u_{pos} as: $\eta' = \eta + n_{\mathsf{pos}} - u_{\mathsf{pos}}$, where η denotes the size of the hint of the starting NIPoS and n_{pos} denotes the size of the proof. This follows naturally from the construction of the hint-producing function of PExt-NIPoS, as the hint for the proof extractor needs to contain enough information to extract both id' and $\pi_{id'}$. Now id' can be extracted from the hint produced via the starting NIPoS (by standard extractability); given id' the proof-extractor can compute the first u_{pos} bits of $\pi_{id'}$; but the remaining part, which has length $n_{\mathsf{pos}} - u_{\mathsf{pos}}$, must be separately output by the hint-producing function of PExt-NIPoS. Notice that, maximal uniqueness means $u_{\mathsf{pos}} = n_{\mathsf{pos}}$ which in turn implies $\eta' = \eta$. Hence, if FHMV's encoding scheme is instantiated with a maximally unique NIPoS, part of the leakage of the resulting CSNMC would be determined by only η and hence would be minimal. We leave the task of constructing a maximally unique NIPoS as an interesting open problem. On the other hand, we observe that the NIPoS considered by FHMV has $u_{\mathsf{pos}} \approx 0$ and hence the leakage is largely dominated by $\eta + n_{\mathsf{pos}}$, resulting in much worse parameters.

Partially unique-NIPoS from memory-hard functions. We are able to construct an NIPoS with reasonably large u_{pos} from heuristic assumptions on memory-hard functions. The construction is very simple: let M be a concrete instantiation of a memory-hard function, which guarantees that any space-bounded adversary can not compute the function on a randomly chosen input in polynomial time. Let us assume a verifiable computation scheme (VC) where the verification can be done in small-space. Then the NIPoS prover works as follows: given an identity id, first compute a hash (that is assumed to be a random oracle) to generate a random value $x := \mathcal{H}(id)$, then compute $y := M(x)$ and finally run the VC prover to produce a proof π_{vc} to prove that y is indeed obtained by computing $M(x)$. The proof-of-space is then defined to be the pair $(M(x), \pi_{\mathsf{vc}})$. The NIPoS verifier works naturally by first computing $x = \mathcal{H}(id)$ and then verifying the proof π_{vc} in small-space.

To see that the construction above yields a NIPoS with good uniqueness, first note that the extractability follows from the fact that the function M is memory-hard and can not be computed on a random input by a space-bounded "small adversary"; hence, the "big adversary" must have queried on id' beforehand enabling extraction of id' from B's RO queries. Note that here we also need to rely on the soundness of VC as otherwise the small adversary could just compute

a different "memory-easy" function and "fake" the proof of computation to fool the verifier. Moreover, note that, the first part of the NIPoS proof is indeed uniquely determined (with overwhelming probability any other string would fail to verify as guaranteed by the soundness of the VC scheme), whereas the second part, i.e. the proof π_{vc}, is not. So, overall we have a NIPoS with $u_{\mathsf{pos}} = |y|$. Since the VC produces a short proof to enable small-space verification (we use Pinocchio [PHGR13] to instantiate), we are able to have a NIPoS with fairly large u_{pos}, which in turn leads to a CSNMC with very good parameters.

PExt-NIPoS from Challenge-hard graphs (CHG). In addition to the heuristic construction above, we also construct a provably secure PExt-NIPoS in the random oracle model, albeit with an additional restriction on the class of space-bounded adversaries, namely assuming that the description size of a small-space adversary is also bounded (as discussed in Section 1.1).

To do so, we define a new notion of memory-hard graphs, called challenge-hard graphs (CHG). Recall that, special types of DAGs are used for memory-hardness and for constructing proof-of-space via graph-labeling games. Usually, labels are the output of the hash functions modeled as random oracles (therefore are not "compressible"). In a graph-based proof of space constructions (e.g. [RD16]), an honest prover computes the labeling of the entire graph ensuring the usage of significant amount of space. Small-space verification is done by checking the labels of a few randomly selected nodes (or challenge nodes) of the graph — this guarantees that the "small adversary" cannot put too many fake labelings (a.k.a. faults) without storing them and thereby ending up using less memory.

However, such verification leaves room for computing a small part of the proof inside the small-space device — for example, consider a multi-layered DAG (e.g. a stack of bipartite graphs), for which a "big adversary" computes the labeling of the entire graph except for *a single* node in the last layer, and the "small adversary" easily computes the label of the node inside the small-space device. As a result the entire proof can not be extracted only from B's RO queries, making proof-extractability impossible.

To remedy this issue, we replace the traditional memory hard graphs with CHG, which contains another carefully chosen set of challenges and guarantees that, even if a "big adversary" computes the labeling of the entire graph except for a few nodes and send a bounded hint to the "small adversary", it is still infeasible to compute the labels of the new challenge nodes with a small-space device. Let us remark that such a guarantee is only possible when the small adversary has a small description size (i.e., the hint from the "big adversary" is small), as otherwise the small adversary, for example, can hard-code the entire labeling for whole graph including all possible challenges, making challenge hardness impossible. As discussed in Section 1.1, we propose two instantiations of CHGs with different merits with respect to their parameters.

2 Related Works

Our work can be categorized among the work on non-malleable codes against global tampering, where the entire codeword is subject to tampering, as opposed to granular tampering, where the codeword is split into independently tamperable parts. In the NMC literature, majority of work, e.g. [LL12, GPR16, ADL14, AAG+16, CG14, KOS18, KOS17, CGM+16a] falls into the the later category; among them [CGM+16a] considers, a weaker notion (non-malleability with replacement) of NMC like us (leaky-NMC). A few other works, e.g. [FMVW14, AGM+15, BDKM16, BDKM18] consider global tampering. Moreover, most of these work consider one-time tampering. Continuous tampering, first proposed in [FMNV14], is addressed also in [FNSV18, ADN+17, OPVV18, AKO17, FMNV15]. Except FHMV [FHMV17a], the recent work by Ball et al. [BDKM18] also considers space-bounded NMC, albeit in a streaming model. Our modeling of

space-bounded adversary, which is also adapted in FHMV is used in earlier woks like [DKW11b, DKW11a] for constructing different schemes. For more detail on different NMC-based compilers for tamper-resilience we refer to [Muk15].

3 Preliminaries

3.1 Notation

For a string x, we denote its length by $|x|$; a truncated string from i-th bit to j-th bit is denoted by $x[i \dots j]$; for a $a \in \mathbb{N}$, $\mathsf{bit}(a) \in \{0,1\}^*$ is its boolean representation and bit^{-1} is the corresponding inverse function; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the number of elements in $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow\!\!\$\ \mathcal{X}$. When A is an algorithm, we write $y \leftarrow \mathsf{A}(x)$ to denote a run of A on input x and output y; if A is probabilistic, then y is a random variable and $\mathsf{A}(x;r)$ denotes a run of A on input x and randomness r. An algorithm A is *probabilistic polynomial-time* (PPT) if A is probabilistic and for any input x and a randomly chosen $r \in \{0,1\}^*$ the computation of $\mathsf{A}(x;r)$ terminates in at most a polynomial (in the input size) number of steps. We often consider *oracle-aided* algorithms $\mathsf{A}^{\mathcal{O}(\cdot)}$, with access to an oracle $\mathcal{O}(\cdot)$.

For any string x, and any hash function $\mathcal{H}$, we use the notation $\mathcal{H}_x$ to denote the specialized hash function that accepts only inputs with prefix equal to x. Often the hash function is modeled as a random oracle.

We consider Turing Machine as our model of computation where any algorithm A is formally represented as a binary string. Any string w hardwired into A is denoted in the subscript as A_w and also becomes part of its description. An algorithm A has a state $\mathsf{st}_\mathsf{A} \in \{0,1\}^*$ that does not include the description of A. st_A is typically initialized with the input x and (optionally) some other auxiliary information. At each time step st_A is updated. At termination A returns an output y also denoted as $\mathsf{A}(x)$. If A is a stateful algorithm then it also outputs the state st_A.

We denote with $\lambda \in \mathbb{N}$ the security parameter. In the rest of the paper λ will always be an implicit security parameter and any other parameter will be a function of λ. A function $\nu : \mathbb{N} \to [0,1]$ is negligible in the security parameter (or simply negligible), denoted $\nu(\lambda) \in \mathrm{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ, i.e. $\nu(\lambda) = \lambda^{-\omega(1)}$. A function $\mu : \mathbb{N} \to \mathbb{R}$ is a polynomial in the security parameter, written $\mu(\lambda) \in \mathrm{poly}(\lambda)$, if, for some constant $c \geq 1$, we have $\mu(\lambda) \in O(\lambda^c)$.

3.2 Basic Definitions

We present a concrete definition of pseudorandom function.

Definition 1 (Pseudorandom Function)**.** For parameters $k, \mu, n \in \mathbb{N}$ and $\varepsilon_{\mathsf{pr}} \in [0,1)$ a keyed family of functions $\{P_\chi : \{0,1\}^k \to \{0,1\}^n\}_{\chi \in \{0,1\}^\mu}$ is called a $(k, n, \mu, \varepsilon_{\mathsf{pr}})$-pseudorandom function[7] if for any PPT adversary A the probability of the following real-or-random game to output 1 is at most $\frac{1}{2} + \varepsilon_{\mathsf{pr}}$:

Real-or-Random Game for P

- Choose a random key $\chi \leftarrow\!\!\$\ \{0,1\}^\mu$.
- Choose a random bit $b \leftarrow\!\!\$\ \{0,1\}$.
- For each query x from A do as follows depending on b:

[7] Note that $\varepsilon_{\mathsf{pr}}$ would be a function of the security parameter and the number of queries made by A in the real-or-random game. For simplicity we will fix a upper bound on the number of queries and then choose $\varepsilon_{\mathsf{pr}}$ accordingly. Concretely we will fix all query-bound to be 2^{64}. For more details see Section 9.

- $b = 0$: if x queried earlier, send the same reply as before; otherwise send a fresh random value $y \leftarrow\!\!\$\ \{0,1\}^n$ to A.
- $b = 1$: send $y := P_\chi(x)$ to A.

- Receive a guess b' from A. Return 1 if and only if $b = b'$ and 0 otherwise.

Sometimes we will use PRF with arbitrary large domain (that is $\{0,1\}^*$). Such PRF can be constructed from any PRF satisfying above definition by Merkle-Damgård domain extension technique and called a $(*, n, \mu, \varepsilon_{\mathsf{pr}})$-PRF.

We will be using the following tail bound in the proof of Lemma 7.

Lemma 1 (Hoeffding inequality)**.** *Let $Z_1, \ldots, Z_n$ be independent random variables that are identically distributed, and $0 \le Z_i \le 1$ for every $i \in [n]$. Denote by $X_n := \sum_{i=1}^n Z_i$. Then, for any $\varepsilon \in (0,1)$, it holds that*

$$\Pr\left[X_n \le \mathbb{E}\left[X_n\right] - \varepsilon n\right] \le \exp(-2\varepsilon^2 n)\,.$$

Let us recall the standard definitions of coding schemes for boolean messages.

Definition 2 (Coding schemes/Codes)**.** A (k,n)-code $\Pi = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ is a triple of algorithms specified as follows: (i) The (randomized) generation algorithm Init takes as input $\lambda \in \mathbb{N}$ and returns public parameters $\mathsf{pp} \in \{0,1\}^*$; (ii) The (randomized) encoding algorithm Encode takes as input hard-wired public parameters $\mathsf{pp} \in \{0,1\}^*$ and a value $x \in \{0,1\}^k$, and returns a codeword $c \in \{0,1\}^n$; (iii) The (deterministic) decoding algorithm Decode takes as input hard-wired public parameters $\mathsf{pp} \in \{0,1\}^*$ and a codeword $c \in \{0,1\}^n$, and outputs a value in $\{0,1\}^k \cup \{\perp\}$, where $\perp$ denotes an invalid codeword.

We say that Π satisfies correctness if for all $\mathsf{pp} \in \{0,1\}^*$ output by $\mathsf{Init}(1^\lambda)$ and for all $x \in \{0,1\}^k$, $\mathsf{Decode}_{\mathsf{pp}}(\mathsf{Encode}_{\mathsf{pp}}(x)) = x$ with overwhelming probability over the randomness of the encoding algorithm. A codeword c is called *valid* if $\mathsf{Decode}_{\mathsf{pp}}(c) \neq \perp$.

3.3 Bounded Algorithms

In this paper we will be dealing with algorithms that are restricted in terms of different resources. In particular we consider two main types of resource: time and space. Importantly, in contrast with [FHMV17a] we split the space-usage into two parts: (i) the space required to store the algorithm and (ii) additional space used by it. Faust et al. [FHMV17a] only assumes concrete measure of the later one and the former one was implicitly assumed to be an unbounded polynomial in the security parameter. We formalize the notion of bounded algorithms below.

Definition 3 (Bounded algorithms)**.** Let A be an algorithm such that (i) f-bits are sufficient to describe the code of A, (ii) at any time during its execution, the state of A can be described by at most s bits and (iii) on any input, A runs for at most t time-steps. Then we say that A is a (s, f, t)-bounded algorithm. For such algorithms we have $f_{\mathsf{A}} \le f, s_{\mathsf{A}} \le s$ and $t_{\mathsf{A}} \le t$ (with the obvious meaning). Sometimes, for simplicity, we will call an $(s, \mathrm{poly}(\lambda), \mathrm{poly}(\lambda))$-bounded algorithm just s-space-bounded, an $(s, \mathrm{poly}(\lambda), t)$-bounded algorithm (s,t)-space-time bounded and an $(s, f, \mathrm{poly}(\lambda))$-bounded algorithm (s, f)-total-space-bounded.

Note that the bound f the size of A is also an upper bound on the hardwired auxiliary information. We stress that, similarly to previous works [DKW11a, DKW11b], in case A is modeled as a Turing machine, we count the length of the input tape and the position of all the tape heads within the space bound s. Given an input $x \in \{0,1\}^n$, and an initial configuration $\sigma \in \{0,1\}^{s-n}$, we write $(y, \tilde{\sigma}) := \mathsf{A}(x; \sigma)$ for the output y of A including its final configuration $\tilde{\sigma} \in \{0,1\}^{s-n}$.

Intuitively, a coding scheme can be decoded in bounded space if the decoding algorithm is space bounded.

Definition 4 (Space-bounded decoding)**.** Let $\Pi = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ be a (k, n)-code, and $d \in \mathbb{N}$. We call Π a (k, n)-code with d-space-bounded decoding, if for all pp output by $\mathsf{Init}(1^\lambda)$ the decoding algorithm $\mathsf{Decode}_{\mathsf{pp}}(\cdot)$ is d-space-bounded.

3.4 Random Oracles

All our results are in the random oracle model (ROM). Therefore we first discuss some basic conventions and definitions related to random oracles. This section is taken almost verbatim from [FHMV17a]. First, recall that in the ROM, at setup, a hash function $\mathcal{H}$ is sampled uniformly at random, and all algorithms, including the adversary, are given oracle access to $\mathcal{H}$ (unless stated otherwise). For instance, we let $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$ be a coding scheme in the ROM. Second, without loss of generality, we will always consider a random oracle $\mathcal{H}$ with a type $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$.

We emphasize that unlike many other proofs in the ROM, we will not need the *full programmability* of random oracles in pur proofs. In fact, looking ahead, in the security proof of our code constructions, we can just assume that the random oracle is *non-adaptively programmable* as defined in [BM15].[8] The basic idea is that the simulator/reduction samples a partially defined "random-looking function" at the beginning of the security game, and uses that function as the random oracle $\mathcal{H}$. In particular, by fixing a function ahead of time, the reduction fixes all future responses to random oracle calls—this is in contrast to programmable random oracles, which allow the simulator to choose random values adaptively in the game, and also to program the output of the oracle in a convenient manner. In particular, in most of our proofs we will be simulating the random oracles with a PRF.

We additionally make the following convention. Random oracle queries are stored in query tables. Let $\mathcal{Q}_{\mathcal{H}}$ be such a table. $\mathcal{Q}_{\mathcal{H}}$ is initialized as $\mathcal{Q}_{\mathcal{H}} := \emptyset$. Hence, when the random oracle $\mathcal{H}$ is queried on a value u, a new tuple $(\mathsf{bit}(u), u, \mathcal{H}(u))$ is appended to the table $\mathcal{Q}_{\mathcal{H}}$ where $\mathsf{bit} : \{0,1\}^* \to \{0,1\}^{O(\log \lambda)}$ is an injective function that maps each input u to a unique identifier, represented in bits. We call $\mathsf{bit}(u)$ the index of $(u, \mathcal{H}(u))$ in $\mathcal{Q}_{\mathcal{H}}$. Clearly, for any tuple $(i, u, \mathcal{H}(u))$ we have that $\mathsf{bit}^{-1}(i) = u$.

4 Continuous Space-bounded Tampering

Space-bounded Tampering algorithms. We assume that tampering algorithms are deterministic[9], sequential and (s, f)-total-space-bounded, where $s, f \in \mathbb{N}$ are tunable parameters and are usually functions of the security parameter λ. Let us denote the class of all such algorithms by $\mathcal{A}^{s,f}_{\mathsf{space}}$. When the context is clear, we might just refer to $\mathcal{A}^{s,f}_{\mathsf{space}}$ by $\mathcal{A}_{\mathrm{space}}$ for simplicity. Generally any $\mathsf{A} \in \mathcal{A}^{s,f}_{\mathsf{space}}$ will be often referred to as a *space-bounded tampering algorithm.*

Oracles. Next we define *space-bounded tampering oracle with self-destruct.* In contrast with [FHMV17a] (Definition 5) our tampering oracle has the "self-destruct" mechanism.

Definition 5 (Space-bounded Tampering Oracle with Self-destruct)**.** A *space-bounded tampering oracle with self-destruct* $\mathcal{O}^{\Pi,x,\mathsf{pp},s,f,p}_{\text{real-sd}}$ is parametrized by a (k, n)-code $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$, a string $x \in \{0,1\}^k$, public parameters $\mathsf{pp} \in \{0,1\}^*$ and integers $s, p \in \mathbb{N}$ (with $s \geq p \geq n$). Initially, the oracle assigns a flag $\mathsf{sd} := 0$, and sets a state $\mathsf{st} := (c, \sigma)$, where $c := \mathsf{Encode}^{\mathcal{H}}(\mathsf{pp}, x)$,

[8] In [BM15], the authors show that such random oracles are equivalent to non-programmable ones, as defined in [FLR+10].

[9] This is without loss of generality, as in the tampering setting A is chosen by PPT distinguisher D ("big adversary" in our case) who can just hardwires its truly random coin to A.

and $\sigma := \sigma_0||\sigma_1 := 0^{p-n}||0^{s-p}$. Given input a space-bounded tampering algorithm $\mathsf{A} \in \mathcal{A}^{s,f}_{\mathsf{space}}$, the oracle works as follows:

$\underline{\text{Oracle } \mathcal{O}^{\Pi,x,\mathsf{pp},s,f,p}_{\text{real-sd}}(\mathsf{A}):}$
Parse $\mathtt{st} = (c, \sigma_0, \sigma_1)$
$(\tilde{c}, \tilde{\sigma}_0, \tilde{\sigma}_1) := \mathsf{A}^{\mathcal{H}}(c; \sigma_0||\sigma_1)$
Update $\mathtt{st} := (\tilde{c}, \tilde{\sigma}_0, 0^{s-p})$
$\tilde{x} := \mathsf{Decode}^{\mathcal{H}}(\mathsf{pp}, \tilde{c})$; If $\tilde{x} = \perp$ then $\mathsf{sd} := 1$
If $\mathsf{sd} = 1$ return $\perp$
Return $\tilde{x}$.

Remark 1. *We assume that the tampering algorithm has access to the public parameter but we do not explicitly hardwire it inside* A *and hence it is neither accounted for in the bound* f *nor in* s*. Similar to the setting of non-malleable codes in the common random string model (cf. [LL12, FMNV14])* pp *is considered to be untamperable. In reality* pp *can be part of a* read-only memory *to which the attacker does not have write access. We do not formalize that explicitly.*

We recall from [FHMV17a] the definitions of the leakage $\mathcal{O}^{\ell,x}_{\text{leak}}$ that can be queried in order to retrieve up-to ℓ bits of information about x and the simulation oracle which would use the leakage oracle to simulate the output of the tampering experiment.

Definition 6 (Leakage oracle)**.** A *leakage oracle* $\mathcal{O}^{\ell,x}_{\text{leak}}$ is a stateful oracle that maintains a counter $\mathtt{ctr}$ that is initially set to 0. The oracle is parametrized by a string $x \in \{0,1\}^k$ and a value $\ell \in \mathbb{N}$. When $\mathcal{O}^{\ell,x}_{\text{leak}}$ is invoked on a polynomial-time computable leakage function L, the value $L(x)$ is computed, its length is added to $\mathtt{ctr}$, and if $\mathtt{ctr} \leq \ell$, then $L(x)$ is returned; otherwise, $\perp$ is returned.

Definition 7 (Simulation oracle)**.** A *simulation* oracle $\mathcal{O}^{\mathsf{S}_2,\ell,x,s,f,\mathsf{pp}}_{\text{sim}}$ is an oracle parametrized by a stateful PPT algorithm S_2, values $\ell, s \in \mathbb{N}$, some string $x \in \{0,1\}^k$, and public parameters $\mathsf{pp} \in \{0,1\}^*$. Upon input a space-bounded tampering algorithm $\mathsf{A} \in \mathcal{A}^{s,f}_{\mathsf{space}}$, the output of the oracle is defined as follows.

$\underline{\text{Oracle } \mathcal{O}^{\mathsf{S}_2,\ell,x,s,f,\mathsf{pp}}_{\text{sim}}(\mathsf{A}):}$
Let $\tilde{x} \leftarrow \mathsf{S}_2^{\mathcal{O}^{\ell,x}_{\text{leak}}(\cdot)}(1^\lambda, \mathsf{pp}, \mathsf{A})$
If $\tilde{x} = \mathsf{same}^\star$ set $\tilde{x} := x$.
Return $\tilde{x}$.

Space-bounded Continuous Non-malleability. Our definition is broadly the same as in [FHMV17a] with slight modifications: here the real tampering oracle $\mathcal{O}_{\text{real-sd}}$ has self-destruct in it and we consider a concrete non-malleability error-bound $\varepsilon_{\mathsf{nm}}$.

Definition 8 (Space-bounded continuous non-malleability with self-destruct)**.** For parameters $k, n, \ell, s, f, p, \theta, d, n_{\mathcal{H}} \in \mathbb{N}$ (with $s \geq p \geq n$) and $\varepsilon_{\mathsf{nm}} \in [0,1)$ let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a random oracle, then we say a (k,n)-code $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$ is an ℓ-leaky (s,f,p)-space-bounded[10] $(\theta, \varepsilon_{\mathsf{nm}})$-continuously non-malleable code with self-destruct with d-space-bounded decoding (or $(\ell, s, f, p, \theta, d, \varepsilon_{\mathsf{nm}})$-SP-NMC-SD) in the ROM if Π satisfies the following conditions:

[10]Note that the terminology "space-bounded" is slightly overloaded as we use it both for an encoding scheme as well as for an algorithm (cf. Definition 3.)

- **Space-bounded decoding:** $\mathsf{Decode}^{\mathcal{H}}$ is d-space-bounded.
- $(\ell, \theta, \varepsilon_{\mathsf{nm}})$**-continuous non-malleability:** For any PPT distinguisher D that makes at most θ queries to the tampering oracle $\mathcal{O}_{\text{real-sd}}$, there exists a pair of PPT algorithms (also called the simulator) $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$, such that for all $x \in \{0,1\}^k$ and $\lambda \in \mathbb{N}$,

$$\Big|\Pr\left[\mathsf{D}^{\mathcal{H}(\cdot),\mathcal{O}^{\Pi,x,\mathsf{pp},s,f,p}_{\text{real-sd}}(\cdot)}(\mathsf{pp}) = 1 :\ \mathsf{pp} \leftarrow \mathsf{Init}^{\mathcal{H}}(1^\lambda)\right] - \Pr\left[\mathsf{D}^{\mathsf{S}_1(\cdot),\mathcal{O}^{\mathsf{S}_2,\ell,x,s,f,\mathsf{pp}}_{\text{sim}}(\cdot)}(\mathsf{pp}) = 1 :\ \mathsf{pp} \leftarrow \mathsf{Init}^{\mathsf{S}_1}(1^\lambda)\right]\Big| \leq \varepsilon_{\mathsf{nm}},$$

  the randomness coming from $\mathcal{H}$, Init, D, $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ and encoding of $\mathcal{O}_{\text{real-sd}}$.

Remark 2. *The roles of S_1 and S_2 are the same as in [FHMV17a]. Intuitively, S_1 simulates all the random oracle queries and S_2 simulates the tampering queries with the help of the leakage oracle. We implicitly assume that S_1 and S_2 may share states. For readers familiar with the notion of non-malleable codes in the common reference string model (c.f. [LL12, FMNV14]), we remark that the simulator is not required to program the public parameters (but is instead allowed to program the random oracle).*

We are interested in constructing an encoding scheme which satisfies Definition 8 with any choice of $\theta = \text{poly}(\lambda)$. Recall from Section 3.2 of [FHMV17a] that, in this case, self-destruct is necessary in order to achieve a meaningful notion of non-malleability as otherwise whenever $\theta \geq n$ it is impossible to achieve space-bounded non-malleability for any non-trivial[11] leakage ℓ.

