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

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

[1] University of California, Santa Barbara, USA
binyichen@cs.ucsb.edu
[2] VISA Research, USA
{yilchen,pratmukh}@visa.com
[3] Technische Universität Darmstadt, Germany
kristina.hostakova@cs.tu-darmstadt.de

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

* Work done at VISA Research

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.

1 Introduction

Non-malleable codes and tamper-resilience. The notion of non-malleable codes (NMC) was put forward by Dziembowski, Pietrzak and Wichs [20] as an abstract tool for protecting cryptographic devices against tampering attacks (e.g. [12]). 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 [32,24,1,16,25,29,2,15]) 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. [22] (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. [22] 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 [23] for more details).

Continuous space-bounded tampering. Traditional NMC (as defined in [20]) guarantees non-malleability when the attacker tampers only once. To use such NMC for tamper-resilience (see [20] for more details), one needs to refresh the encoding after each tampering. To combat this issue, in 2014, Faust et al. [24] 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 [23]) 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 [23]).

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. $\log(\theta)$.[4] 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. [24]). 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) 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.

[4] 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 $\ell \propto \log(\theta)$.

Proof-extractability from any NIPoS with uniqueness. Our starting point is the construction of FHMV [22] 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 $|c|$ is the codeword length and p is the available (persistent) space.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 [38]) 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 they formalize the notion of space. In FHMV, which in turn follows the notion introduced by Dziembowski et al. [19], 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 [23]). 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 [38] 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. [37] and used by Dziembowski et al. [17], 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. For example, for a message of size 1 MB, the size of a codeword is almost 800 MB for the CHG-based NMC constructions. 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 [38], 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 kilobytes), 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.Note that any MHF, while used in practice with concrete hash functions (for example SHA3) for important practical applications [39], provides provable guarantees only in the random oracle model (see, e.g. [6]). Instead, we 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 [22]. Overall we propose *four* different constructions of different merits; we provide a comparison in Table 1:

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.

– 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 [22]. 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 [7,17]. 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 [27] 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

[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.

$(id', \pi_{id'})$. Furthermore, the simulator also needs an extra leakage, which consists of the "extra persistent space" (of size $p - |c|$)— now the simulator reconstructs the entire persistent tampered state and can continue the rest of 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 $\ell \propto \log(\theta) + \eta' + (p - |c|)$ 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 1) 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 [36] 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. [38]), 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. [32,28,2,1,15,31,30,13] falls into the the later category; among them [13] considers, a weaker notion (non-malleability with replacement) of NMC like us (leaky-NMC). A few other works, e.g. [26,5,9,10] consider global tampering. Moreover, most of these work consider one-time tampering. Continuous tampering, first proposed in [24], is addressed also in [21,3,35,4,25]. Except FHMV [22], the recent work by Ball et al. [10] 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 [19,18] for constructing different schemes. For more detail on different NMC-based compilers for tamper-resilience we refer to [34].

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 $\mu(\lambda) \in O(\lambda^c)$.

We defer a few basic definitions to the full version [14].

3.2 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 [22] 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. [22] only assumes concrete measure of the latter 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 1 (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 $f_\mathsf{A} \leq f, s_\mathsf{A} \leq s$ and $t_\mathsf{A} \leq t$ (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 [18,19], 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. A formal definition is deferred to full version [14].

4 Continuous Space-bounded Tampering

Space-bounded Tampering algorithms. We assume that tampering algorithms are deterministic[6], 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 $\mathcal{A}_{\text{space}}$ 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 [22] (Definition 5) our tampering oracle has the "self-destruct" mechanism.

Definition 2 (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* $\mathtt{st} := (c, \sigma)$*, where* $c := \mathsf{Encode}^{\mathcal{H}}(\mathsf{pp}, x)$*, 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:*

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 $\tilde{x}$*.*

We recall from [22] 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 3 (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 4 (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.*

[6] 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.

Oracle $\mathcal{O}_{\text{sim}}^{\mathsf{S}_2,\ell,x,s,f,\mathsf{pp}}(\mathsf{A})$*:*

Let $\tilde{x} \leftarrow \mathsf{S}_2^{\mathcal{O}_{\text{leak}}^{\ell,x}(\cdot)}(1^\lambda, \mathsf{pp}, \mathsf{A})$
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 [22] 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 5 (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*[7] $(\theta, \varepsilon_{\mathsf{nm}})$*-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:*

- **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* $\lambda \in \mathbb{N}$*,*

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

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

We are interested in constructing an encoding scheme which satisfies Definition 5 with any choice of $\theta = \text{poly}(\lambda)$. Recall from Section 3.2 of [22] 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[8] leakage ℓ.

