Block-wise Non-malleable Codes*

Nishanth Chandran†1, Vipul Goyal‡1, Pratyay Mukherjee§2, Omkant Pandey¶3, and Jalaj Upadhyay‖4

[1] *Microsoft Research India*
[2] *University of California, Berkeley, USA*
[3] *Drexel University, USA*
[4] *Pennsylvania State University, USA*

April 26, 2016

Abstract

Non-malleable codes, introduced by Dziembowski, Pietrzak, and Wichs (ICS '10) provide the guarantee that if a codeword c of a message m, is modified by a tampering function f to c', then c' either decodes to m or to "something unrelated" to m. It is known that non-malleable codes cannot exist for the class of all tampering functions and hence a lot of work has focused on explicitly constructing such codes against a large and natural class of tampering functions. One such popular, but restricted, class is the so-called *split-state* model in which the tampering function operates on different parts of the codeword *independently*.

In this work, we consider a stronger adversarial model called *block-wise tampering* model, in which we allow tampering to depend on more than one block: if a codeword consists of two blocks $c = (c_1, c_2)$, then the first tampering function f_1 could produce a tampered part $c_1' = f_1(c_1)$ and the second tampering function f_2 could produce $c_2' = f_2(c_1, c_2)$ depending on *both* c_2 and c_1. The notion similarly extends to multiple blocks where tampering of block c_i could happen with the knowledge of all c_j for $j \leq i$. We argue this is a natural notion where, for example, the blocks are sent one by one and the adversary must send the tampered block before it gets the next block.

A little thought reveals however that one cannot construct such codes that are non-malleable (in the standard sense) against such a powerful adversary: indeed, upon receiving the last block, an adversary could decode the entire codeword and then can tamper depending on the message.

---

*An extended abstract of this paper is published in the proceedings of the 43rd Inter International Colloquium on Automata, Languages, and Programming — ICALP 2016. This is the full version.

†E-mail: `nichandr@microsoft.com`

‡E-mail: `vipul@microsoft.com`

§Research supported in part from a DARPA/ARL SAFEWARE award, AFOSR Award FA9550-15-1-0274, and NSF CRII Award 1464397. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government. Part of this work was when this author was a PhD student at Aarhus University supported by supported by a European Research Commission Starting Grant (no. 279447), the CTIC and CFEM research center (under the Sino-Danish grant no. 61061130540) and while visiting Microsoft Research India. E-mail: `pratyay85@gmail.com`

¶Work done in part while visiting Microsoft Research India. E-mail: `omkant@gmail.com`

‖Work done in part while visiting Microsoft Research India. E-mail: `jalaj@psu.edu`

In light of this impossibility, we consider a natural relaxation called *non-malleable codes with replacement* which requires the adversary to produce not only related but also a valid codeword in order to succeed. Unfortunately, we show that even this relaxed definition is not achievable in the information-theoretic setting (i.e., when the tampering functions can be unbounded) which implies that we must turn our attention towards computationally bounded adversaries.

As our main result, we show how to construct a block-wise non-malleable code from sub-exponentially hard one-way permutations. We provide an interesting connection between block-wise non-malleable codes and non-malleable commitments. We show that any block-wise non-malleable code can be converted into a non-malleable (w.r.t. opening) commitment scheme. Our techniques, quite surprisingly, give rise to a non-malleable commitment scheme (secure against so-called synchronizing adversaries), in which *only* the committer sends messages. We believe this result to be of independent interest. In the other direction, we show that any non-interactive non-malleable (w.r.t. opening) commitment can be used to construct a block-wise non-malleable code only with 2 blocks. Unfortunately, such commitment scheme exists only under highly non-standard assumptions (adaptive one-way functions) and hence can not substitute our main construction.