Remark 3. *We notice that, if the parameter f is such that adversaries from $\mathcal{A}^{s,f}_{\mathsf{space}}$ can have valid encodings (c_1, c_2) of two different messages hard-wired into their description, the leakage ℓ will be (approximately) at least as big as the size of the "extra persistent space" $p - n$. Otherwise, the distinguisher D can send a space-bounded adversary A with hard-wired (c_1, c_2), which first copies any $p - n$ bits of the target message into the extra persistent space and then depending on the first bit of that overwrites the target encoding with c_1 or c_2. Repeating this process, D learns the entire extra persistent space after only a few ($\theta \approx p - n$) tampering. Therefore, in our setting where we allow sufficiently large f and any unbounded $\theta \in \text{poly}(\lambda)$, the leakage ℓ always contains $p - n$ as an additive factor.*

5 Non-Interactive Proof of Space (NIPoS)

As in [FHMV17a], the main building block of our NMC construction is Non-Interactive Proof of Space (for short NIPoS). Intuitively, a NIPoS allows a prover to convince a verifier that she has a lot of space/memory. Importantly, the verification done on the verifier's side is space efficient.

We start by recalling the definition of NIPoS from [FHMV17a] adjusted to (s, f, t)-bounded algorithms. We split the definitions completeness and extractability here. Then we define property called *proof-extractability*. We made some syntactical changes to the definition of extractability to align it with the proof-extractability definition. Finally we define a new quantitative measure of NIPoS called *uniqueness* and show that uniqueness, when combined with extractability gives proof-extractability.

Definition 9 (Non-interactive proof of space (NIPoS))**.** For parameters $s_{\mathsf{P}}, s_{\mathsf{V}}, k_{\mathsf{pos}}, n_{\mathsf{pos}} \in \mathbb{N}$ with $s_{\mathsf{V}} \leq s < s_{\mathsf{P}}$ an $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-non-interactive proof of space scheme (NIPoS for short) in the ROM consists of a tuple of PPT algorithms $(\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ with the following syntax.

[11] Recall that for any non-trivial leakage we must have $\ell \leq k - \omega(\log k)$ as otherwise the tampering adversary learns (almost) all information about the input rendering the notion useless.

- $\mathsf{Setup}^{\mathcal{H}}(1^\lambda)$: This is a randomized polynomial-time (in λ) algorithm with no space restriction. It takes as input the security parameter and outputs public parameters $\mathsf{pp}_{\mathsf{pos}} \in \{0,1\}^*$.
- $\mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id)$: This is a probabilistic polynomial-time (in λ) algorithm that is s_{P}-space-bounded. It takes as input an identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$ and hard-wired public parameters $\mathsf{pp}_{\mathsf{pos}}$, and it returns a proof of space $\pi \in \{0,1\}^{n_{\mathsf{pos}}}$.
- $\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id,\pi)$: This algorithm is s_{V}-space-bounded and deterministic. It takes as input an identity id, hard-wired public parameters $\mathsf{pp}_{\mathsf{pos}}$, and a candidate proof of space π, and it returns a decision bit.

We require completeness to hold:

Completeness: For all $id \in \{0,1\}^{k_{\mathsf{pos}}}$, we have that

$$\Pr\left[\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id,\pi)=1 :\ \mathsf{pp}_{\mathsf{pos}} \leftarrow \mathsf{Setup}^{\mathcal{H}}(1^\lambda); \pi \leftarrow \mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id)\right] = 1,$$

where the probability is taken over the internal random coins of the algorithms Setup and P, and over the choice of the random oracle.

We define the extractability of a NIPoS separately as follows.

Definition 10 (Extractability of NIPoS)**.** Let $\mathsf{NIPoS} = (\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ be an $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-non-interactive proof of space scheme. Let $s, f, t, \eta \in \mathbb{N}$ and $\varepsilon_{\mathsf{pos}} \in [0,1)$ be parameters with $s_{\mathsf{V}} \leq s < s_{\mathsf{P}}$. Then we say that NIPoS is $(s,f,t,\eta,\varepsilon_{\mathsf{pos}})$-extractable (Ext-NIPoS) if there exists a polynomial-time deterministic algorithm K (the knowledge extractor) and a deterministic efficiently computable function $F_{\mathsf{hint}} : \{0,1\}^* \to \{0,1\}^\eta$ such that for any probabilistic polynomial-time algorithm B, we have

$$\Pr[\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda)=1] \leq \varepsilon_{\mathsf{pos}},$$

for the game $\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda)$ defined as follows:

Game $\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda)$:

1. Sample $\mathsf{pp}_{\mathsf{pos}} \leftarrow \mathsf{Setup}^{\mathcal{H}}(1^\lambda)$ and $\pi \leftarrow \mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id)$.
2. Let $\mathsf{A} \leftarrow \mathsf{B}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id,\pi)$ such that $\mathsf{A} \in \mathcal{A}^{s,f}_{\mathsf{space}}$ (if this condition fails, then output 0 and stop).
3. Let $(\tilde{id},\tilde{\pi}) := \mathsf{A}^{\mathcal{H}}(id,\pi)$.
4. Let $z := F_{\mathsf{hint}}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), \tilde{id})$.
5. Let $\alpha := \mathsf{K}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), z)$.
6. Output 1 if and only if: (i) $\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(\tilde{id},\tilde{\pi}) = 1$; (ii) $\tilde{id} \neq id$ and (iii) $\tilde{id} \neq \alpha$; otherwise output 0,

where the set $\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$ contains the sequence of queries of B to $\mathcal{H}$ and the corresponding answers, and where the probability is taken over the coin tosses of $\mathsf{Setup}, \mathsf{B}, \mathsf{P}$ and over the choice of the random oracle.

Remark 4. *Note that, we made two changes from the one used in FHMV: first we introduce a new hint-producing function which can return a "small string" (for example, the the index of $\tilde{id}$ in the RO table $\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$) and given that hint, the extractor can now find out the target identity from the same table. Secondly, the extractor here only returns the target identity instead of all identities found in the table (as done in FHMV). We stress that it is important for the parameter η to be small, as otherwise the definition is trivially satisfied. Looking ahead, the final measure of leakage in the NMC construction will be controlled by the size of this hint.*

Extractability guarantees that if the space bounded adversary A successfully tampers to a new pair $(\tilde{id}, \tilde{\pi})$, the identity $\tilde{id}$ can be extracted from the query table of the algorithm B, i.e., the pair $(\tilde{id}, \tilde{\pi})$ was (partially) precomputed by B. Let us stress that knowledge of $\tilde{id}$ does not generally imply knowledge of the entire pair $(\tilde{id}, \tilde{\pi})$. This is because there might be many different $\tilde{\pi}$ for which $\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(\tilde{id}, \tilde{\pi}) = 1$, unless, of course, there is a unique such $\tilde{\pi}$. In order guarantee extraction of the entire pair $(\tilde{id}, \tilde{\pi})$, we need NIPoS to satisfy a stronger extractability property, which we call Proof-Extractability and define next.

Definition 11 (Proof-Extractability of NIPoS)**.** Let $\mathsf{NIPoS} := (\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ be a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-non-interactive proof of space scheme. Let $s, f, t, \eta \in \mathbb{N}$ and $\varepsilon_{\mathsf{p\text{-}ext}} \in [0, 1)$ be parameters such that $s_{\mathsf{V}} \leq s < s_{\mathsf{P}}$. Then NIPoS is called $(s, f, t, \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof extractable (PExt-NIPoS) if there exists a polynomial time deterministic algorithm K (the proof-extractor) and an efficiently computable deterministic function $F_{\mathsf{hint}} : \{0,1\}^* \to \{0,1\}^{\eta}$ such that for any PPT algorithm B and any identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$, it holds that

$$\Pr[\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda) = 1] \leq \varepsilon_{\mathsf{p\text{-}ext}},$$

for the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$ defined as follows:

Game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$:

1. Sample $\mathsf{pp}_{\mathsf{pos}} \leftarrow \mathsf{Setup}^{\mathcal{H}}(1^{\lambda})$ and $\pi \leftarrow \mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id)$.
2. Let $\mathsf{A} \leftarrow \mathsf{B}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id, \pi)$ such that $\mathsf{A} \in \mathcal{A}^{s,f}_{\mathsf{space}}$ (if this condition fails, then output 0 and stop).
3. Let $(\tilde{id}, \tilde{\pi}) := \mathsf{A}^{\mathcal{H}}(id, \pi)$.
4. Let $z := F_{\mathsf{hint}}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), (\tilde{id}, \tilde{\pi}))$
5. Let $\alpha := \mathsf{K}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), z)$
6. Output 1 if and only if: (i) $\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(\tilde{id}, \tilde{\pi}) = 1$; (ii) $\tilde{id} \neq id$ and (iii) $(\tilde{id}, \tilde{\pi}) \neq \alpha$; otherwise output 0,

where the set $\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$ is the random oracle query table of B.[12] The probability is over the choice of the random oracle, and the coin tosses of $\mathsf{Setup}, \mathsf{B}$.

Remark 5. *Note that, in the above definition the hint-producing function takes the pair $(\tilde{id}, \tilde{\pi})$ as opposed to only $\tilde{id}$ as in Definition 10. Intuitively this means that, given some small hint, the extractor does not only return the changed identity, but the identity-proof pair. Clearly this makes the later definition stronger.*

As mentioned above, when there is a unique valid proof corresponding to each identity, then proof-extractability reduces to simply extractability. Nevertheless, it may also be possible that only a part of the proof is uniquely determined. We formalize this by the following definition.

[12] Note that B does not make RO queries after outputting the small adversary A.

Definition 12 (Uniqueness of NIPoS)**.** Let $\mathsf{NIPoS} := (\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ be a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS. Then NIPoS is called $(u_{\mathsf{pos}}, \varepsilon_{\mathsf{unique}})$-unique (where $u_{\mathsf{pos}} \leq n_{\mathsf{pos}}$, $u_{\mathsf{pos}} \in \mathbb{N}$ and $\varepsilon_{\mathsf{unique}} \in \mathrm{negl}(\lambda)$) if for any $\lambda \in \mathbb{N}$, there is a deterministic function $J : \{0,1\}^* \times \{0,1\}^{k_{\mathsf{pos}}} \to \{0,1\}^{u_{\mathsf{pos}}}$ such that for $\mathsf{pp}_{\mathsf{pos}} \leftarrow \mathsf{Setup}^{\mathcal{H}}(\lambda)$, any identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$ and any $\pi \in \{0,1\}^{n_{\mathsf{pos}}}$, if $\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}}(id, \pi) = 1$, then $J(\mathsf{pp}_{\mathsf{pos}}, id) = \pi[1 \ldots u_{\mathsf{pos}}]$ with probability at least $1 - \varepsilon_{\mathsf{unique}}$ (where the probability is over the randomnesses of $\mathsf{Setup}^{\mathcal{H}}$ and $\mathsf{P}^{\mathcal{H}}$).

Remark 6. *Intuitively, the definition says that for a valid proof π, a part of π (first u_{pos} bits in this case) can be* uniquely *and* efficiently *determined given the id and the public parameters* pp *with overwhelming probability.*

In the following lemma, we formally show that uniqueness and extractability together imply proof-extractability. To see this, observe that, e.g., maximal uniqueness implies that given $\tilde{id}$, the corresponding $\pi_{\tilde{id}}$ is fixed and hence it suffices to provide the PExt-NIPoS hint-producer only with $\tilde{id}$.

Lemma 2. *Let* $\mathsf{NIPoS} := (\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ *be a* $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$*-NIPoS that is* $(u_{\mathsf{pos}}, \varepsilon_{\mathsf{unique}})$*-unique and* $(s, f, t, \eta, \varepsilon_{\mathsf{pos}})$*-extractable. Then* NIPoS *is* $(s, f, t, \eta', \varepsilon_{\mathsf{p\text{-}ext}})$*-proof-extractable where*

$$\eta' = \eta + n_{\mathsf{pos}} - u_{\mathsf{pos}} \qquad \varepsilon_{\mathsf{p\text{-}ext}} \leq \varepsilon_{\mathsf{pos}} + \varepsilon_{\mathsf{unique}}$$

Proof. Given the hint-producing function F_{hint} and the extractor K for the extractable NIPoS we construct a hint-producing function F'_{hint} and an extractor K' for proof-extractability as follows:

$\underline{F'_{\mathsf{hint}}}$: It takes $(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), (\tilde{id}, \tilde{\pi}))$ as input and runs F_{hint} on $(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), \tilde{id})$ to obtain ℓ. It returns ℓ' where $\ell' = \ell \| \beta$ for $\beta := \tilde{\pi}[u_{\mathsf{pos}} + 1 \ldots n_{\mathsf{pos}}]$.

$\underline{\mathsf{K}'}$: It takes $(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), \ell')$ as input, parses $\ell' = \ell \| \beta$. Then it runs $\alpha := \mathsf{K}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), \ell)$ and $\gamma := J(\mathsf{pp}_{\mathsf{pos}}, \alpha)$. It returns (α, α') where $\alpha' = \gamma \| \beta$.

It is clear that, whenever the standard extractor K is able to extract the correct $\tilde{id}$, the proof-extractor will be able to extract the correct pair $(\tilde{id}, \tilde{\pi})$ as the part of $\tilde{\pi}$ was uniquely derived by J and the rest of it is obtained from the additional hint. Therefore, the only case when K' would fail is exactly when (i) K fails or (ii) J fails. Hence we have $\varepsilon_{\mathsf{p\text{-}ext}} \leq \varepsilon_{\mathsf{pos}} + \varepsilon_{\mathsf{unique}}$. On the other hand, since F'_{hint} needs to output an additional hint of $\tilde{\pi}[u_{\mathsf{pos}} + 1 \ldots n_{\mathsf{pos}}]$ the hint η' is more than η by exactly $n_{\mathsf{pos}} - u_{\mathsf{pos}}$ and hence we have $\eta' = \eta + n_{\mathsf{pos}} - u_{\mathsf{pos}}$. This completes the proof.

□

6 Space-bounded NMC from Proof-Extractable NIPoS

We are now prepared to show that the encoding scheme of FHMV satisfies Definition 8 for any unbounded $\theta \in \mathrm{poly}(\lambda)$ when instantiated with any PExt-NIPoS. For completeness, we first recall the FHMV construction. This is taken (almost) verbatim from [FHMV17a].

The Encoding Scheme of [FHMV17a]. Let $(\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ be a $(k, n, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS in the ROM where $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ denotes the random oracle for some $n_{\mathcal{H}} \in \mathrm{poly}(\lambda)$. We define a (k, n)-coding scheme $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$ as follows.

$\mathsf{Init}^{\mathcal{H}}(1^\lambda)$**:** Given as input a security parameter λ, it generates the public parameters for the NIPoS as $\mathsf{pp}_{\mathsf{pos}} \leftarrow\!\!\$\ \mathsf{Setup}^{\mathcal{H}}(1^\lambda)$, and outputs $\mathsf{pp} := \mathsf{pp}_{\mathsf{pos}}$.

$\mathsf{Encode}^{\mathcal{H}}_{\mathsf{pp}}(x)$**:** Given as input the public parameters $\mathsf{pp} = \mathsf{pp}_{\mathsf{pos}}$ and a message $x \in \{0,1\}^k$, it runs the NIPoS prover to generate a proof of space $\pi \leftarrow \mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(x)$ using the message x as identity. Then it outputs $c := (x, \pi) \in \{0,1\}^n$ as a codeword.

$\mathsf{Decode}^{\mathcal{H}}_{\mathsf{pp}}(c)$**:** Given a codeword c, it first parses c as (x, π). Then it runs the NIPoS verifier $b := \mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(x, \pi)$. If $b = 1$ it outputs x, otherwise it outputs $\perp$.

The following theorem formally states that the above construction is a continuous non-malleable code for any $\theta \in \mathrm{poly}(\lambda)$.

Theorem 1. *Let λ be a security parameter and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a hash function modeled as a random oracle. Let $\{\mathsf{PRF}_\chi : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}\}_{\chi \in \{0,1\}^{n_{\mathsf{key}}}}$ be any $(*, n_{\mathcal{H}}, n_{\mathsf{key}}, \varepsilon_{\mathsf{pr}})$-PRF, where $n_{\mathsf{key}} \in \mathrm{poly}(\lambda)$. Let $(\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ be any $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, f, \mathrm{poly}(\lambda), \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof-extractable. Then for any $\theta \in \mathrm{poly}(\lambda)$, the (k, n)-code $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$ defined above is an $(\ell, s, f, p, \theta, s_{\mathsf{V}}, \varepsilon_{\mathsf{nm}})$-SP-NMC-SD in the ROM, where*

$$
\begin{aligned}
k &= k_{\mathsf{pos}} \\
n &= k_{\mathsf{pos}} + n_{\mathsf{pos}} \\
k_{\mathsf{pos}} + n_{\mathsf{pos}} &\le p < 2n - O(\log(\lambda)) \\
\ell &= p - n + \lceil \log \theta \rceil + \eta \\
\varepsilon_{\mathsf{nm}} &\le \varepsilon_{\mathsf{pr}} + \varepsilon_{\mathsf{p\text{-}ext}}
\end{aligned}
$$

The above theorem together with Lemma 2 imply that the encoding scheme of Faust et al. satisfies Definition 8 also when instantiated with any Ext-NIPoS with (partial) uniqueness. This is formalized in the following corollary:

Corollary 1. *Let λ be a security parameter and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a hash function modeled as a random oracle. Let $\{\mathsf{PRF}_\chi : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}\}_{\chi \in \{0,1\}^{n_{\mathsf{key}}}}$ be any $(*, n_{\mathcal{H}}, n_{\mathsf{key}}, \varepsilon_{\mathsf{pr}})$-PRF where $n_{\mathsf{key}} \in \mathrm{poly}(\lambda)$. Let $(\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ be any $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, \mathrm{poly}(\lambda), \mathrm{poly}(\lambda), \eta, \varepsilon_{\mathsf{pos}})$-extractable and $(u_{\mathsf{pos}}, \varepsilon_{\mathsf{unique}})$-unique. Then for any $\theta \in \mathrm{poly}(\lambda)$, the (k, n)-code $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$ of FHMV is an $(\ell, s, \mathrm{poly}(\lambda), p, \theta, s_{\mathsf{V}}, \varepsilon_{\mathsf{nm}})$-SP-NMC-SD in the ROM, where*

$$
\begin{aligned}
&k = k_{\mathsf{pos}} \qquad n = k_{\mathsf{pos}} + n_{\mathsf{pos}} \qquad k_{\mathsf{pos}} + n_{\mathsf{pos}} \le p < n + k - O(\log(\lambda)) \\
&\ell = p - k - u_{\mathsf{pos}} + \lceil \log \theta \rceil + \eta + 2 \qquad\qquad \varepsilon_{\mathsf{nm}} = \varepsilon_{\mathsf{pr}} + \varepsilon_{\mathsf{pos}} + \varepsilon_{\mathsf{unique}}.
\end{aligned}
$$

Proof of Theorem 1. We remark that, due to close similarity with the proof of [FHMV17a, Theorem 3], we will be using many texts that are taken verbatim from [FHMV17a].

We fix the oracle output length $n_{\mathcal{H}} \in \mathbb{N}$ and the parameters $k, n, \ell, s, f, p, \theta = \mathrm{poly}(\lambda)$ for the (k, n)-code $\Pi = (\mathsf{Init}^{\mathcal{H}}, \mathsf{Encode}^{\mathcal{H}}, \mathsf{Decode}^{\mathcal{H}})$. The correctness of the coding scheme is guaranteed by the perfect completeness of the underlying NIPoS. Moreover, since the decoding algorithm simply runs the verifier of the NIPoS, it is straightforward to observe that decoding is s_{V} bounded.

To prove Theorem 1, it is sufficient to show that there exists an explicit construction of PPT simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ such that the (k, n)-code Π satisfies $(\ell, \theta, \varepsilon_{\mathsf{nm}})$-continuous non-malleability. Before we construct such simulator, let us recall that proof-extractability of the underlying NIPoS implies existence of a hint-producing function F_{hint} and a knowledge extractor K. On high level, the hint-producing function takes as input a valid pair (id, π) and a table of random oracle queries Q and outputs a string z of length η. Given this hint and the query table Q, the knowledge extractor K is able to reconstruct the pair (id, π). The simulator we now define heavily relies on these two algorithms.

Constructing the simulator. We now describe the simulator $\mathsf{S}^{\mathsf{D}} = (\mathsf{S}_1^{\mathsf{D}}, \mathsf{S}_2^{\mathsf{D}})$, depending on a PPT distinguisher D.[13] A formal definition of the simulator is given in Fig. 1; we provide a high level description of the simulator below.

Informally, algorithm S_1 simulates the random oracle $\mathcal{H}$ by sampling a uniform random key $\chi \leftarrow\!\!\$\ \{0,1\}^{n_{\mathsf{key}}}$ for a pseudorandom function (PRF) $\mathsf{PRF}_\chi : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$; hence, it defines $\mathcal{H}(u) := \mathsf{PRF}_\chi(u)$ for any $u \in \{0,1\}^*$.[14] S_2 receives the description of the RO (i.e., the PRF key χ) from S_1.

The simulator S_2 then makes a query to the leakage oracle with the function L_1 that has hard-coded: the coding scheme Π, the description of the simulated RO, the code of D and the encoding randomness ρ_{enc}. The function L_1 first encodes (using the randomness ρ_{enc}) the target message x to generate a codeword c. Then it runs the code of D to obtain the first tampering algorithm A_1 which it applies to c and obtains a tampered codeword $\tilde{c}$. In case the tampered codeword decodes to the target message x, the leakage function continues running D to obtain the second tampering algorithm A_2 which it applies to $\tilde{c}$. Let $\mathsf{A}_{j^\star}$ be the first tampering algorithm that tampers to a codeword that does not decoded to the target message x. The leakage function L_1 returns the binary representation of $j^\star$ to the simulator S_2. In case no such $\mathsf{A}_{j^\star}$ exists, the L_1 signals this to S_2 by returning the flag 0^{ℓ_1}.

If the output of the leakage oracle if the flag 0^{ℓ_1}, S_2 replies all tampering queries $\mathsf{A}_1, \ldots, \mathsf{A}_\theta$ by outputting the symbol $\mathsf{same}^\star$. Otherwise, the simulator S_2 learns the value $j^\star$. In that case, S_2 answers all the tampering queries $\mathsf{A}_1, \ldots, \mathsf{A}_{j^\star-1}$ by outputting the symbol $\mathsf{same}^\star$. In order to answer the $j^\star$-tampering query, the simulator S_2 makes another query to the leakage oracle. This time with the function L_2 which has hard-coded: the coding scheme Π, the description of the simulated RO, the query table $\mathcal{Q}_{\mathcal{H}}(\mathsf{D})$ consisting of all RO queries made by D (until this point), the code of all tampering algorithms $\mathsf{A}_1, \ldots, \mathsf{A}_{j^\star}$, the encoding randomness ρ_{enc} and the code of the hint-producing function F_{hint}.

The function L_2 first encodes (using the randomness ρ_{enc}) the target message x to generate a codeword c. Then it applies the composed function $\mathsf{A}_{j^\star} \circ \mathsf{A}_{j^\star-1} \circ \cdots \circ \mathsf{A}_1$ on c to generate the tampered codeword $\tilde{c}$ which it decodes to obtain a value $\tilde{x} \neq x$. If $\tilde{x}$ is equal to $\perp$, i.e. $\tilde{c}$ is an invalid codeword, then the leakage function signals this fact by returning the flag 0^{ℓ_2} to the simulator S_2. Otherwise, the leakage function runs the hint-producing function F_{hint} to obtain a string z defining how to reconstruct the tampered codeword from the query table $\mathcal{Q}_{\mathcal{H}}(\mathsf{D})$. The leakage function returns z and $p - n$ bits of the additional persistent space.

If the output of the leakage oracle is the flag 0^{ℓ_2}, the simulator S_2 replies to all tampering queries $\mathsf{A}_{j^\star}, \ldots, \mathsf{A}_\theta$ by $\perp$. Otherwise, the simulator S_2 replies to the $j^\star$-th tampering query by the message $\tilde{x}$, where $(\tilde{x}, \tilde{\pi}) = \tilde{c} := \mathsf{K}(\mathsf{pp}, \mathcal{Q}_{\mathcal{H}}(\mathsf{D}), z)$. S_2 recovers the entire persistent space and, hence, is able to answer all follow up tampering queries without any further leakage query.

Some intuitions. Firstly, note that in the real experiment the random oracle is a truly random function, whereas in the simulation random oracle queries are answered using a PRF. However, using the security of the PRF, we can move to a mental experiment that is exactly the same as the simulated game, but replaces the PRF with a truly random function.

Secondly, observe that if none of the tampering algorithms A_i tampers with the codeword or none of the algorithms tampers to a codeword $\tilde{c}$ which decodes to a different message than the target message x, i.e. the first leakage function returns 0^{ℓ_1}, then the simulator perfectly simulates the real experiment since it answers all the tampering queries by the symbol $\mathsf{same}^\star$ which is exactly what the tampering oracle does.

[13] In the rest of the proof we drop the superscript D, and just let $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$.

[14] Such a PRF can be instantiated using any PRF with fixed domain, and then applying the standard Merkle-Damgård transformation to extend the input domain to arbitrary-length strings.

Simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$

1. Let $\mathsf{PRF}_\chi : \{0,1\}^* \to \{0,1\}^{n_\mathcal{H}}$ be a PRF. The simulator S_1 samples a uniform random key $\chi \leftarrow \{0,1\}^{n_{\mathsf{key}}}$ and defines $\mathcal{H} := \mathsf{PRF}_\chi$. The query table $\mathcal{Q}_\mathcal{H}(\mathsf{D})$ is initially empty.

2. S_2 receives χ from S_1. Then it queries the leakage oracle on the following leakage function:

   Leakage function $L_1 : \{0,1\}^* \to \{0,1\}^{\ell_1}$

   L_1 is hard-coded with the description of $\mathcal{H}$ (i.e. with PRF_χ) and thus it consistently answers all RO queries from D and all A_i. Furthermore, it is hard-coded with the encoding scheme Π, the code of D and the encoding randomness ρ_{enc}.

   (a) Produce the codeword $c \leftarrow\!\!\$\ \mathsf{Encode}^\mathcal{H}_{\mathsf{pp}}(x)$ using randomness ρ_{enc} and initialize the auxiliary space $\sigma := \sigma_0||\sigma_1 := 0^{p-n}||0^{s-p}$. Let $i := 1$ and $j^\star := 0$.