5 Non-Interactive Proof of Space (NIPoS)

As in [22], 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.

[7] 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 1.)

[8] 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.

We start by recalling the definition of NIPoS from [22] 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 6 (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.*

- $\mathsf{Setup}^{\mathcal{H}}(1^{\lambda})$*: 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 7 (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.*

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 8 (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*.*[9] *The probability is over the choice of the random oracle, and the coin tosses of* $\mathsf{Setup}, \mathsf{B}$*.*

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

Remark 1. 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 7. 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 latter 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.

Definition 9 (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* $J(\mathsf{pp}_{\mathsf{pos}}, id) = \pi[1 \dots u_{\mathsf{pos}}]$ *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 2. 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 1. *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*

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

The proof is deferred to the full version [14].

6 Space-bounded NMC from Proof-Extractable NIPoS

The following theorem, which is proven in the full version [14], 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, (defined formally in full version [14]) 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*

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

The above theorem together with Lemma 1 imply that the encoding scheme of Faust et al. satisfies Definition 5 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*

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

7 Constructing Proof-Extractable NIPoS from CHG

7.1 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. We remark that the notion of challenge hardness has similarities with a notion introduced in [17]. In particular, in Section 4 of [17], the authors informally described a pebbling game which is similar to the game in our notion (Definition 10).

Challenge hard graphs are parameterized by the following variables: N_c, β, N, τ_c, t, ε, 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 10 (Challenge Hard Graphs (CHG)). *A family of directed acyclic graphs $\{G_\lambda\}_{\lambda \in \mathbb{N}}$ (with constant in-degree)*[10] *is $(\beta, N_c, N, \tau_c, t, \varepsilon)$*-challenge-hard

[10] 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 [11] for more discussions.

(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.

7.2 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 [22] 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 $V_c^{(1)}, \ldots, V_c^{(\tau_c)} \subseteq V$ 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.[11] Note that by a typical domain separation technique (e.g., used in [7] and [33]), 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 1. 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.

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 $\tau \cdot (\mathsf{deg}+1) + \tau_c$ 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*.

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 $\mathsf{deg} = O(1)$, let $\gamma_{\mathsf{sp}} = N_c/\tau_c$ and $\gamma_{\mathsf{cp}} = N/N_c$ denote the challenge sparseness and graph compactness of G. $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{n_{\mathcal{H}}}$ 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 1) built upon G, where*

$$\nu := (\tau \cdot (\mathsf{deg}+1) + \tau_c) \cdot \log N + 1 \approx N_c \log N/\gamma_{\mathsf{sp}} .$$

[11] 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.

Fig. 1: 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.

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[12]

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

where $\kappa = \tau \cdot \delta^* \cdot N_c/N = \tau \cdot \delta^*/\gamma_{\mathsf{cp}}$.

Remark 3. To guarantee that the verifier space $s_{\mathsf{V}} \approx \nu \cdot n_{\mathcal{H}}$ is smaller than the tampering space $s \approx N_c \cdot n_{\mathcal{H}}$, we need the underlying CHG to be **at least** $\Omega(\log N)$-challenge sparse (defined as N_c/τ_c).

7.3 Instantiating CHG

We propose two instantiations (details deferred to full version [14]). First, we provide a new construction of CHG from Stack of Localized Expander Graphs (SoLEG) (see the full version [14] for details on SoLEGs) used by Ren and Devadas [38] in the context of proof-of-space. We use a novel technique to construct an **extension of SoLEG** by connecting a gadget in order to "boot-strap" challenge sparseness. Second, as observed by [17] (in Section 6.1 of [17]), the graph introduced by Paul, Tarjan and Celoni [37] (in short, **PTC's graph**) does satisfy challenge hardness.

7.4 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 and the PTC's graph respectively into Theorem 2. The details of the concrete instantiations and the comparison of the two PExt-NIPoS constructions are deferred to full version [14].

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.

8.1 Memory-hard Functions

Here we formalize memory-hard functions. The definition of our second building block, publicly verifiable computation, can be found in full version [14].

Definition 11 (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:*

[12] The polynomial factor in $\varepsilon_{\mathsf{p\text{-}ext}}$ depends on the number of RO queries made by the adversary. We refer to full version [14] for the exact probability upper bound.

- *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 4. 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.

8.2 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 $\mathsf{Ver}_{vk_M}(x, y, \pi_{\mathsf{vc}})$.