Contents

1 Introduction

Non-malleable codes. Error correcting codes allow a message m to be encoded into a codeword c, such that m can always recovered even from a tampered codeword c', only if the tampering is done in a specific way. More formally, the class of tampering functions, $\mathcal{F}_{\mathsf{frac}}$, tolerated by traditional error correction codes are ones that erase or modify only a constant fraction of the codeword c. In particular, no guarantees are provided on the output of the decoding algorithm when the tampering function $f \notin \mathcal{F}_{\mathsf{frac}}$. A more relaxed notion, error detecting codes, allow the decoder to also output a special symbol $\perp$, when m is unrecoverable from c', but here too, the codes can not tolerate simple tampering functions $f \in \mathcal{F}_{\mathsf{const}}$ where $\mathcal{F}_{\mathsf{const}}$ contains all constant functions[1]. To address this shortcoming of error correction/detection codes, Dziembowski, Pietrzak, and Wichs [18], introduced a more flexible notion of *non-malleable codes* (NMC). Informally, an encoding scheme $\mathsf{Code} := (\mathsf{Enc}, \mathsf{Dec})$ is a NMC against a class of tampering functions, $\mathcal{F}$, if the following holds: the decoded message $m' = \mathsf{Dec}(c')$ is either equal to the original message m or is completely unrelated to m, when $c' = f(\mathsf{Enc}(m))$ for some $f \in \mathcal{F}$. In general, NMC cannot exist for the set of all tampering functions $\mathcal{F}_{\mathsf{all}}$. To see this, observe that a tampering function that simply runs the decode algorithm to retrieve m, and then encodes a message related to m, trivially defeats the requirement above. However, somewhat surprisingly, Dziembowski *et al.* [18] showed the (probabilistic) existence of a NMC against a function family, $\mathcal{F}_{\mathsf{almost}}$, that is only slightly smaller than the set of all functions. They also constructed an efficient NMC against the class of tampering functions, $\mathcal{F}_{\mathsf{bit}}$, that can tamper each bit of the codeword independently. NMC has found important applications in tamper-resilient cryptography [18, 30, 19, 20].

Split-state Tampering. Arguably, one of the strongest class of tampering functions for which explicit constructions of NMC are known, is in the so called *split-state model*. Informally, a split-state model with ℓ states has the following attributes: (i) the codeword is assumed to be partitioned into ℓ-disjoint blocks $(c_1, \cdots, c_\ell)$, and (ii) the class of tampering functions, $\mathcal{F}^{\ell}_{\mathsf{split}}$, consists of all the functions $(f_1, \cdots, f_\ell)$ where f_i operates *independently* on c_i[2]. Dziembowski *et al.* [18] gave a construction of a NMC against the tampering class $\mathcal{F}^2_{\mathsf{split}}$ in the random oracle model. Constructions of NMC against $\mathcal{F}^2_{\mathsf{split}}$ are now known both in the computational [30][3] and information-theoretic settings [3, 11, 17], with Chattopadhyay and Zuckerman [9] showing an explicit information-theoretic NMC against $\mathcal{F}^{10}_{\mathsf{split}}$. Recently, the work of Aggarwal *et al.* [2] showed how to construct explicit information-theoretic NMC against $\mathcal{F}^2_{\mathsf{split}}$.

Going beyond split-state: Block-wise Tampering. A severe restriction of the split-state model is that every block of the codeword can only be tampered *independently* of all other blocks. In particular f_i modifies c_i with absolutely no knowledge about c_j, for any $j \neq i$. In this work, we address this restriction by allowing modification of *each* block *depending* on more than one-block. In particular, each c_i can be modified in any arbitrary way based on the first i blocks $(c_1, \ldots, c_i)$. Such a code is called *block-wise* NMC.[4] More formally a code is called a *block-wise* NMC if it is a

[1]In particular if f always outputs some valid codeword c', then it is impossible to detect the error. For some cryptographic application like protecting against memory tampering attack this found to be too restrictive

[2]Note that the class $\mathcal{F}_{\mathsf{bit}}$ can be viewed as $\mathcal{F}^n_{\mathsf{split}}$, where n is the length of the codeword c.

[3]In the computational setting, the functions f_i are assumed to run in polynomial time.

[4]We remark that even if we call our notion *block-wise non-malleable codes* it is identical to the notion of look-ahead non-malleable codes defined in the concurrent and independent work [2, 1]. We choose to use the term block-wise as

NMC against the class of tampering functions $\mathcal{F}_{\mathsf{la}}^{\ell}$: a set of functions $(f_1, \cdots, f_\ell) \in \mathcal{F}_{\mathsf{la}}^{\ell}$ if each f_i modify c_i to some c_i' depending on the first i-blocks[5]. A natural scenario is a synchronous streaming model when the blocks are coming in one by one and the adversary on the channel sends across each modified blocks before the next block arrives.