   (b) If $i \leq \theta$, do as follows:

      i. Run D to get tampering algorithm A_i. Let $(\tilde{c}, \tilde{\sigma}_0||\tilde{\sigma}_1) := \mathsf{A}_i(c; \sigma_0||\sigma_1)$.

      ii. Compute $\tilde{x} := \mathsf{Decode}^\mathcal{H}_{\mathsf{pp}}(\tilde{c})$.

      iii. If $\tilde{x} \neq x$, then set $j^\star := i$ and go to Step 2c. Else increment i by 1, define $(c, \sigma_0||\sigma_1) := (\tilde{c}, \tilde{\sigma}_0||0^{s-p})$ and go back to Step 2b.

   (c) Return $\zeta_1 := \mathsf{bit}(j^\star)$.

3. Once S_2 obtains $\zeta_1 \in \{0,1\}^{\ell_1}$, it proceeds as follows. If $\zeta_1 = 0^{\ell_1}$, then it sets $j^\star := \theta + 1$, else it sets $j^\star := \mathsf{bit}^{-1}(\zeta_1)$. Then it returns $\mathsf{same}^\star$ for all queries A_i, where $0 < i < j^\star$. If $j^\star = \theta + 1$, then the simulator stops. Else it proceeds to Step 4.

4. For the $j^\star$-th tampering query $\mathsf{A}_{j^\star}$, S_2 makes another call to its leakage oracle with the following leakage function:

   Leakage function $L_2 : \{0,1\}^* \to \{0,1\}^{\ell_2}$:

   L_2 is hard-coded with the description of $\mathcal{H}$ (i.e., with PRF_χ), the table $\mathcal{Q}_\mathcal{H}(\mathsf{D})$, the code of $(\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_{j^\star})$, the encoding scheme Π, the code of the hint-producing function F_{hint} of the NIPoS and the same encoding randomness ρ_{enc}.

   (a) Produce the codeword $c \leftarrow\!\!\$\ \mathsf{Encode}^\mathcal{H}_{\mathsf{pp}}(x)$ using the same ρ_{enc} and initialize the auxiliary space $\sigma := \sigma_0||\sigma_1 := 0^{p-n}||0^{s-p}$.

   (b) Let $\widetilde{\mathsf{A}} := \mathsf{A}_{j^\star} \circ \mathsf{A}_{j^\star - 1} \circ \cdots \circ \mathsf{A}_1$. Run $\widetilde{\mathsf{A}}$ to get $(\tilde{c}, \tilde{\sigma}_0||\tilde{\sigma}_1) := \widetilde{\mathsf{A}}(c; \sigma_0||\sigma_1)$.

   (c) Compute $\tilde{x} := \mathsf{Decode}^\mathcal{H}_{\mathsf{pp}}(\tilde{c})$. If $\tilde{x} = \perp$, output the flag 0^{ℓ_2}, otherwise run $z := F_{\mathsf{hint}}(\mathsf{pp}, \mathcal{Q}_\mathcal{H}(\mathsf{D}), \tilde{c})$ and return $\zeta_2 := z \parallel \tilde{\sigma}_0 \parallel 1$.

5. Depending on the ℓ_2-bit string ζ_2, the simulator S_2 proceeds as follows:

   (a) If $\zeta_2 = 0^{\ell_2}$, then S_2 returns $\perp$ for all tampering queries A_i, where $i \geq j^\star$.

   (b) Else S_2 parses ζ_2 as $z \parallel \tilde{\sigma}_0 \parallel 1$, runs $(\tilde{x}, \tilde{\pi}) \leftarrow \mathsf{K}(\mathsf{pp}, \mathcal{Q}_\mathcal{H}(\mathsf{D}), z)$ and outputs $\tilde{x}$ for the $j^\star$-th tampering query $\mathsf{A}_{j^\star}$. S_2 it reconstructs the entire memory (c, σ) as $c := (\tilde{x}, \tilde{\pi})$ and $\sigma := \tilde{\sigma}_0||0^{s-p}$. It answers all the follow up tampering queries A_i, where $i > j^\star$, without any further access to the leakage oracle, as follows

      i. Run A_i to obtain $(\tilde{c}, \tilde{\sigma}_0||\tilde{\sigma}_1) := \mathsf{A}_i(c; \sigma_0||\sigma_1)$.

      ii. Output $\mathsf{Decode}^\mathcal{H}_{\mathsf{pp}}(\tilde{c})$ and set $(c, \sigma_0||\sigma_1) := (\tilde{c}, \tilde{\sigma}_0||0^{s-p})$.

Figure 1: Description of the simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$

Assume now that the first leakage query returns $\mathsf{bit}(j^\star)$ such that $0 < j^\star \leq \theta$. Intuitively, the only case in which the simulation strategy of answering the $j^\star$-th tampering query goes wrong is when the tampered codeword $\tilde{c}$ is valid, but the knowledge extractor K fails to reconstruct the codeword $\tilde{c}$. We denote this event as NotExtr. We prove that NotExtr occurs exactly when the adversary D violates the proof-extractibility property of the underlying NIPoS, which happens only with negligible probability.

Finally, if the simulator answers reconstructs the tampered codeword after the $j^\star$-th tampering query, it can answer all the follow up tampering queries perfectly.

The detailed formal analysis is given in Appendix A.

7 Constructing Proof-Extractable NIPoS from CHG

In this section, we present a concrete Proof-Extractable NIPoS construction whose main building blocks are: (i) a novel family of hard-to-pebble graphs called *challenge-hard graphs* (CHG) and (ii) Merkle Commitments. For completeness, we first recall the definition of Merkle Commitments and the concept of graph pebbling. Thereafter, we introduce and formally define CHGs in Section 7.3. Our NIPoS construction is then presented in Section 7.4 and the proof that it satisfies Proof-Extractablility in given in Appendix B. We conclude this section by presenting and comparing two concrete CHG constructions and stating the concrete parameters of the resulting Proof-Extractable NIPoS.

7.1 Merkle Commitments

For completeness, we adapt the notations from [FHMV17a] and briefly recall the construction of Merkle commitment [Mer88] from there. Merkle commitment is built upon a hash function $\mathcal{H}_{\mathsf{com}}$ which will be modeled as a random oracle throughout our paper. Intuitively, during the committing phase, a sender exploits a so-called hash tree to commit a vector of N elements $\mathbf{z} := (z_1, \ldots, z_N)$ using $N-1$ invocation of $\mathcal{H}_{\mathsf{com}}$. At a later point, one can open any of the values z_i by providing the hash labels of a Merkle tree path with size logarithmic in N.

Definition 13 (Merkle commitment)**.** An $(n_{\mathcal{H}}, N)$-Merkle commitment scheme (or MC scheme) in the ROM is a tuple of algorithms $(\mathsf{MCom}^{\mathcal{H}_{\mathsf{com}}}, \mathsf{MOpen}^{\mathcal{H}_{\mathsf{com}}}, \mathsf{MVer}^{\mathcal{H}_{\mathsf{com}}})$ described as follows.

- $\mathsf{MCom}^{\mathcal{H}_{\mathsf{com}}}(\mathbf{z})$: On input an N-tuple $\mathbf{z} = (z_1, \ldots, z_N)$, where $z_i \in \{0,1\}^{n_{\mathcal{H}}}$, this algorithm outputs a commitment $\phi \in \{0,1\}^{n_{\mathcal{H}}}$.
- $\mathsf{MOpen}^{\mathcal{H}_{\mathsf{com}}}(\mathbf{z}, i)$: On input a vector $\mathbf{z} = (z_1, \ldots, z_N) \in \{0,1\}^{n_{\mathcal{H}} N}$, and $i \in [N]$, this algorithm outputs an opening $(z_i, \psi) \in \{0,1\}^{(1+\log N) n_{\mathcal{H}}}$.
- $\mathsf{MVer}^{\mathcal{H}_{\mathsf{com}}}(i, \phi, (z, \psi))$: On input an index $i \in [N]$, and a commitment/opening pair $(\phi, (z, \psi))$, this algorithm outputs a decision bit.

7.2 Graph Pebbling and Labeling

We recall basic definitions, facts and lemmas regarding graph pebbling from prior works, in particular from [FHMV17a]. Throughout this paper $G = (V, E)$ is considered to be a directed acyclic graph (DAG), where V is the set of vertices and E is the set of edges of the graph G. Without loss of generality we assume that the vertices of G are ordered lexicographically and are represented by integers in $[N]$, where $N = |V|$. Vertices with no incoming edges are called *input vertices* or *sources*, and vertices with no outgoing edges are called *output vertices* or *sinks*. We

Merkle Commitment

$\mathsf{MCom}^{\mathcal{H}_{\mathsf{com}}}(z_0, \ldots, z_{N-1})$: Output $\phi := \mathsf{Root}^{\mathcal{H}_{\mathsf{com}}}(N, (z_0, \ldots, z_{N-1}))$.

$\mathsf{MOpen}^{\mathcal{H}_{\mathsf{com}}}((z_0, \ldots, z_{N-1}), i)$:

- If $i \equiv 0 \mod 2$ then $\psi_1 := z_{i+1}$; else $\psi_1 := z_{i-1}$.
- For $j = 2$ to $\log(N)$ do
  - $i := i$ div 2
  - If $i \equiv 0 \mod 2$ then $\psi_j := \mathsf{Root}^{\mathcal{H}_{\mathsf{com}}}(2^{j-1}, (z_{(i+1)2^{j-1}}, \ldots, z_{(i+2)2^{j-1}-1}))$; else $\psi_j := \mathsf{Root}^{\mathcal{H}_{\mathsf{com}}}(2^{j-1}, (z_{(i-1)2^{j-1}}, \ldots, z_{i2^{j-1}-1}))$.
- Output $(z_i, (\psi_1, \ldots, \psi_{\log N}))$.

$\mathsf{MVer}^{\mathcal{H}_{\mathsf{com}}}(i, \phi, (z, (\psi_1, \ldots, \psi_{\log N})))$:

- If $i \equiv 0 \mod 2$ then $\phi' := \mathcal{H}_{\mathsf{com}}(z||\psi_1)$; else $\phi' := \mathcal{H}_{\mathsf{com}}(\psi_1||z)$.
- For $j = 2$ to $\log(N)$ do
  - $i := i$ div 2
  - If $i \equiv 0 \mod 2$ then $\phi' := \mathcal{H}_{\mathsf{com}}(\phi'||\psi_j)$; else $\phi' := \mathcal{H}_{\mathsf{com}}(\psi_j||\phi')$.
- If $\phi = \phi'$ then output 1, otherwise output 0.

Figure 2: Construction of a Merkle commitment scheme

denote $\mathsf{deg}(v)$, the set of all predecessors of the vertex v. Formally, $\mathsf{deg}(v) = \{w \in V : (w, v) \in E\}$.

In this section we briefly explain the concept of graph labeling and its connection to the abstract game called graph pebbling which has been introduced in [DNW05]. For more details we refer to [RD16, ABFG14, DFKP15]. We follow the conventions from [RD16] and will use the results from the same. Sometimes for completeness we will use the texts verbatim from the same paper.

Labeling of a graph. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a hash function (often modeled as a random oracle). The $\mathcal{H}$-labeling of a graph G is a function which assigns a label to each vertex in the graph; more precisely, it is a function $\mathsf{label} : V \to \{0,1\}^{n_{\mathcal{H}}}$ which maps each vertex $v \in V$ to a bit string $\mathsf{label}(v) := \mathcal{H}(q_v)$, where we denote by $|\{v^{(1)}, \ldots, v^{(\deg)}\}| = \mathsf{deg}(v)$ and let

$$q_v := \begin{cases} v & \text{if } v \text{ is an input vertex,} \\ v \;||\; \mathsf{label}(v^{(1)}) \;||\; \ldots \;||\; \mathsf{label}(v^{(\deg)}) & \text{otherwise.} \end{cases}$$

An algorithm A labels a subset of vertices $W \subseteq V$ if it computes $\mathsf{label}(W)$. Specifically, A labels the graph G if it computes $\mathsf{label}(V)$.

Additionally, for $m \leq |V|$, we define the $\mathcal{H}$-labeling of the graph G with m faults[15] as a function $\mathsf{label} : V \to \{0,1\}^{n_{\mathcal{H}}}$ such that, for some subset of vertices $M \subset V$ of size m,

$$\mathsf{label}(v) \neq \mathcal{H}(q_v), \text{ for every } v \in M$$
$$\mathsf{label}(v) = \mathcal{H}(q_v), \text{ for every } v \in V \setminus M.$$

[15] One can also define an analogy of faults in the pebbling game by adding a second kind of pebbles. These pebbles are called *red pebbles* in [DFKP15] and *wild cards* in [ABFG14].

Sometimes we refer to labeling with faults as partial labeling.

When $\mathcal{H}$ is modeled as a random oracle one can show an interesting property about graph labeling as we present next as a lemma that appeared in form of a discussion in [RD16]. It is based on an observation previously made in [DFKP15].

Lemma 3 ([RD16, Section 5.2])**.** *Suppose $\mathcal{H}$ is modeled as a random oracle. Let $\mathsf{A}^{\mathcal{H}}$ be an (s,t)-bounded algorithm which computes the labeling of a DAG G with $m \in \mathbb{N}$ faults. Then there exists an $(s + m \cdot n_{\mathcal{H}}, t)$-bounded algorithm $\tilde{\mathsf{A}}^{\mathcal{H}}$ that computes the labeling of G without faults but gets m correct labels to start with (they are initially stored in the memory of $\tilde{\mathsf{A}}^{\mathcal{H}}$ and sometimes called initial labels).*

Intuitively the above lemma follows because the algorithm $\tilde{\mathsf{A}}^{\mathcal{H}}$ can overwrite the additional space it has, once the initial labels stored there are not needed.

Pebbling game. The pebbling of a DAG $G = (V, E)$ is defined as a single-player game. The game is described by a sequence of pebbling configurations $\mathbf{P} = (P_0, \ldots, P_T)$, where $P_i \subseteq V$ is the set of pebbled vertices after the i-th move. In our model, the initial configuration P_0 does not have to be empty. The rules of the pebbling game are the following. During one move (translation from P_i to P_{i+1}), the player can place one pebble on a vertex v if v is an input vertex or if all predecessors of v already have a pebble. After placing one pebble, the player can remove pebbles from arbitrary many vertices.[16] We say that the sequence $\mathbf{P}$ pebbles a set of vertices $W \subseteq V$ if $W \subseteq \bigcup_{i \in [0,T]} P_i$.

The time complexity of the pebbling game $\mathbf{P}$ is defined as the number of moves $t(\mathbf{P}) := T$. The space complexity of $\mathbf{P}$ is defined as the maximal number of pebbles needed at any pebbling step; formally, $s(\mathbf{P}) := \max_{i \in [0,T]}\{|P_i|\}$.

Ex-post-facto pebbling of a DAG (in the ROM). Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a random oracle. Let $\mathsf{A}^{\mathcal{H}}$ be an algorithm that computes the (partial) $\mathcal{H}$-labeling of a DAG G. The ex-post-facto pebbling bases on the transcript of the graph labeling. It processes all oracle queries made by $\mathsf{A}^{\mathcal{H}}$ during the graph labeling (one at a time and in the order they were made). Informally, every oracle query of the form q_v, for some $v \in V$, results in placing a pebble on the vertex v in the ex-post-facto pebbling game. This provides us a link between labeling and pebbling of the graph G. The formal definition follows.

Let $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a random oracle and $\mathcal{Q}_{\mathcal{H}}$ a table of all random oracle calls made by $\mathsf{A}^{\mathcal{H}}$ during the graph labeling. Then we define the *ex-post-facto pebbling* $\mathbf{P}$ *of the graph* G as follows:

- The initial configuration P_0 contains every vertex $v \in V$ such that $\mathsf{label}(v)$ has been used for some oracle query (e.g. some query of the form $\mathcal{H}(\cdots \| \mathsf{label}(v) \| \cdots)$) at some point in the transcript but the query q_v is not listed in the part of the transcript preceding such query.
- Assume that the current configuration is P_i, for some $i \geq 0$. Then find the next unprocessed oracle query which is of the form q_v, for some vertex v, and define P_{i+1} as follows:
  1. Place a pebble on the vertex v.
  2. Remove all *unnecessary* pebbles. A pebble on a vertex v is called unnecessary if $\mathsf{label}(v)$ is not used for any future oracle query, or if the query q_v is listed in the

[16] Similar to [RD16] in our model we assume that removing pebbles is for free as it does not involve any oracle query

succeeding part of the transcript before $\mathsf{label}(v)$ is used in an argument of some other query later. Intuitively, either $\mathsf{label}(v)$ is never used again, or $\mathsf{A}^{\mathcal{H}}$ anyway queries q_v before it is used again.

The lemma below appeared in several variations in the literature (see, for example, [ABFG14, RD16]), depending on the definition of graph pebbling.

Lemma 4 (Labeling Lemma (in the ROM))**.** *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a random oracle. Let G be a DAG. Consider an (s,t)-bounded adversary $\mathsf{A}^{\mathcal{H}}$ which computes the $\mathcal{H}$-labeling of the graph G. Also assume that $\mathsf{A}^{\mathcal{H}}$ does not guess any correct output of $\mathcal{H}$ without querying it. Then the ex-post facto pebbling strategy $\mathbf{P}$ described above pebbles the graph G, and the complexity of $\mathbf{P}$ is*

$$s(\mathbf{P}) \leq \frac{s}{n_{\mathcal{H}}} \quad \textit{and} \quad t(\mathbf{P}) \leq t.$$

Proof. By definition of ex-post-facto pebbling, it is straightforward to observe that if $\mathsf{A}^{\mathcal{H}}$ computes the $\mathcal{H}$-labeling of the graph G, then the ex-post-facto pebbling $\mathbf{P}$ pebbles the graph. Since we assume that the adversary does not guess the correct label, the only way $\mathsf{A}^{\mathcal{H}}$ can learn the label of the vertex v is by querying the random oracle. The bound on $t(\mathbf{P})$ is immediate. Again, by definition of the ex-post-facto pebbling, there is no unnecessary pebble at any time. Thus, the number of required pebbles is equal to the maximum number of labels that $\mathsf{A}^{\mathcal{H}}$ needs to store at once. Hence, the space bound follows directly from the fact that each label consists of $n_{\mathcal{H}}$ bits and that the algorithm $\mathsf{A}^{\mathcal{H}}$ is s-space bounded. □

Localized expander graphs. A $(N_c, \gamma_1, \gamma_2)$-bipartite expander, for $0 < \gamma_1 < \gamma_2 < 1$, is a DAG with N_c sources and N_c sinks such that any $\gamma_1 N_c$ sinks are connected to at least $\gamma_2 N_c$ sources. We can define a DAG $G'_{N_c,k_G,\gamma_1,\gamma_2}$ by stacking k_G ($\in \mathbb{N}$) bipartite expanders. Informally, stacking means that the sinks of the i-th bipartite expander are the sources of the i+1-st bipartite expander. It is easy to see that such a graph has $N_c(k_G+1)$ nodes which are partitioned into k_G+1 sets (which we call layers) of size N_c. A Stack of Localized Expander Graphs (SoLEG) is a DAG $G_{N_c,k_G,\gamma_1,\gamma_2}$ obtained by applying the transformation called *localization* (see [RD16, Section 3.4] for a definition) on each layer of the graph $G'_{N_c,k_G,\gamma_1,\gamma_2}$.

We restate two lemmas about pebbling complexity of SoLEG from [RD16]. The latter appeared in [RD16] in the form of a discussion.

Lemma 5 ([RD16, Theorem 4])**.** *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a random oracle. Let $G_{N_c,k_G,\gamma_1,\gamma_2}$ be a SoLEG where $\beta := \gamma_2 - 2\gamma_1 > 0$. Let $\mathbf{P} = (P_0, \ldots, P_{t(\mathbf{P})})$ be a pebbling strategy that pebbles at least $\gamma_1 N_c$ output vertices of the graph $G_{N_c,k_G,\gamma_1,\gamma_2}$ which were not initially pebbled, where the initial pebbling configuration is such that $|P_0| \leq \beta N_c$, and the space complexity of $\mathbf{P}$ is bounded by $s(\mathbf{P}) \leq \beta N_c$. Then the time complexity of $\mathbf{P}$ has the following lower bound:*

$$t(\mathbf{P}) \geq 2^{k_G} \gamma_1 N_c.$$

Lemma 6 ([RD16, Section 5.2])**.** *Let $G_{N_c,k_G,\gamma_1,\gamma_2}$ be a SoLEG and $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a random oracle. There exists a polynomial time algorithm $\mathsf{A}^{\mathcal{H}}$ that computes the $\mathcal{H}$-labeling of the graph $G_{N_c,k_G,\gamma_1,\gamma_2}$ in $N_c n_{\mathcal{H}}$-space.*

7.3 Challenge-Hard Graphs (CHG)

In this section we introduce the concept of *challenge-hard graphs* (CHG for short) which we use it to construct proof-extractable NIPoS. Informally challenge-hard graphs satisfy the following property with respect to graph pebbling: with small budget on the number of pebbles available

(for example if an algorithm is space-bounded) no pebbling strategy can put pebbles on *multiple* random challenge vertices of the graph in a reasonable amount of time. The property should hold even if the player gets a few (but not too many) pebbles on arbitrary vertices at the beginning. We remark that the notion of challenge hardness has similarities with a notion introduced in [DFKP15]. In particular, in Section 4 of [DFKP15], the authors informally described a pebbling game which is similar to the game in our notion (Definition 14).

Challenge hard graphs are parameterized by the following variables: $N_c, \beta, N, \tau_c, t, \varepsilon$, where N is the size of the graph; τ_c is the number of challenge nodes, where all the challenges are in a pre-defined target set V_c; N_c is the size of the target set V_c; $\beta \cdot N_c = \Omega(N_c)$ is the budget on the number of pebbles available; t is an upper bound on the running time of pebbling strategies; and ε is an upper bound on the winning probability of the pebbling challenge game.

Definition 14 (Challenge Hard Graphs (CHG))**.** A family of directed acyclic graphs $\{G_\lambda\}_{\lambda\in\mathbb{N}}$ (with constant in-degree)[17] is $(\beta, N_c, N, \tau_c, t, \varepsilon)$-*challenge-hard* (where $\beta \in (0,1)$ is a constant, and other parameters are functions of λ), if for every $\lambda \in \mathbb{N}$ and graph $G = G_\lambda = (V, E)$ (with $N = N(\lambda)$ vertices), there exist τ_c target sets (possibly with overlapping) $V_c^{(1)}, \ldots, V_c^{(\tau_c)} \subseteq V$ such that the union of the target sets

$$V_c := V_c^{(1)} \cup \cdots \cup V_c^{(\tau_c)} \subseteq V$$

has N_c vertices, and the following property is satisfied:

For any pebbling strategy $\mathsf{B} = (\mathsf{B}_1, \mathsf{B}_2)$ it holds that

$$\mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda) := \Pr\left[\mathbf{G}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda) = 1\right] \leq \varepsilon\,,$$

where the pebbling game $\mathbf{G}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda)$ is defined as follows.

Game $\mathbf{G}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda)$:

1. Let $P_0 \leftarrow \mathsf{B}_1$ be a pebbling configuration, where $|P_0| \leq \beta \cdot N_c$.
2. Let $\mathsf{chal} \leftarrow \mathcal{D}^{\tau_c}$ be τ_c random challenge vertices (possibly with overlapping), where $\mathcal{D}^{\tau_c}$ is the uniform distribution over $V_c^1 \times \cdots \times V_c^{(\tau_c)}$.
3. Let $\mathbf{P} = (P_0, \ldots, P_{t(\mathbf{P})}) \leftarrow \mathsf{B}_2(P_0, \mathsf{chal})$ be a pebbling strategy.
4. Output 1 iff
   - $\mathbf{P}$ follows the rule of a sequential pebbling strategy.
   - For every $i \in \{0, \ldots, t(\mathbf{P})\}$, it holds that $|P_i| \leq \beta \cdot N_c$.
   - $\mathsf{chal} \subseteq P_0 \cup \cdots \cup P_{t(\mathbf{P})}$.
   - $t(\mathbf{P}) \leq t$.

We define N_c/τ_c and N/N_c as the **challenge sparseness** and **graph compactness** of G.

Intuitively, challenge sparseness defines what fraction of the target nodes will be challenged. Graph compactness determines what fraction of all node in the graph are in the target set. Looking ahead, these two metrics of CHG will play crucial roles in determining the parameters of the NIPoS and the encoding schemes.

[17] We require the in-degree of the graph to be a constant, because for graph-labeling in the ROM this captures the essence of the standard model. To see this assume that $\mathcal{H}$ is implemented by an iteration-based scheme (e.g., Merkle-Damgård extension), and thereby to compute the hash output, it is sufficient to store only a few labels at each iteration step. However, while in the ROM computing a label $\mathsf{label}(v) := \mathcal{H}(v, \mathsf{label}(\mathsf{pred}(v)))$ is only possible if the entire labeling $\mathsf{label}(\mathsf{pred}(v))$ is stored. If the in-degree is high (e.g. super-constant) this distinction would affect the parameters. We refer to Appendix B.3 in [BCS16] for more discussions.

7.4 Construction of PExt-NIPoS from CHG

Now we present our main PExt-NIPoS construction based on challenge-hard graphs and show that it satisfies proof-extractability. The construction is quite similar to the one presented in [FHMV17a] with only a few minor modifications.

The scheme consists of three algorithms $(\mathsf{Setup}^{\mathcal{H}}, \mathsf{P}^{\mathcal{H}}, \mathsf{V}^{\mathcal{H}})$ that use the following ingredients:

- a DAG $G = (V, E)$ with $N = |V|$ vertices and maximal in-degree $\mathsf{deg} \in O(1)$, which has τ_c target sets [18] $V_c^{(1)}, \ldots, V_c^{(\tau_c)} \subseteq V$ such that

$$V_c := V_c^{(1)} \cup \cdots \cup V_c^{(\tau_c)} \subseteq V$$

  and V_c has N_c vertices.