Lemma 2. *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.

8.3 Instantiating MHF

Our MHF instantiation is a slight variant of a graph-based proof of space construction;[13] in particular, we choose the one provided in [38] (also used in [22]). 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 11) 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 full version [14].)

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* $\gamma_{\mathsf{hs}} \in (0, \frac{1}{2}]$.

[13] Since popular memory-hard functions like SCrypt [39] are not conjectured to provide exponential space-time trade-off, we are unable to use them here.

We defer some important notes on the above assumptions to full version [14].

From Assumption 2 we get the following corollary about our MHF-candidate:[14]

Corollary 2. *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:*

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

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:*

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

8.4 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 [36].

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 [36].

Assumption 3 (Space-bounded Verification) *Let* $\mathbb{G}$ *be a (as considered in [36]) 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.*[15] *Then for a function* $F : \{0,1\}^k \to \{0,1\}^n$*, the Pinocchio verification algorithm (see Protocol 2 of [36]) requires* $k+n+O(\lambda)$*-bit space asymptotically and* $\approx k + n + 300 \cdot \lceil \log p \rceil$ *bits concretely.*

[14] We remark that this corollary is very similar to Corollary 1 of [23] as one may expect. However the parameters here are much better in terms of efficiency.

[15] To achieve 128-bits of security, as suggested by [8], we will set $\lceil \log(p) \rceil \approx 1536$.

We defer some important notes on the above assumption to full version [14].

Assumption 4 (Lower-bound on Prover's space) *Let $\mathbb{G}$ be a (as considered in [36]) 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 [36]) requires at least s_F-bit space.*

Combining Assumption 3 and Assumption 4 we conclude:

Corollary 3. *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 $(s_F, \mathrm{poly}(\lambda), s_{\mathsf{V}}^{vc}, \mathrm{poly}(\lambda), s_{\mathsf{P}}^{vc}, \mathrm{poly}(\lambda), k, n, n_{\mathsf{vc}}, \mathrm{negl}(\lambda))$-non-interactive publicly verifiable computation construction, where:*

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

Furthermore, in concrete terms, to get $\varepsilon_{\mathsf{vc}} \approx 2^{-128}$, choosing $\lceil \log p \rceil \approx 1536$ (following [8]) we can have estimations of the verifier's space $s_{\mathsf{V}}^{vc} \approx 58$ KB$+k+n$ and the proof-size $n_{\mathsf{vc}} \approx 3$ KB.

8.5 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 2 and Corollary 3 into Lemma 2, we obtain the following corollary:

Corollary 4 (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:*

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

9 Instantiating and comparing our NMC constructions

9.1 Instantiations from different PExt-NIPoS

We obtain four constructions in total, two of them through CHG and other two (including FHMV one) through the uniqueness. For more details we refer to the full version [14].

9.2 Comparing concrete parameters

We propose four constructions of space-bounded (leaky) non-malleable codes that support unbounded tampering. All of them are based on NIPoS. Two of them (described in Section 7) require proof-extractability whereas other two can be based on standard extractability (described in Section 6 and Section 8). Our constructions have different merits. While the asymptotic bounds for the parameters have already been provided, we believe that a comparison with respect to the concrete values is important.

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 5 — 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: $\theta = 2^{16}$.[16] 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 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 (see the full version [14] for detail) and plugging-in them to Theorem 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.

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. 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

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

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 of the above constructions we need a PRF with standard security as the proofs depend on the pseudorandomness guarantee of the PRF.

References

1. 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.
2. 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.
3. 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`.
4. 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.
5. 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.
6. 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.
7. 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.
8. Aurore Guillevic and François Morain. Discrete logarithms. Book Chapter 9. `https://hal.inria.fr/hal-01420485v1/document`.
9. 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.
10. 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.
11. 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`.
12. 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.
13. 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.
14. Binyi Chen, Yilei Chen, Kristina Hostáková, and Pratyay Mukherjee. Continuous space-bounded non-malleable codes from stronger proofs-of-space. Cryptology ePrint Archive, Report 2019/552, 2019. `https://eprint.iacr.org/2019/552`.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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`.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.
29. 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.
30. 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.
31. 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.
32. 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.
33. 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.
34. Mukherjee, Pratyay. *Protecting Cryptographic Memory against Tampering Attack*. PhD thesis, 2015.
35. 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.
36. 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.
37. 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.
38. 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.
39. Tarsnap. The scrypt key derivation function. Webpage. `https://eprint.iacr.org/2017/1125`.