NMC for $\mathcal{F}_{\mathsf{block}}^{\ell}$ is impossible. One can see that it is impossible to construct NMC against $\mathcal{F}_{\mathsf{block}}^{\ell}$ (for any ℓ): consider a tampering function, where the first $\ell-1$ functions, $(f_1, \ldots, f_{\ell-1})$ are identity functions and the function f_ℓ (which gets the entire codeword as input) simply decodes the message and depending on the message, keeps it the same or overwrites it to something "invalid" (i.e., the modified codeword decodes to $\perp$). Note that, in this case the distribution of the (decoding of the) tampered codeword will indeed depend on the message, thereby violating non-malleability. In particular, such a tampering attack makes the decoder output $\perp$ with a probability distribution that depends on the input message. Therefore, we seek for a natural relaxation of the traditional definition of NMC such that it is achievable for the class $\mathcal{F}_{\mathsf{block}}^{\ell}$ and at the same time sufficient for interesting applications. In particular, we show that such relaxed NMC is sufficient to construct a simple non-malleable commitment scheme in a black-box manner[6].

NMC with replacement (NMCwR). Essentially in the above attack the adversary breaks non-malleability by making the codeword "invalid". So, we take the most natural direction to relax the definition, in that the adversary is considered to be successful only if it produces some *valid and related* codeword via tampering. In particular, the adversary may selectively "destroy" a codeword depending upon the message we encode, however we show that in some sense, this the "only attack" it can perform. Intuitively the guarantee provided by such an encoding scheme is that any adversary, by tampering with some encoded data can not produce a related encoded data without destroying it. However, formalizing such intuition turns out to be non-trivial. We take inspiration from the literature of non-malleable commitment w.r.t. replacement (introduced by Goyal [23]) and formalize such a relaxation by introducing an algorithm (possibly inefficient) called *replacer* which comes into play only when the tampered codeword is invalid, and in that case it replaces the $\perp$ by "anything" of his choice. Essentially, the idea is that if the invalidity depends on the input message (like described in the above attack) then the replacer would rectify the output to remove such dependency. We call the new notion *non-malleable codes with replacement (NMCwR)*. More details and intuition about this notion is provided later.

Block-wise Non-malleable Codes (BNMC). In this paper we explore the properties, constructions and applications of NMCwR with respect to the class of block-wise tampering functions $\mathcal{F}_{\mathsf{block}}^{\ell}$. We call such code *block-wise non-malleable codes* (BNMC). Below we provide an overview of the results presented in this paper.

---

it is more appropriate in our setting. See later in this section for more discussion on [2].

[5]We also consider a stronger class of functions where the tampering can be done in *any* order. In particular f_i can modify any c_j depending on any i blocks. See later in this section and Sec. B for more detail.

[6]Notice that the traditional application to tamper-resilient cryptography does not work with the relaxed version for obvious reason.

1.1 Our results

Information theoretic impossibility. Similar to the notion of continuous non-malleable codes [20](CNMC) here also we found that any BNMC must satisfy a *uniqueness* property (a slightly different one than CNMC). For two blocks uniqueness means that there can not exists two different valid codewords of the form (c_1, c_2) and (c_1, c_2') which decodes to different messages[7]. Otherwise an attack similar to the above is possible even without making the codeword invalid: the adversary can just always tampers the first block to c_1 and depending on the message (since f_2 gets the entire codeword) tampers to one of c_2 or c_2' hence making the output distribution depend on the message. Consequently, just like CNMC, an information theoretic impossibility is evident: in that setting the functions are unbounded and therefore (for two blocks) the function f_1 can derive the unique message corresponding to c_1 by brute-force and thus break the scheme. Henceforth, in this paper we focus on constructing BNMC based on computational assumptions.