- a set of random oracles $\{\mathcal{H}_{id}\}_{id \in \{0,1\}^{k_{\mathsf{pos}}}} \cup \{\mathcal{H}_{\mathsf{com}}\} \cup \{\mathcal{H}_{\mathsf{chal}}\}$ defined as follows: $\mathcal{H}_{id} : \{0,1\}^{\leq \log N + \mathsf{deg} \cdot n_{\mathcal{H}}} \to \{0,1\}^{n_{\mathcal{H}}}$ for every $id \in \{0,1\}^{k_{\mathsf{pos}}}$; $\mathcal{H}_{\mathsf{com}} : \{0,1\}^{2n_{\mathcal{H}}} \to \{0,1\}^{n_{\mathcal{H}}}$; $\mathcal{H}_{\mathsf{chal}}$ takes as input a $\{0,1\}^{k_{\mathsf{pos}}+n_{\mathcal{H}}}$-bit string and outputs a random challenge set check plus τ_c challenge nodes:

$$(\mathsf{check}, \mathsf{chal} := (\mathsf{chal}_1, \ldots, \mathsf{chal}_{\tau_c})) \in V^{\tau} \times V_c^{(1)} \times \cdots \times V_c^{(\tau_c)} .$$

  For simplicity of explanation, we assume that the output length of $\mathcal{H}_{\mathsf{chal}}$ is exactly $n_{\mathcal{H}}$ (where $n_{\mathcal{H}} \geq \tau \cdot \log |V| + \tau_c \cdot \log |V_c|$), and we define the corresponding challenge sets $(\mathsf{check}, \mathsf{chal})$ as the first $\tau \cdot \log |V| + \tau_c \cdot \log |V_c|$ bits of the RO output.[19] Note that by a typical domain separation technique (e.g., used in [ABFG14] and [MMV13]), we can instantiate the three random oracles using a unified random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$.

The construction is presented in detail in Figure 3. We provide a high-level overview here. The prover first computes the labeling of a graph $G = (V, E)$, and then commits the labeling using a Merkle tree. From the merkle root value $\tilde{\phi}_\ell$, the prover computes the Fiat-Shamir challenge $\mathcal{H}(\tilde{\phi}_\ell)$, which consists of two sets $(\mathsf{check}, \mathsf{chal})$. The set check contains τ random nodes in V, and the set chal has τ_c random nodes in a target set $V_c \subseteq V$. The proof is the Merkle tree opening paths for nodes in $\mathsf{check} \cup \mathsf{pred}(\mathsf{check}) \cup \mathsf{chal}$, where $\mathsf{pred}(\mathsf{check})$ are the parents of nodes in check.

Remark 7. *Our scheme can be viewed as a generalization of the NIPoS scheme in [FHMV17a]. In particular, besides opening nodes' labels (Merkle-commitments) for checking graph consistency, the proof also includes the labels for a small challenge set* chal. *In particular, setting* chal *empty we can obtain the FHMV NIPoS which in turn is based on the proof-of-space construction from [RD16].*

Memory usage of the prover and the verifier. In our PExt-NIPoS construction, the honest prover has to store the complete labeling of the graph G plus the entire Merkle tree, thus the size of the prover's space is

$$s_{\mathsf{P}} := N \cdot n_{\mathcal{H}} + (N - 1) \cdot n_{\mathcal{H}} ,$$

where $n_{\mathcal{H}}$ is the random oracle output length. On the other hand, the verifier only needs to store a single proof-of-space, which consists of a Merkle root value, two challenge sets, and

[18] Note that the target sets can have overlapping parts, that is they may share some nodes.

[19] For ease of explanation, we assume that $|V|$ and $|V_c|$ are powers of 2.

PExt-NIPoS Construction

$\mathsf{Setup}^{\mathcal{H}}(1^\lambda)$**:** On input the security parameter 1^λ output a set of public parameters $\mathsf{pp}_{\mathsf{pos}} \in \{0,1\}^*$, which consist of values $\tau, \tau_c, N_c, N \in \mathbb{N}$, the DAG G as described above.

$\mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id)$**:** Given public parameters $\mathsf{pp}_{\mathsf{pos}} \in \{0,1\}^*$ and identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$, do as follows:

1. For every node $v \in V$, compute a $\mathcal{H}_{id}$-labeling of v as $\mathsf{label}(v) := \mathcal{H}_{id}(v, \mathsf{label}(\mathsf{pred}(v))$, where $\mathsf{label}(\mathsf{pred}(v))$ are the $\mathcal{H}_{id}$-labelings of v's parents in G. Let $\ell := (\mathsf{label}(v))_{v \in V} \in \{0,1\}^{N \cdot n_{\mathcal{H}}}$ be the $\mathcal{H}_{id}$-labeling of the graph G.
2. Given labeling ℓ compute the Merkle commitment $\phi_\ell := \mathsf{MCom}^{\mathcal{H}_{\mathsf{com}}}(\ell)$, where $\phi_\ell \in \{0,1\}^{n_{\mathcal{H}}}$ is the Merkle root.
3. Determine the set of challenges $(\mathsf{check}, \mathsf{chal}) := \mathcal{H}_{\mathsf{chal}}(id, \phi_\ell)$.
4. Output the proof-of-space π which consists of two parts:
   - The Merkle-root value and the two challenge sets
   $$(\phi_\ell, \mathsf{check}, \mathsf{chal}) \in \{0,1\}^{n_{\mathcal{H}}} \times V^\tau \times V_c^{(1)} \times \cdots \times V_c^{(\tau_c)} .$$
   - Let $\mathsf{pred}(\mathsf{check})$ be the set of predecessors for nodes in check. For every node $v \in \mathsf{check} \cup \mathsf{pred}(\mathsf{check}) \cup \mathsf{chal}$, output the Merkle-tree opening path from the v-th leaf (with label $\mathsf{label}(v)$) to the Merkle-root (with value ϕ_ℓ): $\mathsf{MOpen}^{\mathcal{H}_{\mathsf{com}}}(\ell, v)$.

$\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id, \pi)$**:** Given public parameters $\mathsf{pp}_{\mathsf{pos}}$, identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$ and a *candidate* proof-of-space $\pi \in \{0,1\}^{n_{\mathsf{pos}}}$, check the correctness of π with respec to id as follows:

1. Parse
$$\left[(\phi_\ell, \mathsf{check}, \mathsf{chal}), \left\{(z_v; (y_v^{(1)}, \ldots, y_v^{(\log N)})\right\}_{v \in \mathsf{check} \cup \mathsf{pred}(\mathsf{check}) \cup \mathsf{chal}}\right] := \pi$$
2. Check $(\mathsf{check}, \mathsf{chal}) = \mathcal{H}_{\mathsf{chal}}(id, \phi_\ell)$.
3. For every node $v \in \mathsf{check}$, denote by z_v the opening for v, and $z_{\mathsf{pred}(v)}$ the openings for v's parents in graph G. The check: $z_v = \mathcal{H}_{id}(v, z_{\mathsf{pred}(v)})$
4. For every node $v \in \mathsf{check} \cup \mathsf{pred}(\mathsf{check}) \cup \mathsf{chal}$, denote by $(z_v, (y_v^{(1)}, \ldots, y_v^{(\log N)}))$ the opening path for v, V checks that
$$\mathsf{MVer}^{\mathcal{H}_{\mathsf{com}}}(v, \phi_\ell, z_v, (y_v^{(1)}, \ldots, y_v^{(\log N)})) = 1 .$$
5. Output 1 if and only if all of the above check passes; otherwise output 0.

Figure 3: Our PExt-NIPoS construction: Denoting by ν the number of RO input-output pairs in the proof we call this construction a (τ_c, τ, ν)-**Merkle-tree-based PExt-NIPoS** scheme built upon the DAG G.

$\tau \cdot (\mathsf{deg} + 1) + \tau_c$ tree paths. Since each tree path is of length $\log N$, the size of the verifier's space is given by:

$$s_{\mathsf{V}} := n_{\mathcal{H}} + \tau \cdot \log N + \tau_c \cdot \log N_c + (\tau \cdot (\mathsf{deg} + 1) + \tau_c) \cdot \log N \cdot n_{\mathcal{H}} .$$

It is not hard to see that our PExt-NIPoS scheme satisfies *completeness*. We formally show

that our NIPoS scheme satisfies *proof extractability* by proving the theorem stated next.

Theorem 2. *Let λ be a security parameter. Suppose $G := G_{\text{HARD}}$ is a $(\beta, N_c, N, \tau_c, t, \epsilon_{\mathsf{peb}})$-challenge hard graph with indegree $\mathsf{deg} = O(1)$; $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ is a hash function modeled as a random oracle; and Π_G is a (τ_c, τ, ν)-Merkle-tree-based PExt-NIPoS scheme (defined in Figure 3) built upon G, where*

$$\nu := (\tau \cdot (\mathsf{deg} + 1) + \tau_c) \cdot \log N + 1 \,.$$

For any $s, f \in \mathbb{N}$ such that there exists a constant $\delta^* \in (0,1)$ where

$$s + f \leq (\beta - \delta^* - 0.01) \cdot N_c \cdot n_{\mathcal{H}} \,,$$

it holds that Π_G is a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, f, t, \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof-extractable, as long as[20]

$$s_{\mathsf{P}} \geq k_{\mathsf{pos}} + (2N-1) \cdot n_{\mathcal{H}} \qquad s \geq s_{\mathsf{V}} \geq k_{\mathsf{pos}} + \nu \cdot n_{\mathcal{H}} \qquad \eta = O(\nu \log \lambda)$$
$$n_{\mathsf{pos}} = \nu \cdot n_{\mathcal{H}} \qquad \varepsilon_{\mathsf{p\text{-}ext}} \leq \mathrm{poly}(\lambda) \cdot \left(2^{-n_{\mathcal{H}}} + \exp(-\kappa) + \epsilon_{\mathsf{peb}}\right) \,,$$

where $\kappa = \tau \cdot N_c \cdot \delta^* / N$.

Remark 8. *To guarantee that the verifier space $s_{\mathsf{V}} := \nu \cdot n_{\mathcal{H}}$ is smaller than the tampering space s, we require τ_c to be significantly smaller than $N_c / \log N$. Hence we need the underlying CHG to be* **at least** *$\Omega(\log N)$-challenge sparse (defined as N_c/τ_c).*

Proof intuition. Consider the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$, denote by $(\tilde{id}, \tilde{\pi})$ the adversary's output identity and proof-of-space, and $\tilde{\phi}_\ell$ the Merkle root value in $\tilde{\pi}$. To explain intuitively, we consider the simpler case where $\tilde{\phi}_\ell$ is the commitment of the $\mathcal{H}_{\tilde{id}}$-labeling of the graph.[21] To argue proof extractability, observe that in the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$:

- If the big adversary B computes the graph labeling and the Merkle commitment herself, that is, the RO input-output pairs in $(\tilde{id}, \tilde{\pi})$ are queried by B, then a knowledge extractor can extract $(\tilde{id}, \tilde{\pi})$ from the query table of B.
- Otherwise, we can bound the advantage of the proof extractability game by the advantage of a pebbling challenge game. By challenge-hardness of the graph, the probability that a space-bounded adversary A generates a new proof-of-space in time bound t is negligible.

We provide the formal proof of Theorem 2, that is, our NIPoS scheme satisfies proof extractability in Appendix B.

Remark 9. *Similar to [FHMV17a] and [RD16], we consider a restricted storage model where the adversary always stores the graph labels in their entirety before using them, that is, the adversary never compress the graph labels. We leave the proof of Theorem 2 in the general storage model (where the adversary can store arbitrary function of the graph labels) as an interesting open question.*

[20] The polynomial factor in $\varepsilon_{\mathsf{p\text{-}ext}}$ depends on the number of RO queries made by the adversary. We refer to Inequality 7 for the exact probability upper bound.

[21] The more complicated case, where a labeling with faults is committed, can be handled by requiring the adversary to open a few random nodes and their neighborhoods.

Proof outline. To bound the advantage of proof-extractability game, we first bound the probability of a few bad events. Conditioned on bad events do not happen, it is guaranteed that the space-bounded adversary A wins the game only if it outputs a $(\tilde{id}, \tilde{\pi})$ such that i) the merkle root value $\tilde{\phi}_\ell \in \tilde{\pi}$ is a commitment of a graph labeling that is mostly consistent with the $\mathcal{H}_{\tilde{id}}$-labeling of graph G, and ii) $\tilde{\pi}$ contains the opening labels of a few *random* challenge nodes, that is, the challenge nodes set is unpredictable for A. Hence the winning probability can be naturally bounded by the advantage of a labeling challenge game, which in turn can be in transformed to a pebbling challenge game. And thus we can finally bound the advantage of proof-extractability game by the advantage of pebbling challenge game, which finishes the proof.

7.5 Instantiating CHG

In this section we present two constructions of challenge-hard graphs. First, we propose a new construction of CHG from Stack of Localized Expander Graphs (SoLEG) used by Ren and Devadas [RD16] in the context of proof-of-space. We refer to Section 7.2 for more details on SoLEGs. Then we also show that the graph constructed by Paul, Tarjan and Celoni [PTC76] and used by [DFKP15] satisfies the notion of challenge-hardness, but with different parameters.

First note that a family of SoLEG (with constant in-degree) is already challenge hard if the number of challenge nodes (i.e., τ_c) satisfies that $\tau_c > (\beta + \gamma_1) \cdot N_c$. This follows right away from Lemma 5. However, the challenge sparseness of the graph (i.e., N_c/τ_c) is $O(1)$ which too small for our application.[22] As discussed in Remark 8, we need a challenge-hard graph with $\Omega(\log N)$-challenge sparseness. We resolve this by constructing challenge-hard graphs with larger challenge sparseness by extending a family of SoLEGs. For ease of explanation, in the following we assume that N_c is divisible by τ_c. We note that a similar result holds if N_c/τ_c is not an integer.

Our SoLEG-based Construction. Given $N_c, \tau_c \in \mathbb{N}$ (where N_c/τ_c is an integer), the challenge-hard graph G_{HARD} is a *τ_c-extension of a SoLEG*. It consists of a SoLEG, τ_c gadget graphs, and edges from SoLEG's sink vertices to the gadget graphs.

For every i $(1 \le i \le \tau_c)$, the ith gadget graph $\mathsf{H}^{(i)}_{N_c} = (V^{(i)}, E^{(i)})$ consists of N_c/τ_c lines. For every j $(1 \le j \le N_c/\tau_c)$, the jth line $\mathsf{line}^{(i)}_j$ (in $\mathsf{H}^{(i)}_{N_c}$) consists of N_c/τ_c vertices and a path from the first vertex to the last vertex. For every j $(1 \le j \le N_c/\tau_c)$, we denote by $\mathsf{end}^{(i)}_j$ the sink of $\mathsf{line}^{(i)}_j$, the ith target set $V^{(i)}_c$ is then defined to be

$$V^{(i)}_c := \mathsf{end}^{(i)}_1 \cup \cdots \cup \mathsf{end}^{(i)}_{N_c/\tau_c}.$$

And we see that the union of the target sets

$$V_c := V^{(1)}_c \cup \cdots \cup V^{(\tau_c)}_c$$

contains N_c vertices.

Besides the edges in the SoLEG and gadget graphs, there are also edges from SoLEG's sink vertices to the gadget graphs. In particular, for every i $(1 \le i \le \tau_c)$ and every k $(1 \le k \le N_c/\tau_c)$, the $((i-1)N_c/\tau_c + k)$-th sink vertex of the SoLEG has N_c/τ_c outgoing edges to the gadget graph $\mathsf{H}^{(i)}_{N_c}$, where the jth $(1 \le j \le N_c/\tau_c)$ outgoing edge points to the kth vertex of the jth line $\mathsf{line}^{(i)}_j$.

Let $G_{N_c,k_G,\gamma_1,\gamma_2}$ be the SoLEG, the number of vertices in G_{HARD} is

$$N := k_G \cdot N_c + \tau_c \cdot \left(\frac{N_c}{\tau_c}\right)^2 = k_G \cdot N_c + \frac{N_c^2}{\tau_c}.$$

[22] Recall that β is a constant, thus N_c/τ_c is $O(1)$ if $\tau_c > (\beta + \gamma_1) \cdot N_c$.

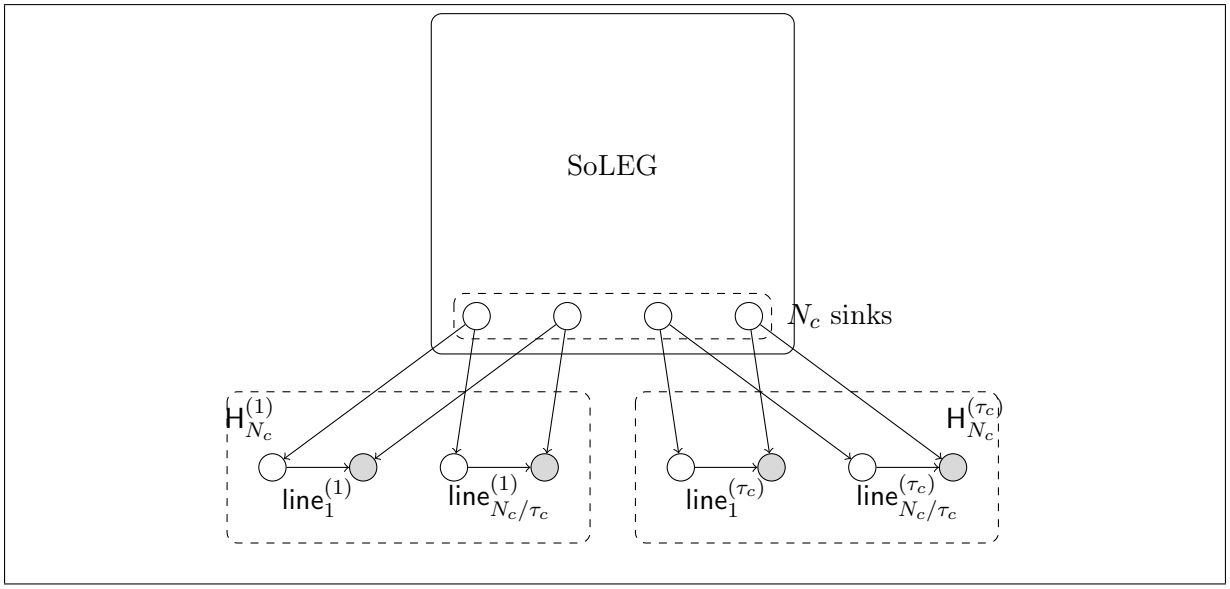

Figure 4: A τ_c-extension of a SoLEG with $\tau_c = 2$ and $N_c = 4$. We ignore the graph structure of SoLEG and only show the sink vertices. Each of the first $N_c/\tau_c = 2$ sink vertices has two outgoing edges, each edge pointing to a line of the first gadget graph $\mathsf{H}_{N_c}^{(1)} := \mathsf{line}_1^{(1)} \cup \mathsf{line}_2^{(1)}$. Similarly, each of the next $N_c/\tau_c = 2$ sink vertices has two outgoing edges, each edge pointing to a line of the second gadget graph $\mathsf{H}_{N_c}^{(2)} := \mathsf{line}_1^{(2)} \cup \mathsf{line}_2^{(2)}$. The left 2 gray nodes consist of the first target set $V_c^{(1)}$. The right 2 gray nodes consist of the second target set $V_c^{(2)}$. The union of the targets has $|V_c| = N_c = 4$ nodes.

We provide a simple example (Figure 4) of a CHG based on a SoLEG below.

It is easy to observe that the challenge sparseness is N_c/τ_c and the graph compactness of the graph is $k_G + N_c/\tau_c$ for the above construction. We prove the following lemma about our construction.

Lemma 7. *Let $G_{N_c,k_G,\gamma_1,\gamma_2}$ be a SoLEG with parameters $N_c, k_G \in \mathbb{N}$, $\gamma_1, \gamma_2 \in (0,1)$. Denote by $\beta := \gamma_2 - 2\gamma_1 > 0$ and $\varepsilon := 1 - \beta - \gamma_1 > 0$. For any $\tau_c \in \mathbb{N}$ (such that N_c/τ_c is an integer), let $G_{\text{HARD}} := (V, E)$ be the τ_c-extension of $G_{N_c,k_G,\gamma_1,\gamma_2}$. Then it holds that G_{HARD} is $(\beta, N_c, N := k_G \cdot N_c + N_c^2/\tau_c, \tau_c, 2^{k_G} \cdot \gamma_1 \cdot N_c, \exp(-\varepsilon^2 \cdot \tau_c))$-challenge hard.*

Proof. Fix any adversary $\mathsf{B} = (\mathsf{B}_1, \mathsf{B}_2)$ and consider the pebbling challenge game $\mathbf{G}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda)$. Denote by $P_0 \subseteq V$ the initial pebbled set output by B_1. Denote by $n_0 := \delta_0 \cdot N_c$ the number of output vertices (in $G_{N_c,k_G,\gamma_1,\gamma_2}$) that were initially pebbled. For every i ($1 \le i \le \tau_c$), recall that the target subset $V_c^{(i)}$ consists of N_c/τ_c lines, and we denote by $n_i := \delta_i \cdot N_c/\tau_c$ the number of lines that have at least one pebble on it (i.e., the lines whose intersection with P_0 are non-empty). Since $|P_0| \le \beta \cdot N_c$, by definition of the game, we have

$$|P_0| \le \sum_{i=0}^{\tau_c} n_i = \delta_0 \cdot N_c + \sum_{i=1}^{\tau_c} \delta_i \cdot \frac{N_c}{\tau_c} \le \beta \cdot N_c \,.$$

By rewriting the second inequality, we have

$$\sum_{i=1}^{\tau_c} \delta_i \cdot \frac{N_c}{\tau_c} \le (\beta - \delta_0) \cdot N_c \,. \tag{1}$$

For every i ($1 \le i \le \tau_c$), we define a random variable $X_i \in \{0,1\}$ for the ith challenge node: Denote by $\mathsf{chal}_i \subseteq V_c^{(i)}$ the challenge node in the ith target set $V_c^{(i)}$; we set $X_i := 1$ if the line that contains chal_i has no initially pebbled vertex (i.e., the line has no intersection with P_0); and we set $X_i := 0$ otherwise. We define

$$X := \sum_{i=1}^{\tau_c} X_i \,.$$

Note $\Pr[X_i = 1] = 1 - \delta_i$ (over the randomness of chal_i), and random variables $\{X_i\}_{i\in[\tau_c]}$ are independent. Recall that $\varepsilon := 1 - \beta - \gamma_1 > 0$, thus by Hoeffding inequality, we have

$$\Pr\left[X \le \mathbb{E}\left[X\right] - \varepsilon \cdot \tau_c\right] \le \exp(-2 \cdot \varepsilon^2 \cdot \tau_c) \,.$$

Next we show that if $X \ge \mathbb{E}\left[X\right] - \varepsilon \cdot \tau_c$, then it takes at least $t := 2^{k_G} \cdot \gamma_1 \cdot N_c$ steps to pebble all the challenge vertices, and hence finish the proof.

Claim 1. *Denote by $\varepsilon := 1 - \beta - \gamma_1 > 0$. If the challenge set chal is chosen so that*

$$X \ge \mathbb{E}\left[X\right] - \varepsilon \cdot \tau_c = \sum_{i=1}^{\tau_c}(1-\delta_i) - (1-\beta-\gamma_1)\cdot\tau_c = (\beta+\gamma_1)\cdot\tau_c - \sum_{i=1}^{\tau_c}\delta_i \,,$$

then it takes the adversary B_2 at least $t := 2^{k_G} \cdot \gamma_1 \cdot N_c$ steps to pebble all the challenge vertices.

Proof. First, we show that the adversary B_2 has to pebble at least $X \cdot N_c/\tau_c$ output vertices (in $G_{N_c,k_G,\gamma_1,\gamma_2}$) to answer all challenges: For every i ($1 \le i \le \tau_c$), if $X_i = 1$, then the line that contains chal_i has no initially pebbled vertex. Thus to pebble the sink node chal_i, the adversary has to pebble the vertices (on the line) step by step, which in turn requires the pebbling of the output vertices $[(i-1)\cdot N_c/\tau_c + 1, \ldots, i \cdot N_c/\tau_c]$ of $G_{N_c,k_G,\gamma_1,\gamma_2}$. Since $X = \sum_{i=1}^{\tau_c} X_i$, the adversary has to pebble at least $X \cdot N_c/\tau_c$ output vertices in total.

On the other hand, if the challenge set chal is chosen so that $X \ge (\beta+\gamma_1)\cdot\tau_c - \sum_{i=1}^{\tau_c}\delta_i$, then by Inequality 1, we have

$$X \cdot \frac{N_c}{\tau_c} \ge (\beta+\gamma_1)\cdot N_c - \sum_{i=1}^{\tau_c}\delta_i \cdot \frac{N_c}{\tau_c} \ge (\beta+\gamma_1)\cdot N_c - (\beta-\delta_0)\cdot N_c = (\gamma_1+\delta_0)\cdot N_c \,.$$

Since at most $\delta_0 \cdot N_c$ output vertices were pebbled initially, B_2 has pebbled at least $\gamma_1 \cdot N_c$ output vertices that were not initially pebbled. By Lemma 5, we have the time lower bound $t \ge 2^{k_G} \cdot \gamma_1 \cdot N_c$. □

Combining the above arguments, with probability at least $1 - \exp(-2\cdot\varepsilon^2\cdot\tau_c)$ (over the choice of the challenge set), it takes at least $t := 2^{k_G} \cdot \gamma_1 \cdot N_c$ steps to pebble all the challenge vertices, and thus the graph G_{HARD} is $(\beta, N_c, N, \tau_c, 2^{k_G} \cdot \gamma_1 \cdot N_c, \exp(-\varepsilon^2 \cdot \tau_c))$-challenge hard. □

Construction of CHG from PTC [PTC76]'s graphs As observed by [DFKP15] (in Section 6.1 of [DFKP15]), the graph introduced by Paul, Tarjan and Celoni [PTC76] (in short, PTC's graph) does satisfy challenge hardness. Adapting Theorem 1 of [DFKP15] into the multiple challenge setting, we obtain the following lemma.

Lemma 8 ([DFKP15, Theorem 1])**.** *Let $\{G\}_{\lambda\in\mathbb{N}}$ be a family of PTC's graph (DAG) with graph parameters $N_c = N_c(\lambda)$, $N := N_c \log N_c$ and in-degree 2. Then for any $\tau_c \in \mathbb{N}$ and any $\lambda \in \mathbb{N}$, G_λ is $(\beta, N_c, N, \tau_c, \infty, (1-\beta)^{\tau_c})$-challenge-hard where $\beta = 1/512$.*

7.6 A comparison of the two CHG constructions

We provide a brief comparison between the two CHG constructions described above. First we provide asymptotic bounds on the parameters. Setting $N_c = \lambda^2$, $k_G = \lambda^{0.5}$, and $\tau_c = \Theta(\lambda^{1.5})$ in Lemma 7, we observe that the CHG based on SoLEGs has $\Omega(\lambda^{0.5})$-challenge sparseness, $O(\lambda^{0.5})$-graph compactness and exponential security.[23] Formally,

Corollary 2 (CHG based on SoLEGs)**.** *Let $G_{N_c,k_G,\gamma_1,\gamma_2}$ be a SoLEG with parameters $N_c = \lambda^2, k_G = \Theta(\lambda^{0.5})$, $\gamma_1, \gamma_2 = O(1)$. The $\Theta(\lambda^{1.5})$-extension of $G_{N_c,k_G,\gamma_1,\gamma_2}$ is $(O(1), \lambda^2, O(\lambda^{2.5}), \Theta(\lambda^{1.5}), 2^{\Theta(\lambda^{0.5})}, 2^{-\omega(\lambda)})$-challenge hard. The challenge sparseness is $\Omega(\lambda^{0.5})$, and the graph compactness is $O(\lambda^{0.5})$.*

Under the same level of security, the CHG based on [PTC76]'s graphs achieves better graph compactness *asymptotically*. In particular, setting $N_c = \lambda^2$, $\tau_c = \Theta(\lambda^{1.5})$ in Lemma 8 we observe that the CHG based on PTC's graphs has $\Omega(\lambda^{0.5})$-challenge sparseness, $O(\log \lambda)$-graph compactness and exponential security. Formally,

Corollary 3 (CHG based on PTC's graphs)**.** *Let G_λ be a PTC's graph with graph parameters $N_c = N_c(\lambda)$, $N := N_c \log N_c$ and in-degree 2. Then it satisfies $(O(1), \lambda^2, O(\lambda^2 \log \lambda), \Theta(\lambda^{1.5}), \infty, 2^{-\omega(\lambda)})$-challenge hardness. The challenge sparseness is $\Omega(\lambda^{0.5})$, and the graph compactness is $O(\log \lambda)$.*

It is worth noting that, asymptotically, PTC's graph supports better parameters than our SoLEG-extension graphs for the same security. However due to the presence of a large constant factor (that is $1/\beta = 512$), the graph-size of PTC's graph becomes much larger than the SoLEG-extension graph— this affects the concrete parameters of NIPoS significantly (see, Corollaries 8 and 9).

By setting $\varepsilon = 2^{-80}$, the pebbling budget $\beta \cdot N_c := 2^{10}$, and time bound to be 2^{80}, we calculate the concrete values of the other parameters. We present two corollaries below.

Corollary 4 (CHG based on SoLEGs (concrete))**.** *For the SoLEG-extension construction, setting $\gamma_2 := 2/3$, $\gamma_1 := 1/6$, we obtain $\beta = 1/3$ and $\varepsilon = 1/2$. To obtain 80-bit security in the pebbling game (i.e., $\mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda) \leq 2^{-80}$), the challenge parameter τ_c can be approximated to 2^8. For the pebbling budget bound $\beta \cdot N_c := 2^{10}$ and a typical time bound $t := 2^{80}$, the target set size N_c is no more than 2^{12}, and the graph size N is about 2^{18}.*

Corollary 5 (CHG based on PTC's graphs (concrete))**.** *For the PTC's graph, for a typical 80-bit security of the pebbling game (i.e., $\mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B},\beta,t,\tau_c,G}(\lambda) \leq 2^{-80}$), the challenge parameter τ_c can be approximated to $\approx 2^{15}$. For a typical pebbling budget $\beta \cdot N_c := 2^{10}$, we have to set the target set size N_c to be 2^{19}, and the graph size should be at least $N := 2^{23}$.*

7.7 Instantiations of PExt-NIPoS from CHGs

We obtain two PExt-NIPoS constructions by plugging-in the parameters from two CHG constructions, namely the SoLEG-extension (Corollary 2) and the PTC's graph (Corollary 3) respectively into Theorem 2.

Corollary 6 (PExt-NIPoS from SoLEG-extension)**.** *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_\mathcal{H}}$ be a hash function modeled as a random oracle (where $n_\mathcal{H} = \omega(\lambda)$). Let G_{HARD} be the τ_c-extension of*

[23] Note that choosing $\tau_c = \Theta(\lambda)$ is already sufficient for exponential security. We choose τ_c to be $\Theta(\lambda^{1.5})$ only for shrinking the graph compactness $N/N_c = k_G + N_c/\tau_c$ which is an important factor in constructing NIPoS.

a SoLEG with graph parameter $N_c = \lambda^2$. The (τ_c, τ, ν)-Merkle-tree-based PExt-NIPoS scheme built upon G_{HARD}, where

$$\tau_c = \omega(\lambda^{1.5}) \qquad \tau = \omega(\lambda^{1.5}) \qquad \nu = \omega(\lambda^{1.5} \log \lambda)\,,$$

is a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, f, t, \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof extractable, where

$$k_{\mathsf{pos}} = O(\lambda^{1.5} \log \lambda) \cdot n_{\mathcal{H}} \qquad n_{\mathsf{pos}}, s_{\mathsf{V}} = \omega(\lambda^{1.5} \log \lambda) \cdot n_{\mathcal{H}} \qquad s_{\mathsf{P}} = O(\lambda^{2.5}) \cdot n_{\mathcal{H}}$$
$$s, f = \Theta(\lambda^2) \cdot n_{\mathcal{H}} \qquad t = \Omega(2^{\sqrt{\lambda}}) \qquad \varepsilon_{\mathsf{p\text{-}ext}} = 2^{-\omega(\lambda)} \qquad \eta = \omega(\lambda^{1.5} \log^2 \lambda)\,.$$

Corollary 7 (PExt-NIPoS from PTC's graph)**.** *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a hash function modeled as a random oracle (where $n_{\mathcal{H}} = \omega(\lambda)$). Let G_{HARD} be the PTC's graph [PTC76] with graph parameter $N_c = \lambda^2$. The (τ_c, τ, ν)-Merkle-tree-based PExt-NIPoS scheme built upon G_{HARD}, where*

$$\tau_c = \omega(\lambda) \qquad \tau = \omega(\lambda \log \lambda) \qquad \nu = \omega(\lambda \log^2 \lambda)\,,$$

is a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, f, t, \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof extractable, where

$$k_{\mathsf{pos}} = O(\lambda \log^2 \lambda) \cdot n_{\mathcal{H}} \qquad n_{\mathsf{pos}}, s_{\mathsf{V}} = \omega(\lambda \log^2 \lambda) \cdot n_{\mathcal{H}} \qquad s_{\mathsf{P}} = O(\lambda^2 \log \lambda) \cdot n_{\mathcal{H}}$$
$$s, f = \Theta(\lambda^2) \cdot n_{\mathcal{H}} \qquad t = \infty \qquad \varepsilon_{\mathsf{p\text{-}ext}} = 2^{-\omega(\lambda)} \qquad \eta = \omega(\lambda \log^3(\lambda))\,.$$

We complete the comparison of the two PExt-NIPoS constructions based on CHG, we instantiate the above corollaries with concrete parameters.

Corollary 8 (PExt-NIPoS from SoLEG-ext.(concrete))**.** *Let us fix:*[24]

$$|\mathcal{Q}_{\mathcal{H}}(\mathsf{B})| \le 2^{64} \qquad |\mathcal{Q}_{\mathcal{H}}(\mathsf{A})| \le 2^{64} \qquad n_{\mathcal{H}} = 2^{16} \qquad t = 2^{80} \qquad \varepsilon_{\mathsf{p\text{-}ext}} = 2^{-160}\,,$$

and consider the NIPoS built upon SoLEG-based CHGs. To guarantee that $s_{\mathsf{V}} \le s$, the graph size of the SoLEG-based CHG is at least $N \gtrapprox 100,000,000$. The resulting NIPoS is a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, f, t, \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof extractable, where

$$k_{\mathsf{pos}} \approx 1\text{MB} \qquad n_{\mathsf{pos}}, s_{\mathsf{V}} \approx 800\text{MB} \qquad s_{\mathsf{P}} \approx 1\text{TB} \qquad s + f \approx 1.1\text{GB} \qquad \eta \approx 750\text{KB}\,.$$

Corollary 9 (PExt-NIPoS from PTC (concrete))**.** *Let us fix:*

$$|\mathcal{Q}_{\mathcal{H}}(\mathsf{B})| \le 2^{64} \qquad |\mathcal{Q}_{\mathcal{H}}(\mathsf{A})| \le 2^{64} \qquad n_{\mathcal{H}} = 2^{16} \qquad t = 2^{80} \qquad \varepsilon_{\mathsf{p\text{-}ext}} = 2^{-160}\,,$$

and consider the NIPoS built upon CHGs in [PTC76]. To guarantee that $s_{\mathsf{V}} \le s$, the graph size of the CHG is at least $N \gtrapprox 300,000,000,000$. The resulting NIPoS is a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS that is $(s, f, t, \eta, \varepsilon_{\mathsf{p\text{-}ext}})$-proof extractable, where

$$k_{\mathsf{pos}} \approx 256\text{MB} \qquad n_{\mathsf{pos}}, s_{\mathsf{V}} \approx 256\text{GB} \qquad s_{\mathsf{P}} \approx 2.5\text{PB} \qquad s + f \approx 256\text{GB} \qquad \eta \approx 250\text{MB}\,.$$

Remark 10. *Observe that concrete parameters of the above constructions turn out to be better than the asymptotics, due to the presence of a "large" constant factor ($1/\beta = 512$) in PTC's graph. This phenomenon is an effect of the same phenomenon observed in the CHG constructions (c.f. Section 7.6).*

[24] We set the bound for $\varepsilon_{\mathsf{p\text{-}ext}}$ to be 2^{-160} because later in Section 9.1, for instantiating the SP-NMC-SD with the desired security.

8 PExt-NIPoS from Memory-Hard Functions

In this section we propose a simple construction of NIPoS with extractability. Our construction is based on memory-hard functions (MHF for short) and verifiable computations. Intuitively, an MHF requires that any time-bounded algorithm needs to use significant memory to compute such function on a randomly chosen input. We remark that, in contrast to the previous works [AS15, ACK+16, ACP+17], we treat the hash functions inside the MHF as concrete hash functions and *not* modeled as random oracles. In particular, we require that an MHF is concretely described and can be represented as a circuit. This is crucial in our context as we use verifiable computation which requires non-black-box access to the function (i.e. an MHF) to be verified. Indeed, we pay a price for that: unlike [AS15, ACK+16, ACP+17] we do not have a provable guarantee for the memory-hardness, because the only known way to have such guarantee is to use ROM (that gives provable guarantees based on pebbling-games) which is not compatible with our setting. Instead, we heuristically assume that a hash-based MHF construction, that has provable memory-hardness guarantee in the ROM, is memory-hard when the random oracle is instantiated with a standard hash function (for example SHA3).

8.1 Memory-hard Functions

Here we formalize memory-hard functions.

Definition 15 (Memory-hard Functions (MHF))**.** Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a random oracle. For parameters $k, n, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t \in \mathbb{N}$ and $\varepsilon_{\mathsf{mhf}} \in [0,1)$, where $s_{\mathsf{mhf}} \geq s$, a function $M : \{0,1\}^k \to \{0,1\}^n$ is called a $(k, n, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t, \varepsilon_{\mathsf{mhf}})$-memory-hard function (or MHF for short) in the ROM if:

- M is computable by a $(s_{\mathsf{mhf}}, t_{\mathsf{mhf}})$-space-time-bounded algorithm.
- for any (s, f, t)-bounded deterministic algorithm $\mathsf{A}^{\mathcal{H}}$, any $x \in \{0,1\}^*$ we have that:

$$\Pr_{\mathcal{H}}[M(\mathcal{H}(x)) = \mathsf{A}^{\mathcal{H}}(x)] \leq \varepsilon_{\mathsf{mhf}}$$

Remark 11. *It is worth noting that, though our definition is in the ROM, the function M itself does not have access to random oracles, but in the security game the adversary A has access to the random oracle. Looking ahead, since our notion of NIPoS is also in random oracle model, we stick to random oracle model here. Nevertheless our definition can be generalized to standard model straightforwardly.*

8.2 Publicly verifiable computation

We describe publicly verifiable computation below. Our definition follows prior work in verifiable computation literature (e.g. [PHGR13]), but adjusted to our setting. In particular, we will be explicit about space and time use of each algorithm and the algorithms involved will follow the notion of bounded algorithms (cf. Definition 3).

Definition 16 (Publicly verifiable computation)**.** For parameters $s_F, t_F, s^{vc}_{\mathsf{P}}, t^{vc}_{\mathsf{P}}, s^{vc}_{\mathsf{V}}, \mathsf{t}^{vc}_{\mathsf{V}}, k, n, n_{\mathsf{vc}} \in \mathbb{N}$, with $s^{vc}_{\mathsf{V}} < s_F \leq s^{vc}_{\mathsf{P}}$, $\mathsf{t}^{vc}_{\mathsf{V}} < t_F \leq t^{vc}_{\mathsf{P}}$ let $F : \{0,1\}^k \to \{0,1\}^n$ be a deterministic function that is computable by an (s_F, t_F)-space-time bounded algorithm. Then an $(s_F, t_F, s^{vc}_{\mathsf{P}}, t^{vc}_{\mathsf{P}}, s^{vc}_{\mathsf{V}}, \mathsf{t}^{vc}_{\mathsf{V}}, k, n, n_{\mathsf{vc}}, \varepsilon_{\mathsf{vc}})$-non-interactive publicly verifiable computation (or VC for short) for F, where $\varepsilon_{\mathsf{vc}} \in [0,1)$, consists of a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ with the following syntax.

- $\mathsf{Gen}_F(1^\lambda) \to (ek_F, vk_F)$: This randomized algorithm, hardwired with the description of F, takes as input the security parameter and outputs a public verification key vk_F and a public evaluation key ek_F.
- $\mathsf{Prove}_{ek_F}(x) \to (y, \pi_{\mathsf{vc}})$: This is a $(s^{vc}_{\mathsf{P}}, t^{vc}_{\mathsf{P}})$-space-time bounded randomized algorithm with the public evaluation key ek_F hard-coded which takes an input $x \in \{0,1\}^k$ and returns the value $y = F(x) \in \{0,1\}^n$ and a proof of computation $\pi_{\mathsf{vc}} \in \{0,1\}^{n_{\mathsf{vc}}}$.
- $\mathsf{Ver}_{vk_F}(x, y, \pi_{\mathsf{vc}}) =: 1/0$: This is a $(s^{vc}_{\mathsf{V}}, \mathsf{t}^{vc}_{\mathsf{V}})$-space-time bounded deterministic algorithm that takes as input the public verification key vk_F, an input $x \in \{0,1\}^k$, an output $y \in \{0,1\}^n$ and a candidate proof $\pi_{\mathsf{vc}} \in \{0,1\}^{n_{\mathsf{vc}}}$ and returns a decision bit.

We require the following properties to hold for any security parameter $\lambda \in \mathbb{N}$:

Completeness: for all $x \in \{0,1\}^k$ we have that:

$$\Pr\left[\mathsf{Ver}_{ek_F}(x, y, \pi_{\mathsf{vc}}) = 1 \;\middle|\; (ek_F, vk_F) \leftarrow \mathsf{Gen}_F(1^\lambda); (y, \pi_{\mathsf{vc}}) \leftarrow \mathsf{Prove}_{ek_F}(x)\right] = 1,$$

where the probability is over the internal random coins of the algorithms Gen and Prove.

Soundness: for all probabilistic polynomial time adversaries A, we have that

$$\Pr\left[\begin{matrix}\mathsf{Ver}_{vk_F}(x^*, y^*, \pi^*_{\mathsf{vc}}) = 1 \\ F(x^*) \neq y^*\end{matrix} \;\middle|\; \begin{matrix}(ek_F, vk_F) \leftarrow \mathsf{Gen}_F(1^\lambda); \\ (x^*, y^*, \pi^*_{\mathsf{vc}}) \leftarrow \mathsf{A}(ek_F, vk_F)\end{matrix}\right] \leq \varepsilon_{\mathsf{vc}},$$

where the probability is over the internal random coins of the algorithms Gen and A.

8.3 Partially-unique Ext-NIPoS from MHF and VC

In this section, we construct a partially-unique NIPoS with extractability based on a MHF and a VC with space-bounded verification. At a high level, the NIPoS scheme is designed as follows. Let M be a memory-hard function and $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ a publicly verifiable scheme. The NIPoS prover on input id first queries the random oracle to obtain $x := \mathcal{H}(id)$ and then runs the algorithm Prove on input x and outputs whatever the algorithm outputs, i.e. the value $y := M(x)$ and the proof of correct computation π_{vc}. The NIPoS verifier on input id and the proof of space (y, π_{vc}) first queries the random oracle to obtain $x := \mathcal{H}(id)$ and the runs the algorithm Ver on input x, y, π_{vc} and outputs whatever the algorithm outputs.

Our Construction. Let M be a $(k, n, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t, \varepsilon_{\mathsf{mhf}})$-MHF, $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ be a $(s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s^{vc}_{\mathsf{P}}, t^{vc}_{\mathsf{P}}, s^{vc}_{\mathsf{V}}, \mathsf{t}^{vc}_{\mathsf{V}}, k, n, n_{\mathsf{vc}}, \varepsilon_{\mathsf{vc}})$-VC scheme for M and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a hash-function modeled as random oracle such that $t_{\mathsf{mhf}}, t^{vc}_{\mathsf{P}}, \mathsf{t}^{vc}_{\mathsf{V}} \in \mathrm{poly}(\lambda)$ and $s^{vc}_{\mathsf{V}} \leq s < s^{vc}_{\mathsf{P}}$. Then define the following algorithms:

$\mathsf{Setup}(1^\lambda)$**:** On input the security parameter, run $(vk_M, ek_M) \leftarrow \mathsf{Gen}_M(1^\lambda)$ and set $\mathsf{pp}_{\mathsf{pos}} := (vk_M, ek_M)$.

$\mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id)$**:** Given public parameters $\mathsf{pp}_{\mathsf{pos}} := (vk_M, ek_M)$ and an identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$, compute the proof-of-space as follows:

1. Obtain $x := \mathcal{H}(id)$ by querying $\mathcal{H}$.
2. Compute $(y, \pi_{\mathsf{vc}}) := \mathsf{Prove}_{ek_M}(x)$.
3. Return $\pi := (y, \pi_{\mathsf{vc}})$.

$\mathsf{V}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id, \pi)$**:** Given public parameters $\mathsf{pp}_{\mathsf{pos}} := (vk_M, ek_M)$ an identity $id \in \{0,1\}^{k_{\mathsf{pos}}}$ and a candidate proof $\pi \in \{0,1\}^{n_{\mathsf{pos}}}$, check the correctness of π with respect to id as follows:

1. Obtain $x := \mathcal{H}(id)$ by querying $\mathcal{H}$.
2. Parse $(y, \pi_{\mathsf{vc}}) := \pi$.
3. Return $\mathsf{Ver}_{vk_M}(x, y, \pi_{\mathsf{vc}})$.

In the following lemma, we formally state that the above construction satisfies uniqueness and extractability properties.

Lemma 9. *The above construction is* $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$*-NIPoS with* $(u_{\mathsf{pos}}, \varepsilon_{\mathsf{unique}})$*-uniqueness and* $(s, f, t, \eta, \varepsilon_{\mathsf{pos}})$*-extractability as long as:*

$$k_{\mathsf{pos}} \in \mathrm{poly}(\lambda) \qquad n_{\mathsf{pos}} = n + n_{\mathsf{vc}} \qquad s_{\mathsf{P}} \geq \max(s^{vc}_{\mathsf{P}}, k_{\mathsf{pos}})$$
$$s_{\mathsf{V}} \leq s^{vc}_{\mathsf{V}} + k + n + k_{\mathsf{pos}} + n_{\mathsf{vc}} \qquad \eta = \log |\mathcal{Q}_{\mathcal{H}}(\mathsf{B})|$$
$$u_{\mathsf{pos}} = n \qquad \varepsilon_{\mathsf{unique}} \leq \varepsilon_{\mathsf{vc}} \qquad \varepsilon_{\mathsf{pos}} \leq \varepsilon_{\mathsf{vc}} + \varepsilon_{\mathsf{mhf}} + \frac{1}{2^k - |\mathcal{Q}_{\mathcal{H}}(\mathsf{B})|}$$

where $|Q_{\mathcal{H}}(\mathsf{B})|$ *is the the total number of random-oracle query made by* B*.*

Proof. The bounds on $n_{\mathsf{pos}}, u_{\mathsf{pos}}$ are obvious. The parameter k_{pos} can be set to any polynomial in λ as the random oracle's input domain is unrestricted. If k_{pos} is smaller than s^{vc}_{P}, then s_{P} can be set to s^{vc}_{P}, otherwise it must be set to at least the length of the id (as it needs to store the entire input to make an RO query). The upper bound on s_{V} is obtained just by summing up lengths of all the parameters used with the space required to run the verifier. The uniqueness error $\varepsilon_{\mathsf{unique}}$ can be set up to $\varepsilon_{\mathsf{vc}}$ because as long as soundness of the VC-scheme does not fail uniqueness holds. It only remains to argue about $\varepsilon_{\mathsf{pos}}$ which we do next.

Let us first give some intuition. Since the no (s, f, t)-bounded adversary can compute the function M on a random input $\mathcal{H}(id)$, B's best bet to make the output of the $\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda) = 1$ is to either (i) compute another function, that is not memory-hard and "fake" the proof or (ii) guess the random-oracle output $\mathcal{H}(id)$ correctly. The probabilities are bounded by $\varepsilon_{\mathsf{vc}}$ and $\approx 1/2^k$ respectively for those events and hence the probability of $\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda)$ outputting 1 can be bounded by $\approx \varepsilon_{\mathsf{vc}} + 2^{-k}$. We formalize $\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda)$ (Definition 10) with respect to the above construction:

Game $\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda)$

1. Sample $\mathsf{pp}_{\mathsf{pos}} \leftarrow \mathsf{Setup}(1^\lambda)$ where $\mathsf{pp}_{\mathsf{pos}} = (ek_M, vk_M)$.
2. Compute $x = \mathcal{H}(id)$ and run $(y, \pi_{\mathsf{vc}}) \leftarrow \mathsf{Prove}_{ek_M}(x)$.
3. Let $\mathsf{A} \leftarrow \mathsf{B}^{\mathcal{H}}(\mathsf{pp}_{\mathsf{pos}}, id, (y, \pi_{\mathsf{vc}}))$.
4. Let $(\tilde{id}, \tilde{y}, \tilde{\pi}_{\mathsf{vc}}) := \mathsf{A}^{\mathcal{H}}(id, y, \pi_{\mathsf{vc}})$.
5. Let $z := F_{\mathsf{hint}}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), \tilde{id})$.
6. Let $\alpha := \mathsf{K}(\mathsf{pp}_{\mathsf{pos}}, \mathcal{Q}_{\mathcal{H}}(\mathsf{B}), z)$.
7. Compute $\tilde{x} := \mathcal{H}(\tilde{id})$ and output 1 if an only if (i)$\mathsf{Ver}_{vk_M}(\tilde{x}, \tilde{y}, \tilde{\pi}_{\mathsf{vc}}) = 1$, (ii) $id \neq \tilde{id}$ and (iii) $id \neq \alpha$. Otherwise output 0.

where the hint-producer just leaks the index of $\tilde{id}$ from the table, and if $\tilde{id}$ is not found, then it returns 0^η. The extractor, on receiving z would return id from the table $\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$, unless $z = 0^\eta$, in which case it returns $0 \ldots 0$. Clearly we have that: $\eta = \log |\mathcal{Q}_{\mathcal{H}}(\mathsf{B})|$.

For a fixed $id \in \{0,1\}^{k_{\mathsf{pos}}}$ define the following events with respect to the above game where randomness comes from the randomized procedures and B:

Event UNSOUND: $\tilde{y} \neq M(\tilde{x})$ and $\mathsf{Ver}_{vk_M}(\tilde{x}, \tilde{y}, \tilde{\pi}_{\mathsf{vc}}) = 1$

Event GUESS: B guesses $\tilde{x}$ correctly without asking the RO on $\tilde{id}$.