Connection to Non-malleable Commitment. Since BNMC satisfies a definition weaker (that is NMC with replacement) than the traditional NMC, it is not possible to use such a code to build a tamper-resilient compiler as described in [18, 30] for obvious reason. In fact, it is nevertheless impossible to protect a system against memory tampering attack (see [14, 22, 26] for formal expositions on such attack) against any block-wise tampering. However we are able to show connections with non-malleable commitment with respect to opening (NMCom). To the best of our knowledge this is the first attempt to bridge these two non-malleability notions[8]

1. Given an ℓ-block BNMC we can construct (in a black-box way) a simple $(\ell - 1)$ round commitment protocol which is non-malleable with respect to opening (against synchronizing adversaries) as follows: the committer sends the block c_i in the i-th round and sends the last block c_ℓ as the opening. The receiver sends only acknowledgements after receiving each message. The non-malleability essentially follows from the non-malleability of the underlying BNMC and the perfect binding follows from the uniqueness property described above. To best of our knowledge, this is the first NMCom protocol where the receiver is not required to send any message (e.g. challenge) except for acknowledgement.

2. We also show that from any non-interactive NMCom one can easily construct an BNMC even for only $\ell = 2$ blocks (i.e. optimal for $\mathcal{F}^{\ell}_{\mathsf{block}}$). Unfortunately, the only assumptions under which we know how to construct such commitments are either in the (non-tamperable) CRS model [15] or under the highly non-standard assumption of *adaptive* one-way functions [32]. Evidently this construction can not substitute our main construction which is based on much more standard assumption like sub-exponentially hard OWP.

Note that combining the above, we can conclude that when $\ell = 2$ the NMCom and BNMC are equivalent.

Constructing BNMC. As the main result we provide a construction of BNMC from a standard assumption in the plain model. Precisely, we show that, for any arbitrary constant $\varphi > 0$, how

[7] Another way of describing uniqueness is that for every valid c_1 there is a unique message which it can decode to.

[8] In [5] Agrawal et al. showed how to use NMC to construct non-malleable string-commitment from non-malleable bit-commitment. In their work, NMC is used as a tool, and, no relations are shown between non-malleable commitments and NMC. Recently another work by Goyal et al. [24] constructs round-optimal NMCom from split-state NMC, the full version of which appears after the first version of this work.

to construct a BNMC against $\mathcal{F}^{\ell}_{\mathsf{block}}$ for $\ell = O(\kappa^{2+\varphi})$ (where κ is the security parameter). The security (i.e. non-malleability) of the construction is based on "sub-exponentially" hard one-way permutations which says that there exists one-way permutations (OWP) which are "hard-to-invert" even against an adversary running in sub-exponential time, precisely in time $O(2^{\kappa_{\mathsf{s}}})$ such that $\kappa_{\mathsf{s}} = O(\kappa^{\varepsilon}/2)$ for some $0 < \varepsilon < 1$. In particular, our construction uses any perfectly binding commitment scheme that is computationally hiding against such sub-exponential adversary (and this primitive can be constructed from the above assumption). The key technical challenge, as remarked earlier, is that BNMC is *not* an interactive primitive that allows bi-directional communication. This limitation renders the previously proposed techniques for designing non-malleable protocols inherently unusable. This is because these previous techniques are based on having "challenge-response" rounds similar to the type also used in designing zero-knowledge protocols. Thus, techniques like rewinding the sender are not useful in this setting at all: since there are no receiver messages, one would end up with the same transcript every time. Thus, apriori, it seems unclear what advantage one could get by having multiple blocks. Our final construction is quite clean and in fact, also gives arguably one of the simplest known constructions of non-malleable commitments.

Strong BNMC. Additionally, we also consider a strictly stronger model of tampering: assume any permutation $\pi : [\ell] \to [\ell]$ chosen by the adversary. Then each function f_i takes i blocks $(c_{\pi(1)}, \ldots, c_{\pi(i)})$ as input and modifies the $\pi(i)$-th block. We call this family of function strong block-wise and denote it by $\mathcal{F}^{\ell}_{\mathsf{s\text{-}block}}$. We also provide a definition of strong BNMC which is essentially an explicit presentation of NMCwR for $\mathcal{F}^{\ell}_{\mathsf{s\text{-}block}}$. We provide an unconditional generic transformation to construct strong BNMC from any BNMC which, along with the earlier results imply that any construction of BNMC can be transformed to a strong BNMC (with some blow up in the length of codeword). Details about strong BNMC are provided in Appendix B.

Comparison to Aggarwal *et al.* [2]. We note that Aggarwal, Dodis, Kazana and Obremski [2, 1] coined the notion of block-wise tampering (Def. 17 in [2]). Their work focused mainly on constructing non-malleable codes in the (standard) split state model, and, the notion of block-wise tampering is only used as an intermediate concept in their proof of security. Aggarwal *et al.* did not seek to initiate a comprehensive study of this notion or obtain explicit constructions (owing to the trivial impossibility result mentioned earlier).