Now we have that:

$$\Pr[\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda) = 1] \leq \Pr[\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda) = 1 \mid \neg\text{UNSOUND} \wedge \neg\text{GUESS}] + \Pr[\text{UNSOUND}] + \Pr[\text{GUESS}].$$

However since M is a $(k, n, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t, \varepsilon_{\mathsf{mhf}})$-MHF we have:

$$\Pr[\mathbf{G}^{\mathsf{ext}}_{\mathsf{B},id}(\lambda) = 1 \mid \neg\text{UNSOUND} \wedge \neg\text{GUESS}] \leq \varepsilon_{\mathsf{mhf}}.$$

Furthermore from the soundness of VC we have that $\Pr[\text{UNSOUND}] \leq \varepsilon_{\mathsf{vc}}$ and by a simple counting argument, we get $\Pr[\text{GUESS}] \leq \frac{1}{2^k - |\mathcal{Q}_{\mathcal{H}}(\mathsf{B})|}$ where $|\mathcal{Q}_{\mathcal{H}}(\mathsf{B})|$ is the total number of random oracle queries asked by B. Combining the above results, we obtain that

$$\varepsilon_{\mathsf{pos}} \leq \varepsilon_{\mathsf{vc}} + \varepsilon_{\mathsf{mhf}} + \frac{1}{2^k - |\mathcal{Q}_{\mathcal{H}}(\mathsf{B})|}$$

which concludes the proof. □

8.4 Instantiating MHF

Our MHF instantiation is a slight variant of a graph-based proof of space construction;[25] in particular, we choose the one provided in [RD16] (also used in [FHMV17a]). However, similar formal arguments of space-hardness does not work in our case. Instead, we rely on a heuristic assumption (and also Assumption 1) that our construction, provided below, satisfies our definition of MHF (cf. Definition 15) for useful parameters.

Our construction $M_{G,\mathsf{Hash}}$: On input $x \in \{0,1\}^k$, define the MHF $M_{G,\mathsf{Hash}}$ as follows: consider the SoLEG $G_{N_c,k_G,\gamma_1,\gamma_2}$; recall that the number of nodes of $G_{N_c,k_G,\gamma_1,\gamma_2}$ is given by $N = N_c(k_G + 1)$ and the in-degree is $\deg \in O(1)$. Let $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{n_{\mathsf{hs}}}$ be a standard hash function (for example SHA3) with collision-probability $\varepsilon_{\mathsf{hs}}$. On input $x \in \{0,1\}^k$, first compute a Hash_x-labeling of $G_{N_c,k_G,\gamma_1,\gamma_2}$. Denote the labeling by $\mathbf{z} = (z_1, \ldots, z_N) \in \{0,1\}^{n_{\mathsf{hs}}N}$, where each $z_i \in \{0,1\}^{n_{\mathsf{hs}}}$. Output y where $y := \mathcal{H}_x(\mathbf{z}) \in \{0,1\}^{n_{\mathsf{hs}}}$.

For a standard instantiation of $\mathcal{H}$, we assume the following facts about labeling a SoLEG. For basic definitions and facts about graph labeling we refer to Section 7.2. Our first assumption is essentially a concrete version of Lemma 6, in that the hash function is modeled as a random oracle. Instantiating that with a concrete hash function, we loose that provable guarantee. Nevertheless, we assume that the same property holds.

Assumption 1 (Efficient labeling with Hash)**.** Let $G_{N_c,k_G,\gamma_1,\gamma_2}$ be a SoLEG and $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ be a "standard hash function" like SHA3. There exists a polynomial time algorithm A that computes the $\mathcal{H}$-labeling of the graph $G_{N_c,k_G,\gamma_1,\gamma_2}$ in at most $N_c n_{\mathcal{H}}$-space.

Assumption 2 (Memory-hardness of Graph-labeling with Hash)**.** Suppose that Assumption 1 is true for the hash function $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{n_{\mathsf{hs}}}$ (with collision-probability $\varepsilon_{\mathsf{hs}}$). Then for any $k, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t \in \mathrm{poly}(\lambda)$ such that $t < 2^{k_G}\gamma_1 N_c$ and $s \leq \delta N_c n_{\mathsf{hs}}$ for some $\delta \in [0, \gamma_2 - 2\gamma_1)$, the above construction is $(k, n, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t, \varepsilon_{\mathsf{mhf}})$-MHF where:

$$n = n_{\mathsf{hs}} \qquad s_{\mathsf{mhf}} \geq k + n_{\mathsf{hs}}(N_c + \log(N) + 1) + n$$
$$\varepsilon_{\mathsf{mhf}} \leq \exp\left(\frac{-n_{\mathsf{hs}} N_c(\beta - \delta)}{N \log(N)}\right) + (s + f)\varepsilon_{\mathsf{hs}} + 2^{-\gamma_{\mathsf{hs}} n_{\mathsf{hs}}} + 2^{-k}$$

for $\beta = \gamma_2 - 2\gamma_1$ and a constant $\gamma_{\mathsf{hs}} \in (0, \frac{1}{2}]$.

[25] Since popular memory-hard functions like SCrypt [Tar] are not conjectured to provide exponential space-time trade-off, we are unable to use them here.

Notes on Assumption 2. First notice that, by our construction and Assumption 1, which says that there exists a polynomial-time algorithm that can label the graph $G_{N_c,k_G,\gamma_1,\gamma_2}$, it is straightforward to see the bounds on n and s_{mhf}. In fact those are not even part of Assumption 2, although for simplicity we state them as assumption. The main conjecture is made on the upper-bound of $\varepsilon_{\mathsf{mhf}}$. Note that it has an expression similar to the the bound of NIPoS-error ν_{pos} in Theorem 2 of [FHMV17a] (see Page-17 of [FHMV17b]). First let us focus on the final three three terms $(s+f)\varepsilon_{\mathsf{hs}} + 2^{-\gamma_{\mathsf{hs}} n_{\mathsf{hs}}} + 2^{-k}$. These terms are derived provably (again made part of the assumption for simplicity) from the collision resistance of Hash, guessing probability of the output of $M_{G,\mathsf{Hash}}$ (assuming that it's output has min-entropy at least $\gamma_{\mathsf{hs}} n_{\mathsf{hs}}$ for some $\gamma_{\mathsf{hs}} \in (0, \frac{1}{2}]$) and the guessing probability of RO output (without querying). For the term $\exp\left(\frac{-n_{\mathsf{hs}} N_c(\beta-\delta)}{N\log(N)}\right)$ we use the same expression from NIPoS-error ν_{pos} in Theorem 2 of FHMV. Essentially we assume that if the labeling of the same graph (used in Theorem 2 of FHMV) is done with respect to a concrete hash function (namely Hash) instead of a random oracle, then its space-hardness remains approximately the same. Since a concrete instantiations of a random oracle will definitely not meet all its properties we use a more "conservative" bound: first note that since our MHF-definition (Definition 15) does not have a small-space verification, the adversary's task is harder compared to the extractability game (Definition 10). In particular, in the MHF-definition the adversary wins if and only if it returns exactly the value that is the correct output of the MHF on a random input. In contrast, the NIPoS requires that the returned value (that is the proof $\tilde{\pi}$ in Definition 10) has to be verified correctly only by a "low-space" verifier (leaving room for some "faults"). However, if we naïvely put the number of faults to be zero in the expression from Theorem 2 of [FHMV17a] the corresponding probability would have been equal to zero as well. A closer look into the proof of the same theorem reveals that this would implicitly assume similar graph-labeling and pebbling Lemmas (see Section 7.2 for details) for the concrete hash function Hash. We feel that this would have been a rather strong assumption as we do not know how to even define a pebbling game with respect to a concrete hash function which is not modeled as a random oracle. Therefore, intuitively we just assume that instantiating the random oracle with a standard hash function is somewhat equivalent to allowing a few faults. We leave further analysis on our assumption as an interesting direction for future work.

From Assumption 2 we get the following corollary about our MHF-candidate:[26]

Corollary 10. *Suppose that Assumption 1 holds for the hash function* $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{n_{\mathsf{hs}}}$ *and based on that Assumption 2 holds for our construction based on a SoLEG* $G_{N_c,k_G,\gamma_1,\gamma_2}$ *with* $N = N_c(k_G+1)$ *nodes and* $\deg = O(1)$ *in-degree such that:*

$$n_{\mathsf{hs}} = \lambda^2 \qquad \beta = \gamma_2 - 2\gamma_1 \in (0,1) \qquad k_G = \lambda - 1$$
$$N_c = \lambda^3 \qquad \varepsilon_{\mathsf{hs}} \in \mathrm{negl}(\lambda).$$

Then, for any $\delta \in (0,\beta)$, *any* $\varepsilon > 0$ *and any* $\gamma_{\mathsf{hs}} \in (0, \frac{1}{2}]$ *our construction is a* $(k, n, s_{\mathsf{mhf}}, t_{\mathsf{mhf}}, s, f, t, \varepsilon_{\mathsf{mhf}})$*-MHF for* $t, f, t_{\mathsf{mhf}} \in \mathrm{poly}(\lambda)$ *and:*

$$k = O(\lambda^{\varepsilon}) \qquad n = \lambda^2 \qquad s_{\mathsf{mhf}} = O(\lambda^5)$$
$$s \le \delta \cdot \lambda^5 \qquad \varepsilon_{\mathsf{mhf}} \le \exp\left(\frac{-(\beta-\delta)\lambda}{\log(\lambda)}\right) + \mathrm{negl}(\lambda) \in \mathrm{negl}(\lambda)$$

[26] We remark that this corollary is very similar to Corollary 1 of [FHMV17b] as one may expect. However the parameters here are much better in terms of efficiency.

Furthermore, for making $\varepsilon_{\mathsf{mhf}} \approx 2^{-80}$*, we need to have* $\lambda \approx 2300$*. Choosing standard values for other parameters,* $\delta = 0.1, \beta = 0.9, \gamma_{\mathsf{hs}} = 0.001$ *we get concrete parameters for our MHF-construction as:*

$$k \geq 80 \qquad n \approx 670 \text{ KB} \qquad s_{\mathsf{mhf}} \approx 8000 \text{ TB} \qquad s \leq 800 \text{ TB} \qquad \varepsilon_{\mathsf{mhf}} \approx 2^{-80}$$

8.5 Instantiating VC

Our NIPoS construction can be instantiated with any VC for which the verification can be done in small space (compared to computing the function itself). In this work we concretely consider such a scheme, known as Pinocchio [PHGR13].

Space requirements of Pinocchio Verifier. Without giving formal arguments on the space-bound, we rely on the following assumption on the Pinocchio verification algorithm. Note that these bounds are independent of the space-bound of the function (in this case that is $M_{G,\mathsf{Hash}}$) to be verified. We briefly provide some justifications of that afterwards. We refer the reader for more details about the algorithm and the time complexity to the original paper [PHGR13].

Assumption 3 (Space-bounded Verification)**.** Let $\mathbb{G}$ be a (as considered in [PHGR13]) cyclic subgroup of points in $E(\mathbb{F}_p)$; $E(\mathbb{F}_p)$ denotes an elliptic curve over $\mathbb{F}_p$ where $p \in \exp(\lambda)$ is a prime.[27] Then for a function $F : \{0,1\}^k \to \{0,1\}^n$, the Pinocchio verification algorithm (see Protocol 2 of [PHGR13]) requires $k+n+O(\lambda)$-bit space asymptotically and $\approx k+n+300 \cdot \lceil \log p \rceil$ bits concretely.

Notes on Assumption 3. First notice that the size of the proof is equal to 8 elements of a group $\mathbb{G}$. Hence, the size of the proof is upper bounded by $16 \cdot \lceil \log(p) \rceil$. The verifier has the verifiaction key vk_F hardcoded and it gets as input (i) the input and output values of the function F, i.e. $k+n$ bits in total, and (ii) a proof of correct computation, π_{vc} which consists of 8 group elements. Thus, on high level, the verifier needs to store the input and output,[28] has to be able to further store constant number of group elements, have enough space to add two points on the curve and compute a non-trivial bilinear map $e \colon \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$ [BF01] where $\mathbb{G}'$ is subgroup of $E(\mathbb{F}_{p^2})$ and the target group $\mathbb{G}_T$ is a subgroup of $\mathbb{F}^*_{p^2}$. A point on an elliptic curve can be stored using constantly many field elements. Addition of two curve points requires only constantly more space than performing an arithmetic operation in the field for which we need $O(\log(p))$ space. Computing the bilinear map requires constantly more space than adding two curve points. Thus, asymptotically, the space complexity of the verifier can be assumed to be $k+n+O(\log(p)) = k+n+O(\lambda)$. It is more challenging to have a reasonable measurement of the concrete space complexity of the verifier since it depends not only on the concrete implementation of the verification algorithm itself but also on the concrete implementation of all its subprotocols including the basic arithmetic operations in the finite field $\mathbb{F}_p$. Assuming that Ver is implemented in a sequential manner, e is the modified Weil pairing as defined in [BF01] computed using Miller's algorithm [Mil86], we can roughly assume the space complexity of the verifier as $s^{vc}_{\mathsf{V}} \leq k+n+300 \cdot \lceil \log(p) \rceil$.

[27] To achieve 128-bits of security, as suggested by [Aur], we will set $\lceil \log(p) \rceil \approx 1536$.

[28] An space-optimized verification algorithm might be used to ensure that the verifier never has to read the entire input (or to write the entire output), if those are parsed in a "streaming manner". But we choose to follow a rather conservative approach and assume that the verifier needs to store the input/output at all time.

Space requirement of Pinocchio Prover. The space requirement of the prover algorithm Prove_{ek_F} (with hardcoded evaluation-key ek_F) mainly depends on the space-complexity of the function F to be verified. Suppose that F can be computed by a (s_F, t_F)-bounded algorithm, then the space requirement of Prove_{ek_F} must be at least $\max(s_F, n_{\mathsf{vc}})$ because the prover also outputs the proof of computation of length n_{vc}. For Pinocchio, n_{vc} is equal to $16\lceil \log p \rceil$. But since we are concerned with memory-hard functions, we simply assume that $s_F \gg n_{\mathsf{vc}}$. However, in reality it could be much larger than that as the prover has to (i) evaluate the circuit (ii) interpolate a polynomial of degree d, that is upper bounded by the number of gates in the circuit (more precisely, it is equal to the number of multiplication gates in the circuit) (iii) divide two polynomials of degree d (iv) finish the computation of the proof (that consists of 8 group elements of size $\lceil \log p \rceil$) by scalar multiplications and additions of points on elliptic curve (for more details we refer to [PHGR13]). We will be assuming a (possibly loose) lower bound of Pinocchio prover's space requirement as stated in the assumption below.

Assumption 4 (Lower-bound on Prover's space)**.** Let $\mathbb{G}$ be a (as considered in [PHGR13]) cyclic subgroup of points in $E(\mathbb{F}_p)$, where $E(\mathbb{F}_p)$ denotes an elliptic curve over a $\mathbb{F}_p$ where $p \in \exp(\lambda)$ is a prime. Consider a function F that is computable by a (s_F, t_F)-bounded algorithm for $s_F, t_F \in \mathrm{poly}(\lambda)$ and also assume that $s_F \gg \lceil 16 \log(p) \rceil$. Then the Pinocchio prove algorithm (see Protocol 2 of [PHGR13]) requires at least s_F-bit space.

Combining Assumption 3 and Assumption 4 we conclude:

Corollary 11. *Let $\lambda \in \mathbb{N}$ be the security parameter and let $F : \{0,1\}^k \to \{0,1\}^n$ be a deterministic function that is computable by an $(s_F, \mathrm{poly}(\lambda))$-space-time bounded algorithm. Then there exists an explicit $(s_F, \mathrm{poly}(\lambda), s^{vc}_{\mathsf{V}}, \mathrm{poly}(\lambda), s^{vc}_{\mathsf{P}}, \mathrm{poly}(\lambda), k, n, n_{\mathsf{vc}}, \mathrm{negl}(\lambda))$-non-interactive publicly verifiable computation construction, where:*

$$s^{vc}_{\mathsf{V}} = k + n + O(\lambda) \qquad s^{vc}_{\mathsf{P}} \geq s_F \qquad n_{\mathsf{vc}} = O(\lambda)$$

Furthermore, in concrete terms, to get $\varepsilon_{\mathsf{vc}} \approx 2^{-128}$, choosing $\lceil \log p \rceil \approx 1536$ (following [Aur]) we can have estimations of the verifier's space $s^{vc}_{\mathsf{V}} \approx 58$ KB $+ k + n$ and the proof-size $n_{\mathsf{vc}} \approx 3$ KB.

8.6 Instantiating partially unique NIPoS and PExt-NIPoS

Putting together the instantiations of MHF and VC, we can get a (partially) unique extractable NIPoS based on four heuristic assumptions (Assumptions 1–4). Plugging in the parameters from Corollary 10 and Corollary 11 into Lemma 9, we obtain the following corollary:

Corollary 12 (MHF-based NIPoS with uniqueness)**.** *For any $\varepsilon > 0$ and a $\delta \in (0,1)$ there is an explicit construction of $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS which has $(u_{\mathsf{pos}}, \varepsilon_{\mathsf{unique}})$-uniqueness and $(s, f, t, \eta, \varepsilon_{\mathsf{pos}})$-extractability for any $f, t \in \mathrm{poly}(\lambda)$ as long as:*

$$k_{\mathsf{pos}} \in \mathrm{poly}(\lambda^{\varepsilon}) \qquad n_{\mathsf{pos}} = O(\lambda^2) \qquad s_{\mathsf{P}} = \Omega(\lambda^5) \qquad s \leq \delta\lambda^5$$
$$s_{\mathsf{V}} = O(\lambda^2) \qquad u_{\mathsf{pos}} = \lambda^2 \qquad \varepsilon_{\mathsf{unique}} \in \mathrm{negl}(\lambda) \qquad \eta = O(\log(\lambda)) \qquad \varepsilon_{\mathsf{pos}} \in \mathrm{negl}(\lambda).$$

Instantiating this construction with $\lambda = 2300$ to get $\varepsilon_{\mathsf{pos}} \approx 2^{-80}$, and setting $k_{\mathsf{pos}} = 4$ KB, $|\mathcal{Q}_{\mathcal{H}}(\mathsf{B})| \leq 2^{64}$ and $\delta = 0.1$, we obtain a $(k_{\mathsf{pos}}, n_{\mathsf{pos}}, s_{\mathsf{P}}, s_{\mathsf{V}})$-NIPoS which has $(u_{\mathsf{pos}}, \varepsilon_{\mathsf{unique}})$-uniqueness and $(s, f, t, \eta, \varepsilon_{\mathsf{pos}})$-extractability where f, t can be set to any large enough value and

$$k_{\mathsf{pos}} = 4 \text{ KB} \qquad n_{\mathsf{pos}} \approx 673 \text{ KB} \qquad s_{\mathsf{P}} \geq 8000 \text{ TB} \qquad s \leq 800 \text{ TB}$$
$$s_{\mathsf{V}} \approx 740 \text{ KB} \qquad u_{\mathsf{pos}} \approx 670 \text{ KB} \qquad \varepsilon_{\mathsf{unique}} \approx 2^{-128} \qquad \eta = 64 \qquad \varepsilon_{\mathsf{pos}} \approx 2^{-80}.$$

9 Instantiating and comparing our NMC constructions

We propose four constructions of space-bounded (leaky) non-malleable codes that support unbounded tampering. All of them are based on non-interactive proof of space. Two require proof-extractability whereas the other two are based on standard extractability. I this section we provide asymptotic bounds for parameters of each construction. Additionally, we compare all constructions with respect to concrete values. Although our constructions have different merits, we believe this comparison is important. Let us begin by summarizing all our constructions briefly:

- **Construction-1 (based on SoLEG-extension):** This encoding scheme is constructed from *poof-extractable NIPoS* using Theorem 1. The proof-extractable NIPoS is instantiated with the challenge-hard graph *built in this paper* via SoLEG-extension. The parameters of the proof-extractable NIPoS where stated in Corollary 6.
- **Construction-2 (based on PTC's graph):** This encoding scheme is also constructed from *poof-extractable NIPoS* using Theorem 1. The proof-extractable NIPoS is instantiated with the PTC's challenge-hard graph proposed in [PTC76]. See Corollary 7 for parameters of the NIPoS.
- **Construction-3 (based on FHMV's NIPoS):** This encoding scheme is constructed from an *extractable NIPoS* using Corollary 1. The underlying NIPoS is an instantiation is from FHMV [FHMV17a] which is in turn a variant of [RD16]'s proof-of-space construction. The parameters of the NIPoS can be found in [FHMV17b, Corollary 1].
- **Construction-4 (based on MHF):** This encoding scheme is again constructed from an *extractable NIPoS* with partial uniqueness using Corollary 1. The underlying NIPoS is instantiated by a (heuristic) memory-hard function and a verifiable computation scheme. Corollary 12 summarizes the parameters of this NIPoS construction.

9.1 Instantiations from different PExt-NIPoS

Let us now discuss each code construction in more detail and formally state asymptotic bounds for code parameters.

Instantiating with CHG-based NIPoS. Instantiating with the CHG-based PExt-NIPoS from Section 7.4 we obtain two encoding schemes. Plugging-in the parameters from the SoLEG-based NIPoS (Corollary 6) and the PTC-based NIPoS (Corollary 7) into Theorem 1 respectively we obtain the following two corollaries with asymptotic bounds.

Corollary 13 (Construction-1 based on SoLEG-extension). *Let $n_{\mathcal{H}} \in \omega(\lambda)$ be the RO output length. Assume the existence of a $(*, n_{\mathcal{H}}, \mathrm{poly}(\lambda), \mathrm{negl}(\lambda))$-PRF. Then, for any $\theta \in \mathrm{poly}(\lambda)$ the (k, n)-code built upon the NIPoS in Corollary 6 is $(\ell, s, f, p, \theta, d, \mathrm{negl}(\lambda))$-SP-NMC-SD in the ROM, where*

$$\ell = \omega(\lambda^{1.5} \log^2 \lambda) \qquad \ell + \omega(\log \lambda) \leq k \leq O(\lambda^{1.5} \log \lambda) \cdot n_{\mathcal{H}} \qquad n = \omega(\lambda^{1.5} \log \lambda) \cdot n_{\mathcal{H}}$$
$$s, f = \Theta(\lambda^2) \cdot n_{\mathcal{H}} \qquad n \leq p \leq n + k - O(\log \lambda) \qquad d = \omega(\lambda^{1.5} \log \lambda) \cdot n_{\mathcal{H}} \,.$$

Corollary 14 (Construction-2 based on PTC's graph). *Let $n_{\mathcal{H}} \in \omega(\lambda)$ be the RO output length. Assume the existence of a $(*, n_{\mathcal{H}}, \mathrm{poly}(\lambda), \mathrm{negl}(\lambda))$-PRF. Then, for any $\theta \in \mathrm{poly}(\lambda)$ the*

(k,n)-code built upon the NIPoS in Corollary 7 is $(\ell, s, f, p, \theta, d, \mathrm{negl}(\lambda))$-SP-NMC-SD in the ROM, where

$$\ell = \omega(\lambda \log^3 \lambda) \qquad \ell + \omega(\log \lambda) \leq k \leq O(\lambda \log^2 \lambda) \cdot n_{\mathcal{H}} \qquad n = \omega(\lambda \log^2 \lambda) \cdot n_{\mathcal{H}}$$
$$s, f = \Theta(\lambda^2) \cdot n_{\mathcal{H}} \qquad n \leq p \leq n + k - O(\log \lambda) \qquad d = \omega(\lambda \log^2 \lambda) \cdot n_{\mathcal{H}} .$$

Instantiating with FHMV's NIPoS. First we observe that the NIPoS used in FHMV, that is the one proposed by Ren and Devadas [RD16], has ≈ 0 uniqueness. Combining this observation and plugging-in the asymptotic parameters from FHMV [FHMV17b, Corollary 1] to our Corollary 1, we obtain an encoding scheme that satisfies continuous space-bounded non-malleability with the following parameters.

Corollary 15 (Construction-3 from FHMV's NIPoS). *Consider a $(*, \mathrm{poly}(\lambda), \mathrm{negl}(\lambda))$-PRF. Then for any $\theta \in \mathrm{poly}(\lambda)$ and any $f \in \mathrm{poly}(\lambda)$ and $\delta \in (0,1)$ the (k,n)-code built upon the NIPoS of [FHMV17b, Corollary 1] is $(\ell, s, f, p, \theta, d, \mathrm{negl}(\lambda))$-SP-NMC-SD in the ROM, where*

$$O(\lambda^4) = k > \ell + O(\log(\lambda)) \qquad n = O(\lambda^4) \qquad d = O(\lambda^4)$$
$$\ell = p - k + O(\log \lambda) = O(\lambda^4) \qquad O(\lambda^4) \leq p < n + k - O(\log \lambda) \qquad O(\lambda^4) \leq s \leq \delta\lambda^5 .$$

We provide a remark comparing the above corollary with Theorem 3 of [FHMV17b].

Remark 12. *There are two major differences compare to [FHMV17b, Theroem 3]. First here the leakage grows with $\lceil \log(\theta) \rceil$ compared to in [FHMV17a] where $\ell = \theta \cdot O(\log(\lambda))$, which grows linearly with θ – this implies that we can set θ to be any unbounded polynomial function of λ. Second, here we achieve a weaker notion, in that a self-destruct is present. As explained in Section 4, in our case self-destruct is inevitable as otherwise it is impossible to tolerate an unbounded θ. For a more detail discussion we refer to [FHMV17a].*

Instantiating with MHF-based NIPoS. Finally, let us discuss parameters of our coding scheme based the extractable NIPoS with partial uniqness built from heuristic MHF and VC. Plugging-in Corollary 12 into Corollary 1, we obtain an encoding scheme satisfying space-bounded continuous non-malleability for parameters provided in the following corollary.

Corollary 16 (Construction-4 based on heuristic MHF). *For any $\varepsilon > 0$ and a $\delta \in (0,1)$ there is an explicit construction of a (k,n)-code which is $(\ell, s, f, p, \theta, d, \varepsilon_{\mathsf{nm}})$-SP-NMC-SD as long as $\theta, f \in \mathrm{poly}(\lambda)$ and:*

$$O(\lambda^2) = k > \ell + O(\log \lambda) \qquad n = O(\lambda^2) \qquad \ell = p - k - \lambda^2 + O(\log \lambda)$$
$$O(\lambda^2) < s \leq \delta\lambda^5 \qquad n \leq p < n + k - O(\log \lambda) \qquad d = O(\lambda^2) \qquad \varepsilon_{\mathsf{nm}} \in \mathrm{negl}(\lambda)$$

9.2 Comparing concrete parameters

Assumptions. The first three constructions are based on "memory-hard graphs". The hardness can be proven in the random oracle model via standard pebbling games (see a discussion in Section 7.2). The main proof relies on combinatorial arguments. In contrast Construction-4 relies on heuristic arguments for space bounds. The main assumptions are (Assumption 2 and 1) that memory-hard graphs retain their space-hardness when instantiated with concrete hash functions. This is needed because the standard pebbling arguments fall short when the hash function is not modeled as a random oracle. We also rely on a few other assumptions (namely Assumption 3 and 4) regarding the underlying verfiable computation. For all our constructions we need a PRF with standard security as the proofs depend on the pseudorandomness guarantee of the PRF.

Our setting. Since our constructions are obtained from different techniques and achieve different bounds, it is important to fix a common measure with respect to which a comparison makes sense. We choose to fix a standard security measure. In particular, we set $\varepsilon_{\mathsf{nm}} = 2^{-80}$ in the Definition 8 — that is how we can estimate the values of the other parameters (namely, k, n, ℓ, s, f, p, d) to get 80-bit security. We also choose a reasonable values for the number of tampering queries: $\theta = 2^{16}$.[29] Whenever there is a term that depends on the number of RO queries made by a poly-time adversary, (for example $|\mathcal{Q}_{\mathcal{H}}(\mathsf{A})|$) we set that to 2^{64}. We assume that a poly-time adversary runs for 2^{80} time steps. Since in our setting (as discussed in Remark 3) ℓ is always as big as $p - n$ we compare the parameters considering $p \approx n$ to have *minimal leakage*. We choose small values for k (close to ℓ) within the supported range, although for most of our constructions much higher k is supported.

Using concrete instantiations of PExt-NIPoS (resp. Ext–NIPoS) and plugging-in them to Theorem 1 (resp. Corollary 1), we get the concrete parameters for the resulting CSNMCs. We provide a comparative study in Table 2.

Technique	NIPoS-type	k	$n, (\approx p)$	ℓ	d	$s(+f)$
CHG	SoLEG-based	1 MB	801 MB	0.8 MB	801 MB	1.1 GB$(+f)$
	PTC-based	257 MB	256 GB	256 MB	256 GB	256 GB$(+f)$
Ext	FHMV-based	226 TB	415 TB	225 TB	452 TB	800 TB
	MHF-based	4 KB	677 KB	3 KB	740 KB	800 TB

Table 2: This table shows approximate concrete parameters for the setting when $p \approx n$. Note that for PExt-NIPoS-based constructions the last column has bound on $s + f$, whereas for Ext-NIPoS-based constructions the bound is only on s as f can be set to arbitrary large value.

References

[AAG+16] Divesh Aggarwal, Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Optimal computational split-state non-malleable codes. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 393–417, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[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: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 538–557, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany.

[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, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 358–387, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[29] We stress that this value can be set much higher without affecting the main parameters significantly.

[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, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 33–62, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[ADL14] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 774–783, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[ADN+17] Divesh Aggarwal, Nico Dottling, Jesper Buus Nielsen, Maciej Obremski, and Erick Purwanto. Continuous non-malleable codes in the 8-split-state model. Cryptology ePrint Archive, Report 2017/357, 2017. https://eprint.iacr.org/2017/357.

[AGM+15] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Explicit non-malleable codes against bit-wise tampering and permutations. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 538–557, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[AKO17] Divesh Aggarwal, Tomasz Kazana, and Maciej Obremski. Inception makes non-malleable codes stronger. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 319–343, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 595–603, Portland, OR, USA, June 14–17, 2015. ACM Press.

[Aur] Aurore Guillevic and François Morain. Discrete logarithms. Book Chapter 9. https://hal.inria.fr/hal-01420485v1/document.

[BCS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. Cryptology ePrint Archive, Report 2016/027, 2016. http://eprint.iacr.org/2016/027.

[BDKM16] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 881–908, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[BDKM18] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 618–650, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.

[BM15] Rishiraj Bhattacharyya and Pratyay Mukherjee. Non-adaptive programmability of random oracle. *Theor. Comput. Sci.*, 592:97–114, 2015.

[CG14] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 440–464, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[CGM+16a] Nishanth Chandran, Vipul Goyal, Pratyay Mukherjee, Omkant Pandey, and Jalaj Upadhyay. Block-wise non-malleable codes. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016: 43rd International Colloquium on Automata, Languages and Programming*, volume 55 of *LIPIcs*, pages 31:1–31:14, Rome, Italy, July 11–15, 2016. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[CGM16b] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 499–530, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[DFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[DKW11a] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 335–353, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[DKW11b] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography*

Conference, volume 6597 of *Lecture Notes in Computer Science*, pages 125–143, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[DLSZ15] Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 427–450, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 37–54, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 434–452, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press.

[FHMV17a] Sebastian Faust, Kristina Hostáková, Pratyay Mukherjee, and Daniele Venturi. Non-malleable codes for space-bounded tampering. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 95–126, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[FHMV17b] Sebastian Faust, Kristina Hostakova, Pratyay Mukherjee, and Daniele Venturi. Non-malleable codes for space-bounded tampering. Cryptology ePrint Archive, Report 2017/530, 2017. http://eprint.iacr.org/2017/530.

[FLR+10] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[FMNV14] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 465–488, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[FMNV15] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. A tamper and leakage resilient von neumann architecture. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 579–603, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

[FMVW14] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 111–128, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[FNSV18] Antonio Faonio, Jesper Buus Nielsen, Mark Simkin, and Daniele Venturi. Continuously non-malleable codes with split-state refresh. In *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, Lecture Notes in Computer Science, pages 121–139. Springer, Heidelberg, Germany, 2018.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[GPR16] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 1128–1141, Cambridge, MA, USA, June 18–21, 2016. ACM Press.

[JW15] Zahra Jafargholi and Daniel Wichs. Tamper detection and continuous non-malleable codes. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 451–480, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[KOS17] Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Four-state non-malleable codes with explicit constant rate. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 344–375, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[KOS18] Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Non-malleable randomness encoders and their applications. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 589–617, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[LL12] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 517–532, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

[Mil86] Victor Miller. Short programs for functions on curves. Unpublished manuscript, 1986.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013: 4th Innovations in Theoretical Computer Science*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. Association for Computing Machinery.

[Muk15] Mukherjee, Pratyay. *Protecting Cryptographic Memory against Tampering Attack*. PhD thesis, 2015.

[OPVV18] Rafail Ostrovsky, Giuseppe Persiano, Daniele Venturi, and Ivan Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions. Lecture Notes in Computer Science, pages 608–639, Santa Barbara, CA, USA, 2018. Springer, Heidelberg, Germany.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[Pie18] Krzysztof Pietrzak. Proofs of catalytic space. *IACR Cryptology ePrint Archive*, 2018:194, 2018.

[PTC76] Wolfgang J Paul, Robert Endre Tarjan, and James R Celoni. Space bounds for a game on graphs. *Mathematical systems theory*, 10(1):239–251, 1976.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 262–285, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[Tar] Tarsnap. The scrypt key derivation function. Webpage. https://eprint.iacr.org/2017/1125.

A Proof of Theorem 1

To simplify the notation in the proof, let us write

$$\mathsf{D}^{\mathrm{r}} := \mathsf{D}^{\mathcal{H}(\cdot),\mathcal{O}^{\Pi,x,\mathsf{pp},s,p}_{\text{real-sd}}(\cdot)},\ \mathsf{D}^{\mathrm{m}} := \mathsf{D}^{\mathcal{H}(\cdot),\mathcal{O}^{\mathsf{S}_2,\ell,x,s,\mathsf{pp}}_{\text{sim}}(\cdot)},\ \mathsf{D}^{\mathrm{s}} := \mathsf{D}^{\mathsf{S}_1(\cdot),\mathcal{O}^{\mathsf{S}_2,\ell,x,s,\mathsf{pp}}_{\text{sim}}(\cdot)}$$

to denote the interaction in the real, resp. mental, resp. simulated experiment.

Consider an adversary D which makes θ queries to $\mathcal{O}_{\text{real-sd}}$. By Definition 8, we need to prove that the simulator $\mathsf{S}^{\mathsf{D}} = (\mathsf{S}^{\mathsf{D}}_1, \mathsf{S}^{\mathsf{D}}_2)$ defined in Fig. 1 is such that, for all values $x \in \{0,1\}^k$ satisfying

$$\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \colon \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Init}^{\mathcal{H}}(1^\lambda)\right] - \Pr\left[\mathsf{D}^{\mathrm{s}}(\mathsf{pp}) = 1 \colon \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Init}^{\mathsf{S}_1}(1^\lambda)\right]\right| \leq \varepsilon_{\mathsf{nm}}(\lambda).$$

A straightforward reduction to the pseudorandomness of the PRF yields:

$$\left|\Pr\left[\mathsf{D}^{\mathrm{s}}(\mathsf{pp}) = 1 \colon \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Init}^{\mathsf{S}_1}(1^\lambda)\right] - \Pr[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \colon \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Init}^{\mathcal{H}}(1^\lambda)\right]\right| \leq \varepsilon_{\mathsf{pr}}(\lambda).$$

Hence, our goal is to show that

$$\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \colon \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Init}^{\mathcal{H}}(1^\lambda)\right] - \Pr[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \colon \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Init}^{\mathcal{H}}(1^\lambda)\right]\right| \leq \varepsilon_{\mathsf{pos}}(\lambda).$$

Let us now fix some arbitrary $x \in \{0,1\}^k$ and over the randomnesses of the randomized procedures Init, Encode, the distinguisher D and the simulators $\mathsf{S}_1, \mathsf{S}_2$ let us define an event NoTamp as

$$\textsc{NoTamp} := L_1(x) = 0^{\ell_1}.$$

In other words, the event NoTamp occurs when the first leakage function outputs 0^{ℓ_1} which signals that none of the tampering algorithm tampers to a codeword that decodes to something else than the target message x.

Claim 2. *It holds that*

$$\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \mid \textsc{NoTamp}\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \mid \textsc{NoTamp}\right]\right| = 0$$

Proof. In case of the event NoTamp, the simulator answers all tampering queries by the symbol $\mathsf{same}^\star$ which is exactly what the tampering oracle does in the real experiment. □

The above claim implies that:

$$\begin{aligned} &\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1\right]\right| \\ &\leq \left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \mid \neg\textsc{NoTamp}\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \mid \neg\textsc{NoTamp}\right]\right|. \end{aligned} \tag{2}$$

If the event $\neg$NoTamp occurs, then $j^\star := \mathsf{bit}^{-1}(L_1(x))$ is such that $0 < j^\star \leq \theta$ which means that $\mathsf{A}_{j^\star}$ is the first tampering algorithm that tampers to a codeword $\tilde{c}$ which decodes to $\tilde{x} \neq x$. As a next step, we define an event

$$\textsc{Invalid} := \mathsf{Decode}^{\mathcal{H}}_{\mathsf{pp}}(\tilde{c}) = \perp$$

and prove the following claim.

Claim 3. *It holds that*

$$\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \mid \neg\textsc{NoTamp} \wedge \textsc{Invalid}\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \mid \neg\textsc{NoTamp} \wedge \textsc{Invalid}\right]\right| = 0$$

Proof. If the tampering algorithm $\mathsf{A}_{j^\star}$ tampers to an invalid codeword, then the second leakage oracle returns the flag 0^{ℓ_2}. The simulator S_2 outputs the symbol $\mathsf{same}^\star$ for all tampering queries $\mathsf{A}_1, \ldots, \mathsf{A}_{j^\star-1}$ and $\perp$ for all tampering queries $\mathsf{A}_{j^\star}, \ldots, \mathsf{A}_\theta$. This is exactly what the tampering oracle does in the real experiment (recall that we consider non-malleable codes with self-desctruct mechanism). □

Let us denote $\mathrm{E} := \neg\textsc{NoTamp} \wedge \neg\textsc{Invalid}$. Using the claim from above and Eq. (2), we can bound the the probability that D succeeds as follows:

$$\begin{aligned} &\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1\right]\right| \\ &\qquad \leq \left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \mid \mathrm{E}\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \mid \mathrm{E}\right]\right|. \end{aligned} \tag{3}$$

If the event E occures, then $A_{j^\star}$ tampered to a valid codeword $\tilde{c}$ that decodes to $\tilde{x} \neq x$. Next, we define the event

$$\textsc{NotExtr} := \mathsf{K}(\mathsf{pp}, \mathcal{Q}_{\mathcal{H}}(\mathsf{D}), z) \neq \tilde{c},$$

where $z := F_{\mathsf{hint}}(\mathsf{pp}, \mathcal{Q}_{\mathcal{H}}(\mathsf{D}), \tilde{c})$. In other words, the event $\textsc{NotExtr}$ happens when $\tilde{c}$ is not extraxtable from the oracle query table $\mathcal{Q}_{\mathcal{H}}(\mathsf{D})$. Using Eq. (3), we can bound the probability that D succeeds as follows:

$$\begin{aligned} &\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1\right]\right| \\ &\leq \left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp}) = 1 \mid \mathrm{E} \wedge \neg\textsc{NotExtr}\right] - \Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp}) = 1 \mid \mathrm{E} \wedge \neg\textsc{NotExtr}\right]\right| \\ &\qquad + \Pr\left[\textsc{NotExtr} \mid \mathrm{E}\right]. \end{aligned} \tag{4}$$

We complete the proof by showing the following two claims.

Claim 4. *It hold that* $\Pr\left[\textsc{NotExtr} \mid \mathrm{E}\right] \leq \varepsilon_{\mathsf{pos}}$.

Proof. We construct a PPT algorithm B running in game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},x}(\lambda)$, that attempts to break the proof-extractability of the NIPoS with probability $\varepsilon_{\mathsf{pos}}$:

Algorithm $\mathsf{B}^{\mathcal{H}}_{\mathsf{D}}$:

1. Receive as input $\mathsf{pp}_{\mathsf{pos}} \leftarrow\!\!\$\ \mathsf{Setup}^{\mathcal{H}}(1^\lambda)$, $x \in \{0,1\}^k$, and $\pi \leftarrow\!\!\$\ \mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(x)$.
2. Assign $(c, \sigma) := (x||\pi, 0^{s-n})$, $\mathcal{Q}_{\mathcal{H}}(\mathsf{D}) := \emptyset$, and define $\mathsf{A} := \mathsf{Id}$, where $\mathsf{Id}\colon \{0,1\}^s \to \{0,1\}^s$ is the identity function.
3. For $i \in [\theta]$ proceed as follows:
   (a) Answer random oracle queries made by D, before A_i is chosen, by querying $\mathcal{H}$ in game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},x}(\lambda)$ and forwarding the answers to D; in addition, store these queries in the table $\mathcal{Q}_{\mathcal{H}}(\mathsf{D})$.
   (b) On receiving A_i, set $\mathsf{A} := \mathsf{A} \circ \mathsf{A}_i$ and run $(\tilde{c}, \tilde{\sigma}) := \mathsf{A}_i(c; \sigma)$.
   (c) Compute $\tilde{x} := \mathsf{Decode}^{\mathcal{H}}_{\mathsf{pp}}(\tilde{c})$ and run $z := F_{\mathsf{hint}}(\mathsf{pp}, \mathcal{Q}_{\mathcal{H}}(\mathsf{D}), \tilde{c})$. If $\tilde{x} \neq \{x, \perp\}$ and $\tilde{c} \neq \mathsf{K}(\mathsf{pp}, \mathcal{Q}_{\mathcal{H}}(\mathsf{D}), z)$, then output A and stop. Otherwise let $(c, \sigma) := (\tilde{c}, \tilde{\sigma}_0||0^{s-p})$, where $\tilde{\sigma}_0||\tilde{\sigma}_1 := \tilde{\sigma}$, and send $\tilde{x}$ to D if $\tilde{x} \neq x$ and $\mathsf{same}^\star$ if $\tilde{x} = x$.

We observe that B perfectly simulates the view of D. So, if the event $\textsc{NotExtr}$ happens, then B wins the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},x}(\lambda)$. By proof-extractability, we have

$$\varepsilon_{\mathsf{pos}} \geq \Pr[\mathbf{G}^{\mathsf{pext}}_{\mathsf{D},x}(\lambda) = 1] \geq \Pr[\textsc{NotExtr} \mid \mathrm{E}].$$

□

Claim 5. *It holds that*

$$\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(1^\lambda)=1 \mid \mathrm{E}\wedge\neg\textsc{NotExtr}\right]-\Pr\left[\mathsf{D}^{\mathrm{m}}(1^\lambda)=1 \mid \mathrm{E}\wedge\neg\textsc{NotExtr}\right]\right|=0.$$

Proof. When the simulator is able to reconstruct the codeword $\tilde{c}=(\tilde{x},\tilde{\pi})$ from the query table $\mathcal{Q}_{\mathcal{H}}(\mathsf{D})$, then the simulator answers $\mathsf{same}^\star$ to all tampering queries $\mathsf{A}_1,\ldots,\mathsf{A}_{j^\star-1}$ and outputs the value $\tilde{x}$ to the tampering query $\mathsf{A}_{j^\star}$ which is exactly was the tampering oracle does in the real experiment. What remains to discuss is how the simulator answers the follow up tampering queries $\mathsf{A}_{j^\star+1},\ldots,\mathsf{A}_\theta$. Since the additional persistent space $\tilde{\sigma}_0$ is leaked by the leakage function L_2 the simulator is able to reconstruct the entire internal memory as $(\tilde{c},\tilde{\sigma}_0,0^{s-p})$. Hence, it can perfectly simulate the behavior of the tampering oracle and answer all the remaining tampering queries correctly. □

Combining the above two claims together with Eq. (4), we obtain

$$\left|\Pr\left[\mathsf{D}^{\mathrm{r}}(\mathsf{pp})=1:\ \mathsf{pp}\leftarrow\mathsf{Init}^{\mathcal{H}}(1^\lambda)\right]-\Pr\left[\mathsf{D}^{\mathrm{m}}(\mathsf{pp})=1:\ \mathsf{pp}\leftarrow\mathsf{Init}^{\mathsf{S}_1}(1^\lambda)\right]\right|\leq\varepsilon_{\mathsf{pos}}$$

as desired.

To complete the security analysis, it remains to argue about the size of leakage. The first leakage query L_1 outputs a ℓ_1-bit string which is equal to either the zero bit string or the binary representation of $j^\star$. Since $0<j^\star\leq\theta$, we have $\ell_1:=|\mathsf{bit}(\theta)|=\lceil\log\theta\rceil+1$. The second leakage query L_2 returns an ℓ_2-bit string which is either the zero bit string or contains (i) the hint z which is of length η, (ii) the additional persistent space which is of length $p-n$ and (iii) a bit 1.[30] Hence $\ell_2=\eta+p-n+1$. To conclude, the size of the leakage is

$$\ell=\ell_1+\ell_2=\lceil\log\theta\rceil+\eta+p-n+2.$$

B Proof of Theorem 2

From now on, for the proof-extractability game, we fix the random oracle length $n_{\mathcal{H}}\in\mathbb{N}$, the $(\beta,N_c,N,\tau_c,t,\epsilon_{\mathsf{peb}})$-challenge-hard graph $G:=G_{\mathrm{HARD}}=(V,E)$ with maximal indegree $\mathsf{deg}=O(1)$, the target sets $V_c^{(1)},\ldots,V_c^{(\tau_c)}\subseteq V$, the parameters $s_{\mathsf{P}},s_{\mathsf{V}},k_{\mathsf{pos}},\nu,\tau,\tau_c\in\mathbb{N}$ for the NIPoS scheme, the identity $id\in\{0,1\}^{k_{\mathsf{pos}}}$, and the parameters $s,f,t\in\mathbb{N}$ for the tampering class.

In the proof extractability game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$, we denote by B the outside adversary, A the (s,f,t)-bounded tampering algorithm, and $(\tilde{id},\tilde{\pi})$ the output of A. To make the bound more explicit, we denote by $Q_{\mathsf{B}}\in\mathrm{poly}(\lambda)$ (resp. $Q\in\mathrm{poly}(\lambda)$) an upper bound on the number of RO queries made by B (resp. A).

Step 1: First, we consider a bad event where the adversary guesses the RO output without making the random oracle query.

> Event GUESS: There exists a random oracle input-output pair $(\mathsf{inp},\mathcal{H}(\mathsf{inp}))\in\mathcal{Q}_{\mathcal{H}}(\tilde{\pi})\cup\mathcal{Q}_{\mathcal{H}}(\mathsf{A})\cup\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$, such that B or A guesses the output $\mathcal{H}(\mathsf{inp})$ (i.e., using the output $\mathcal{H}(\mathsf{inp})$ as a part of an RO query input (or the proof-of-space) before $\mathcal{H}(\mathsf{inp})$ is ever queried).

Claim 6.

$$\Pr\left[\textsc{Guess}\right]\leq\frac{\mathsf{deg}\cdot(Q_{\mathsf{B}}+Q)\cdot(Q_{\mathsf{B}}+Q+\nu)}{2^{n_{\mathcal{H}}}},$$

The probability is over the choice of the random oracle, and the coin tosses of $\mathsf{Setup},\mathsf{B}$.

[30] This additional bit is needed to distinguish the string from the flag 0^{ℓ_2}.

Proof. Denote by E_i^* $(1 \le i \le Q_{\mathsf{B}}+Q+\nu)$ the event that B (or A) guesses the ith input-output pair $(\mathsf{inp}_i, \mathcal{H}(\mathsf{inp}_i))$ in $\mathcal{Q}_{\mathcal{H}}(\tilde{\pi}) \cup \mathcal{Q}_{\mathcal{H}}(\mathsf{A}) \cup \mathcal{Q}_{\mathcal{H}}(\mathsf{B})$. We first bound $\Pr[\mathrm{E}_i^*]$ for every i: At the moment that the adversary determines/queries inp_i, $\mathcal{H}(\mathsf{inp}_i)$ is uniformly random as it has not been queried before; since the adversary has at most $\mathsf{deg}\cdot(Q_{\mathsf{B}}+Q)$ input slots to guess the output before knowing $\mathcal{H}(\mathsf{inp}_i)$, the probability of guessing correctly is at most $(\mathsf{deg}\cdot(Q_{\mathsf{B}}+Q))/2^{n_{\mathcal{H}}}$.

Finally, since there are at most $Q_{\mathsf{B}}+Q+\nu$ RO input-output pairs in $\mathcal{Q}_{\mathcal{H}}(\tilde{\pi})\cup\mathcal{Q}_{\mathcal{H}}(\mathsf{A})\cup\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$, the claim holds by union bound.

□

Step 2: We consider the second bad event that the adversary opens the merkle root value $\tilde{\phi}_\ell$ to different labelings, i.e., the binding property is violated.

For a vector $\mathbf{z} \in \{\{0,1\}^{n_{\mathcal{H}}} \cup \{\perp\}\}^N$ and a value $\phi_\ell \in \{0,1\}^{n_{\mathcal{H}}}$, we say that the adversary *maximally committed* $\mathbf{z}$ to ϕ_ℓ, iff 1) for every $v \in [N]$ such that $\mathbf{z}_v \neq \perp$, the adversary computed (via querying RO) a Merkle tree path from position v (with label $\mathbf{z}_v$) to the root value $\tilde{\phi}_\ell$; 2) for every $v \in [N]$ such that $\mathbf{z}_v = \perp$, the adversary has never computed (via querying RO) a Merkle tree path from position v to the root value ϕ_ℓ. We consider the following event.

> Event UNBIND: Denote by $\tilde{\phi}_\ell$ the Merkle root value in the output proof-of-space $\tilde{\pi}$. There exist two different[31] vectors $\mathbf{z_1}, \mathbf{z_2} \in \{\{0,1\}^{n_{\mathcal{H}}} \cup \{\perp\}\}^N$, such that the adversary *maximally committed* both $\mathbf{z_1}$ and $\mathbf{z_2}$ to the Merkle root value $\tilde{\phi}_\ell$.

Claim 7.

$$\Pr[\textsc{Unbind}] \le \varepsilon_{\mathsf{col}},$$

where $\varepsilon_{\mathsf{col}} := (Q_{\mathsf{B}}+Q)^2/2^{n_{\mathcal{H}}}$ *is the collision probability of the hash function* $\mathcal{H}_{\mathsf{com}}$. *The probability is over the choice of the random oracle, and the coin tosses of* $\mathsf{Setup}, \mathsf{B}$.

Proof. First, we show that if the event UNBIND happens, then the adversary outputs an RO collision pair. We denote by $\mathbf{z_1}$ and $\mathbf{z_2}$ the two vectors that the adversary *maximally committed* to the Merkle root value $\tilde{\phi}_\ell$, and denote by $v \in [N]$ the minimal index such that $z_{1,v} \neq z_{2,v}$.

Since at least one of $z_{1,v}, z_{2,v}$ is an $n_{\mathcal{H}}$-bit string, by the second requirement in the definition of the *maximally committing*, both $z_{1,v}$ and $z_{2,v}$ are $n_{\mathcal{H}}$-bit strings. Hence, there exist two opening paths (from leaf v to the root), which have different leaf labels (i.e., $z_{1,v}, z_{2,v} \in \{0,1\}^{n_{\mathcal{H}}}$), but have the same Merkle root value $\tilde{\phi}_\ell$. Therefore, it must be the case that there exists an RO collision pair.

Second, we show that the collision probability $\varepsilon_{\mathsf{col}}$ is at most $(Q_{\mathsf{B}}+Q)^2/2^{n_{\mathcal{H}}}$. For every $i, j \in [Q_{\mathsf{B}}+Q]$ (where $i<j$), we denote by $\mathrm{E}_{i,j}^*$ the event that 1) the output of the adversary's ith RO query collides with that of the jth RO query; 2) the jth RO query $(\mathsf{inp}_j, \mathcal{H}(\mathsf{inp}_j))$ is the first query with input inp_j. We bound $\Pr[\mathrm{E}_{i,j}^*]$ for every i, j $(i<j)$: At the step where the adversary makes the jth RO query (but does not receive the answer), the input inp_j was fixed, the output $\mathcal{H}(\mathsf{inp}_i)$ was fixed, but $\mathcal{H}(\mathsf{inp}_j)$ has not been queried yet and thus is uniformly random. Thus the probability that $\mathcal{H}(\mathsf{inp}_j)$ collides with $\mathcal{H}(\mathsf{inp}_i)$ is at most $1/2^{n_{\mathcal{H}}}$.

Since there are at most $(Q_{\mathsf{B}}+Q)^2$ choices for (i,j), by union bound, the collision probability $\varepsilon_{\mathsf{col}}$ is at most $(Q_{\mathsf{B}}+Q)^2/2^{n_{\mathcal{H}}}$. □

Since

$$\Pr\left[\mathbf{G}_{\mathsf{B},id}^{\mathsf{pext}}(\lambda)=1\right] \le \Pr[\textsc{Guess}] + \Pr[\textsc{Unbind}] + \Pr\left[(\mathbf{G}_{\mathsf{B},id}^{\mathsf{pext}}(\lambda)=1) \wedge \neg\textsc{Guess} \wedge \neg\textsc{Unbind}\right],$$

[31] By different we mean there exists an index $v \in [N]$ such that $z_{1,v} \neq z_{2,v}$.

to bound $\Pr[\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)=1]$, it is now sufficient to bound the probability of the event $\mathrm{E}_1 := (\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)=1) \wedge \neg\textsc{Guess} \wedge \neg\textsc{Unbind}$. Intuitively, E_1 implies that the adversary A generates a valid proof that is not extractable by the knowledge extractor. Moreover, the proof is bound with a unique graph labeling.

Step 3: Next, we consider the third bad event that the labeling committed by the adversaries is *significantly* different from the graph labeling of G, that is, the labeling has large number of faults.

Let $m \in \mathbb{N}$ be a parameter that will be clear later, we define an event $\textsc{NoFault}_m$.

> Event $\textsc{NoFault}_m$: Denote by $\tilde{id}$ the output identity, and $\tilde{\phi}_\ell$ the Merkle root value in the output proof-of-space $\tilde{\pi}$. It happens that the adversaries (including both B and A) maximally committed (altogether) a vector $\mathbf{z}$ into the Merkle root value $\tilde{\phi}_\ell$ such that: $\mathbf{z}$ is an $\mathcal{H}_{\tilde{id}}$-labeling (of the graph G) with at most m faults.

Claim 8. *Denote by event* $\mathrm{E}_1 := (\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)=1) \wedge \neg\textsc{Guess} \wedge \neg\textsc{Unbind}$*, it holds that*

$$\Pr\left[\mathrm{E}_1 \wedge \neg\textsc{NoFault}_m\right] \leq (Q_{\mathsf{B}}+Q)\cdot\left(1-\frac{m}{N}\right)^{\tau} \leq (Q_{\mathsf{B}}+Q)\cdot\exp\left(-\frac{\tau\cdot m}{N}\right),$$

where the probability is over the choice of the random oracle, and the coin tosses of $\mathsf{Setup}, \mathsf{B}$.

Proof. Denote by E^*_i $(1 \leq i \leq Q_{\mathsf{B}}+Q)$ the event that the ith RO query input to $\mathcal{H}_{\mathsf{chal}}$ is $(\tilde{id}, \tilde{\phi}_\ell)$, where $\tilde{id}$ is the output identity and $\tilde{\phi}_\ell$ is the Merkle root value in $\tilde{\pi}$. We first bound $\Pr[\mathrm{E}^*_i \wedge \mathrm{E}_1 \wedge \neg\textsc{NoFault}_m]$.

If the event $\neg\textsc{Unbind}$ happens, then there exists a unique vector $\mathbf{z}$ that the adversary maximally committed to the Merkle root value $\tilde{\phi}_\ell$. Moreover, as the event $\textsc{Guess}$ does not happen, the adversary already committed the vector $\mathbf{z}$ to the Merkle root value $\tilde{\phi}_\ell$ before querying $\mathcal{H}_{\mathsf{chal}}(\tilde{id}, \tilde{\phi}_\ell)$. Since $\textsc{NoFault}_m$ does not happen, $\mathbf{z}$ is an $\mathcal{H}_{\tilde{id}}$-labeling (of the graph G) with more than m faults, i.e., there are at least m *false nodes* that will fail the graph consistency check. Note that the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$ outputs 1 only if the uniformly chosen set check checks no false node, therefore we have

$$\Pr\left[\mathrm{E}^*_i \wedge \mathrm{E}_1 \wedge \neg\textsc{NoFault}_m\right] \leq \left(1-\frac{m}{N}\right)^{\tau} \leq \exp\left(-\frac{\tau\cdot m}{N}\right).$$

Finally, we note that the adversary must have queried $(\tilde{id}, \tilde{\phi}_\ell)$ to $\mathcal{H}_{\mathsf{chal}}$ as $\textsc{Guess}$ does not happen. Since there are at most $Q_{\mathsf{B}}+Q$ random oracle queries, the claim holds by union bound. □

Since

$$\Pr\left[\mathrm{E}_1\right] \leq \Pr\left[\mathrm{E}_1 \wedge \neg\textsc{NoFault}_m\right] + \Pr\left[\mathrm{E}_1 \wedge \textsc{NoFault}_m\right],$$

to bound $\Pr[\mathrm{E}_1]$, it is now sufficient to bound the probability of the event $\mathrm{E}_2 := \mathrm{E}_1 \wedge \textsc{NoFault}_m$. Intuitively, E_2 implies that the adversary A generates a valid proof that is not extractable by the knowledge extractor. Moreover, the proof is bound with a unique graph labeling that has *no more than m faults.*

Step 4: To bound $\Pr[\mathrm{E}_2]$, we first show a useful claim that if the adversary wins the game, the outside adversary B should not have known the challenge set $\mathcal{H}_{\mathsf{chal}}(\tilde{id}, \tilde{\phi}_\ell)$. We define an event called $\textsc{OpenChal}$, meaning that the challenge sets were computed by the outside adversary B.

Event OpenChal: The RO input-output pair $((\tilde{id}, \tilde{\phi}_\ell), \mathcal{H}_{\mathsf{chal}}(\tilde{id}, \tilde{\phi}_\ell))$ is queried by the outside adversary B (i.e., $((\tilde{id}, \tilde{\phi}_\ell), \mathcal{H}_{\mathsf{chal}}(\tilde{id}, \tilde{\phi}_\ell)) \in \mathcal{Q}_{\mathcal{H}}(\mathsf{B})$), where $\tilde{id}$ is the output identity, and $\tilde{\phi}_\ell$ is the Merkle root value in the output proof-of-space $\tilde{\pi}$.

Claim 9. *Denote by event* $\mathrm{E}_1 := \neg\textsc{Unbind} \wedge \neg\textsc{Guess} \wedge (\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda) = 1)$*, it holds that*

$$\Pr\left[\mathrm{E}_1 \wedge \textsc{OpenChal}\right] = 0\,,$$

where the probability is over the choice of the random oracle, and the coin tosses of $\mathsf{Setup}, \mathsf{B}$.

Proof. If the event $\neg\textsc{Guess} \wedge \textsc{OpenChal}$ happens, the adversary must have 1) followed the topological order when computing the Merkle commitment; 2) computed the Merkle root value $\tilde{\phi}_\ell \in \tilde{\pi}$ herself. On the other hand, since Unbind does not happen, the $\tau \cdot (\mathsf{deg}+1) + \tau_c$ opening paths in proof $\tilde{\pi}$ must have been computed by B already, as otherwise, if there is a new opening path in $\tilde{\pi}$, the binding property is violated and an RO collision pair would appear. Therefore we have $\mathcal{Q}_{\mathcal{H}}(\tilde{\pi}) \subseteq \mathcal{Q}_{\mathcal{H}}(\mathsf{B})$. By the winning condition of the game, $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$ outputs 0, thus the claim holds.

□

Note that Claim 9 implies that OpenChal will never happen if the event $\mathrm{E}_2 := \mathrm{E}_1 \wedge \textsc{NoFault}_m$ occurs. Thus we can bound $\Pr[\mathrm{E}_2]$ by analyzing a simpler labeling game where the challenge set is uniformly random.

The labeling challenge game. At the beginning of the labeling game, an outside adversary B outputs a graph labeling ℓ (with m faults) and a space-time bounded algorithm A. Then a set of random challenge nodes is revealed, and the game wins if the algorithm A can recover the labeling of the random challenge nodes.

Let parameters $n_{\mathcal{H}}, k_{\mathsf{pos}}, s, f, t, Q, \tau_c \in \mathbb{N}$, graph G (plus the corresponding target sets), and random oracle $\mathcal{H}$ be the same as in the proof-of-space scheme Π_G. Denote by $s^* := s + n_{\mathcal{H}}$ and $f^* := f + \log Q$. For any PPT adversary B (in the restricted storage model), and any $m \in \mathbb{N}$, we define the labeling challenge game $\mathbf{G}^{\mathsf{lab}}_{\mathsf{B},s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$ as follows.

Game $\mathbf{G}^{\mathsf{lab}}_{\mathsf{B},s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$:

1. Let $(\sigma, \mathsf{A}, id^*, \ell) \leftarrow \mathsf{B}^{\mathcal{H}}$, where σ is a memory state with no more than s^* bits, A is a (s^*, f^*, t)-bounded deterministic algorithm (that is in the restricted storage model and makes at most Q RO queries), $id^* \in \{0,1\}^{k_{\mathsf{pos}}}$ is an identity, and ℓ is an $\mathcal{H}_{id^*}$-labeling (of graph G) with no more than m faults.
2. Let $\mathsf{chal} \leftarrow\!\!\$\ V_c^{(1)} \times \cdots \times V_c^{(\tau_c)}$ be a uniformly chosen challenge set.
3. Let $(\{z_v\}_{v \in \mathsf{chal}}) \leftarrow \mathsf{A}^{\mathcal{H}}(\sigma, \mathsf{chal})$.
4. Output 1 if and only if for every $v \in \mathsf{chal}$, it holds that $z_v = \ell_v$ (where ℓ_v is the vth element of labeling ℓ).

We define the advantage of $\mathbf{G}^{\mathsf{lab}}_{\mathsf{B},s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$ as

$$\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B},s^*,f^*,t,Q,\tau_c,m,G}(\lambda) := \Pr\left[\mathbf{G}^{\mathsf{lab}}_{\mathsf{B},s^*,f^*,t,Q,\tau_c,m,G}(\lambda) = 1\right]\,,$$

where the probability is over the choice of the random oracle, and the coin tosses of $\mathsf{chal}, \mathsf{B}$.

Next we show that $\Pr[\mathrm{E}_2]$ can be bounded by the advantage of labeling challenge game.

Claim 10. *Fix parameters $n_{\mathcal{H}}, k_{\mathsf{pos}}, s, f, t, Q, \tau_c \in \mathbb{N}$, graph G, PPT adversary B (in the restricted storage model) and identity id in the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$. For any $m \in \mathbb{N}$, we define event*

$$\mathrm{E}_2 := \neg\textsc{Guess} \wedge (\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda) = 1) \wedge \neg\textsc{Unbind} \wedge \textsc{NoFault}_m \,.$$

Let $s^* := s + n_{\mathcal{H}}$ and $f^* := f + \log Q$. There exists a PPT adversary B' (in the restricted storage model) such that

$$\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \geq \Pr\left[\mathrm{E}_2\right]/Q \,.$$

Proof. Intuitively, from the adversary B, we build the adversary B' by programming an RO query output with the random challenge set, and hope that B will output a proof-of-space that contains the correct labels for the challenge set.

More formally, the adversary B' (in $\mathbf{G}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$) determines the output $(\sigma, \mathsf{A}, id^*, \ell)$ as follows.

1. The initial state σ is $(id, \pi := \mathsf{P}^{\mathcal{H}}_{\mathsf{pp}_{\mathsf{pos}}}(id))$, where id is the identity in the game $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$.
2. B' samples a uniformly random index $q \leftarrow\!\!\$\, [Q]$.
3. Next, B' determines the identity id^* and the labeling ℓ as follows.
   - With access to the random oracle $\mathcal{H}$, B' simulates the execution $\mathsf{B}^{\mathcal{H}}$ until B outputs a tampering algorithm A. Let $\mathcal{Q}_{\mathcal{H}}(\mathsf{B})$ be the query table of B.
   - B' runs $\mathsf{A}^{\mathcal{H}}(\sigma := (id, \pi))$ until A made q RO queries to $\mathcal{H}_{\mathsf{chal}}$. If A terminates before making q RO queries to $\mathcal{H}_{\mathsf{chal}}$, B' outputs $\perp$ and halts.
   - Let $\mathsf{inp}_q := (id^*, \phi^*_\ell)$ be the qth query input to $\mathcal{H}_{\mathsf{chal}}$, and let $\mathcal{Q}_{\mathcal{H}}(\mathsf{A})$ be the query table of A. B' uses $\mathcal{Q}_{\mathcal{H}}(\mathsf{B}) \cup \mathcal{Q}_{\mathcal{H}}(\mathsf{A})$ to compute ℓ – the sequence of labels underlying the merkle commitment ϕ^*_ℓ. If B' finds more than one vector that maximally commits to the value ϕ^*_ℓ (i.e., the binding property is violated), B' outputs $\perp$ and halts.
   - If ℓ is an $\mathcal{H}_{id^*}$-labeling with more than m faults, B' outputs $\perp$ and halts.
4. Then from the tampering algorithm A, the adversary B' builds an algorithm A' that hardwires the index q. Given inputs a state σ and a challenge set chal, $\mathsf{A}'^{\mathcal{H}}(\sigma, \mathsf{chal})$ runs $\mathsf{A}^{\mathcal{H}}(\sigma)$. After A made the qth RO query to $\mathcal{H}_{\mathsf{chal}}$ (with input $\mathsf{inp}_q := (id^*, \phi^*_\ell)$), instead of directly replying with $\mathcal{H}_{\mathsf{chal}}(\mathsf{inp}_q)$, A' programs the RO output so that it is consistent with the input challenge set chal. Then A' continues running A. Denote by $(\tilde{id}, \tilde{\pi})$ the output of $\mathsf{A}^{\mathcal{H}}(\sigma)$. A' extracts the label values $\{z_v\}_{v \in \mathsf{chal}}$ from $\tilde{\pi}$, and outputs the labeling.
5. Finally, the adversary B' outputs $(\sigma, \mathsf{A}', id^*, \ell)$.

Note that the algorithm A' is (s^*, f^*, t)-bounded since A' requires only the extra memory space to store the challenge set chal, and the hardwired index q has no more than $\log Q$ bits.

Next we argue that $\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \geq \Pr\left[\mathrm{E}_2\right]/Q$. Consider the execution of $\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda)$. Given the randomly chosen $q \leftarrow\!\!\$\, [Q]$, we define a good event E^*_q.

> Event E^*_q: A's qth query to $\mathcal{H}_{\mathsf{chal}}$ (with input (id^*, ϕ^*_ℓ)) is the first $\mathcal{H}_{\mathsf{chal}}$'s query (by B or A) that satisfies $id^* = \tilde{id}$ and $\phi^*_\ell = \tilde{\phi}_\ell$, where $\tilde{id}$ is the ouptut identity by A, and $\tilde{\phi}_\ell$ is the merkle commitment in the output proof-of-space by A.

Define BAD as the event that in the execution $\mathsf{A}'(\sigma, \mathsf{chal})$, after A made the qth queries to $\mathcal{H}_{\mathsf{chal}}$, the query input $\mathsf{inp} := (id^*, \phi^*_\ell)$ was queried by B (or A) before. Note that B' wins the labeling game if $\mathrm{E}^*_q \wedge \mathrm{E}_2$ happens and BAD does not happen, hence

$$\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \geq \Pr\left[\mathrm{E}^*_q \wedge \mathrm{E}_2 \wedge \neg \mathrm{BAD}\right] . \tag{5}$$

On the other hand, if BAD happens, then the event E^*_q does not happen. Thus $\Pr[\mathrm{E}^*_q \wedge \mathrm{E}_2 \wedge \mathrm{BAD}] = 0$, and we have

$$\Pr\left[\mathrm{E}^*_q \wedge \mathrm{E}_2\right] = \Pr\left[\mathrm{E}^*_q \wedge \mathrm{E}_2 \wedge \neg \mathrm{BAD}\right] . \tag{6}$$

Combining Inequality 5 and Inequality 6, we have

$$\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \geq \Pr\left[\mathrm{E}^*_q \wedge \mathrm{E}_2\right] .$$

Next, we show that

$$\Pr\left[\mathrm{E}^*_q \wedge \mathrm{E}_2\right] \geq \Pr\left[\mathrm{E}_2\right]/Q .$$

By Claim 9, E_2 implies that OPENCHAL does not happen (i.e., $((\tilde{id}, \tilde{\phi}_\ell), \mathcal{H}_{\mathsf{chal}}(\tilde{id}, \tilde{\phi}_\ell)) \notin \mathcal{Q}_{\mathcal{H}}(\mathsf{B})$), thus $(\tilde{id}, \tilde{\phi}_\ell)$ can only match to some RO query of A. Since $q \in [Q]$ is chosen independetly and uniformly, conditioned on E_2, it holds that with probability at least $1/Q$, the qth $\mathcal{H}_{\mathsf{chal}}$ query (made by A) is the first $\mathcal{H}_{\mathsf{chal}}$ query that has input $(\tilde{id}, \tilde{\phi}_\ell)$. Hence we have $\Pr[\mathrm{E}^*_q \wedge \mathrm{E}_2] \geq \Pr[\mathrm{E}_2]/Q$.

In summary, we have

$$\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \geq \Pr\left[\mathrm{E}^*_q \wedge \mathrm{E}_2\right] \geq \Pr\left[\mathrm{E}_2\right]/Q ,$$

and the claim holds.

□

Since an upper bound of $\Pr[\mathrm{E}_2]$ can be obtained from the advantage of labeling challenge game, to bound $\Pr[\mathrm{E}_2]$, it is now sufficient to analyze the advantage of the labeling challenge game.

Step 5: Finally, we exploit challenge-hard graphs to bound the advantage of the labeling challenge game. The idea is to generically transform any (sequential) labeling algorithm that wins the labeling challenge game with probability p, into a (sequential) pebbling strategy that wins the pebbling challenge game with probability at least p.

Claim 11. *Fix parameters $n_{\mathcal{H}}, k_{\mathsf{pos}}, s^*, f^*, t, Q, \tau_c, m \in \mathbb{N}$, graph G (with N vertices) and the corresponding target sets. Let B' be any PPT algorithm (in the restricted storage model). Define*

$$\beta^* := \frac{s^* + f^* + m \cdot n_{\mathcal{H}}}{N_c \cdot n_{\mathcal{H}}} .$$

There exists a pebbling strategy B'', such that

$$\mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B}'',\beta^*,t,\tau_c,G}(\lambda) \geq \mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) ,$$

where $\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$ is the advantage of the labeling challenge game $\mathbf{G}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$, and $\mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B}'',\beta^*,t,\tau_c,G}(\lambda)$ is the advantage of the pebbling challenge game $\mathbf{G}^{\mathsf{peb}}_{\mathsf{B}'',\beta^*,t,\tau_c,G}(\lambda)$ (see Definition 14).

Proof. Suppose B' outputs $(\sigma, \mathsf{A}, id^*, \ell)$ in the labeling game $\mathbf{G}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$. We determine the pebbling strategy accordingly as follows. Given id^* and ℓ, let $F \subseteq V$ be the set of false nodes (i.e., the set of nodes whose labels are inconsistent with the graph). For every possible challenge set chal, we denote by $\mathsf{Tr}(\mathsf{A}(\sigma, \mathsf{chal}))$ the transcript of $\mathsf{A}(\sigma, \mathsf{chal})$, and $P^*(\mathsf{chal})$ the set of necessary nodes at step 0 in the ex-post-facto pebbling of $\mathsf{Tr}(\mathsf{A}(\sigma, \mathsf{chal}))$. We define

$$P^* := \bigcup_{\mathsf{chal} \in V_c^{(1)} \times \cdots \times V_c^{(\tau_c)}} P^*(\mathsf{chal})\,,$$

and define the initial pebbled set as $P_0 := P^* \cup F$.

We show that $|P_0| \le \beta^* \cdot N_c$: First, $|F| \le m$ as ℓ is an $\mathcal{H}_{id^*}$-labeling with at most m faults. Second, since we assume that A never compresses or guesses graph labels, by definition of ex-post-facto pebbling, it holds that $|P^*| \le (s^* + f^*)/n_{\mathcal{H}}$. Therefore we have

$$|P_0| \le m + \frac{s^* + f^*}{n_{\mathcal{H}}} = \beta^* \cdot N_c\,.$$

Next, in the pebbling game, given a uniformly chosen challenge set chal, we determines a *sequential* pebbling strategy as follows. Let $\mathsf{Tr}(\mathsf{A}(\sigma, \mathsf{chal}))$ be the transcript of $\mathsf{A}(\sigma, \mathsf{chal})$, and $\mathbf{P}$ be the ex-post-facto pebbling of $\mathsf{Tr}(\mathsf{A}(\sigma, \mathsf{chal}))$. We define the pebbling strategy $\mathbf{P}^* := (P_0, \ldots, P_t)$ as follows: The initial pebbled set is the set $P_0 := P^* \cup F$ defined before. For every step i $(1 \le i \le t)$, let P'_i be the pebbling configuration in $\mathbf{P}$ (at step i), we define the pebbling configuration as $P_i := F \cup P'_i$.

Since $|F| \le m$ and A is a (s^*, f^*, t)-bounded sequential algorithm in the restricted model, it holds that $\mathbf{P}^*$ is a strategy that i) follows the rule of a sequential pebbling, ii) runs at most t steps, and iii) uses at most $m + (s^* + f^*)/n_{\mathcal{H}} = \beta^* \cdot N_c$ pebbles.

Finally, we note that the winning probability of the transformed pebbling strategy is at least $\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda)$, as $\mathbf{P}^*$ pebbles all the challenge vertices if and only $\mathsf{A}(\sigma, \mathsf{chal})$ answers all the challenge labels. And thus the claim holds.

□

Recall that the graph G is $(\beta, N_c, N, \tau_c, t, \epsilon_{\mathsf{peb}})$-challenge hard, and

$$s + f \le (\beta - \delta^* - 0.01) \cdot N_c \cdot n_{\mathcal{H}}\,.$$

Thus for $N_c \ge 200$, we can set $m := \delta^* \cdot N_c$, $s^* := s + n_{\mathcal{H}}$, $f^* := f + \log Q$ so that

$$\beta^* := \frac{s^* + f^* + m \cdot n_{\mathcal{H}}}{N_c \cdot n_{\mathcal{H}}} \le \frac{s + f + m \cdot n_{\mathcal{H}} + 2n_{\mathcal{H}}}{N_c \cdot n_{\mathcal{H}}} = \beta - 0.01 + 2/N_c \le \beta\,.$$

By Claim 11 and by $(\beta, N_c, N, \tau_c, t, \epsilon_{\mathsf{peb}})$-challenge-hardness of G, it holds that

$$\mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \le \mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B}'',\beta^*,t,\tau_c,G}(\lambda) \le \mathsf{Adv}^{\mathsf{peb}}_{\mathsf{B}'',\beta,t,\tau_c,G}(\lambda) \le \epsilon_{\mathsf{peb}}\,.$$

Thus, by Claim 10, we have that

$$\Pr[\mathrm{E}_2] \le Q \cdot \mathsf{Adv}^{\mathsf{lab}}_{\mathsf{B}',s^*,f^*,t,Q,\tau_c,m,G}(\lambda) \le Q \cdot \epsilon_{\mathsf{peb}}\,.$$

Wrap-up: In summary, let $m := \delta^* \cdot N_c$ and $\kappa := \tau \cdot m/N$, define event $\mathrm{E}_1 := (\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda) = 1) \wedge \neg\textsc{Guess} \wedge \neg\textsc{Unbind}$ and event $\mathrm{E}_2 := \mathrm{E}_1 \wedge \textsc{NoFault}_m$, we have

$$\begin{aligned}\Pr\left[\mathbf{G}^{\mathsf{pext}}_{\mathsf{B},id}(\lambda) = 1\right] &\le \Pr[\textsc{Guess}] + \Pr[\textsc{Unbind}] + \Pr[\mathrm{E}_1 \wedge \neg\textsc{NoFault}_m] + \Pr[\mathrm{E}_2] \\ &\le \frac{\mathsf{deg} \cdot (Q_{\mathsf{B}} + Q) \cdot (Q_{\mathsf{B}} + Q + \nu)}{2^{n_{\mathcal{H}}}} + \frac{(Q_{\mathsf{B}} + Q)^2}{2^{n_{\mathcal{H}}}} \\ &\quad + (Q_{\mathsf{B}} + Q) \cdot \exp(-\kappa) + Q \cdot \epsilon_{\mathsf{peb}}\,,\end{aligned} \tag{7}$$

and Theorem 2 holds because $Q, Q_{\mathsf{B}} \in \mathrm{poly}(\lambda)$.

Remark 13. *In the general storage model where the adversary can store arbitrary function of the graph labels, Pietrzak [Pie18] presented an elegant proof that transforms any* parallel *labeling algorithm into a* parallel *pebbling strategy. However, the reduction proof does not extend to the sequential setting, because even if the original labeling strategy is* sequential*, the transformed strategy is still a* parallel *pebbling strategy. A possible approach to avoid the issue seems to construct a graph that is challenge hard against parallel pebbling strategies. We found that Pietrzak [Pie18] indeed defined a similar notion of challenge hard graphs for parallel pebbling strategies. Unfortunately, the graph that they constructed only achieves a time bound that is linear to the graph size, which is too weak to fit into our setting where a tampering algorithm typically has polynomial running time.*