1.2 Overview of our techniques

We now give a brief overview of our main construction of BNMC. The detailed construction is provided in Sec. 5.

First fix a parameter μ (such that $\mu = O(\kappa^{2+\varphi})$ for any arbitrary constant $\varphi > 0$ of our choice where κ is the security parameter) such that we encode a message m using $\ell = (2\mu + 1)$-blocks of codeword for some parameter μ . At a very high level, our encoding is as follows. Let us first fix some index (or *tag*) for the encoder $i \in [\mu]$. The encoder then chooses a *perfectly binding* commitment scheme COM.

Let $\mathsf{COM}_{\kappa_{\mathsf{s}}}(\cdot)$ and $\mathsf{COM}_{\kappa}(\cdot)$ denote that COM is computationally hidden with respect to security parameters κ_{s} and κ respectively, where κ_{s} is as mentioned above. The encoder then computes commitments to the message using $\mathsf{COM}_{\kappa_{\mathsf{s}}}$ and COM_{κ}. The first 2μ blocks of the encoding of m are blocks of all zeroes, except for block i and block $(2\mu - i)$ which are the commitments COM_{κ} and $\mathsf{COM}_{\kappa_{\mathsf{s}}}$, respectively. The $(2\mu+1)^{\text{th}}$ block of the encoding contains the openings to $\mathsf{COM}_{\kappa_{\mathsf{s}}}$ and

COM_κ. The decoding algorithm checks if (i) all the openings are consistent with the commitments and (ii) the messages committed are equal. Now, for a moment, assume that adversary's index i' is not equal to i (this can be removed later on). Then if $i' < i$, then the adversary has to output its first commitment without seeing the first commitment in the input codeword (rather only seeing on the string of zeros). Thus, the first commitment in the output is independent of the first commitment in the input. Moreover, our definition (NMCwR) puts the additional restriction that the adversary has to output a valid codeword in order to succeed. Combining one can see that the output codeword, if valid, must contain a message independent of the message encoded in the input. On the other hand, if $i' > i$, then the second commitment of the adversary has to be independent of the second commitment in the input. In this case, we rely on complexity leveraging to prove non-malleability. Using this key-observation one can prove the non-malleability except in one case: when the index chosen by the adversary i' is equal to i. To prevent mauling in this case we use one-time signatures. The encoder signs the entire codeword using i as a public-key and thus leaving the adversary either to forge the signature or change the index. However, one problem still remains. To use i as a public-key we need it to be sufficiently long, in particular for a concrete instance of such OTS (we consider variant of Lamport [27]) the length needed to be $O(\kappa^{2+\varphi})$ for any arbitrary constant $\varphi > 0$ of our choice. But note that, we have $i \in [\mu]$ and $\ell = 2\mu+1$. Trying to set the size of the index $|i| = \log(\mu)$ to even $\Omega(k)$ would result in an "inefficient" construction with $\ell = 2^{\Omega(k)}$ blocks which is not acceptable. We solve this problem by using a "well-known" technique from non-malleable commitment, so-called DDN-XOR trick. Through that, it is possible to use a long tag of size $t = O(\kappa^{2+\varphi})$ keeping the number of blocks also $O(\kappa^{2+\varphi})$ just by computing t shares (XOR's) of messages and and applying the above construction independently on the shares. So, our final construction would require a one-time signature which works with a public-key of bit-length $\mu = O(\kappa^{2+\varphi})$. The main result we present below as an informal theorem.

Theorem (Main Result (informal))**.** *Assuming the existence of sub-exponentially hard one-way permutations, for any arbitrary constant $\varphi > 0$ we can explicitly construct a block-wise non-malleable encoding scheme with $O(\kappa^{2+\varphi})$ blocks.*

1.3 Related Works

The theory of non-malleable code was introduced by Dziembowski, Pietrzak, and Wichs [18], who gave the first explicit construction of non-malleable codes for a family of function $\mathcal{F}_{\mathsf{bit}}$, which can tamper every bit of the codeword independently. They also gave an existential proof for the existence of non-malleable codes for almost the whole set of all functions, $\mathcal{F}_{\mathsf{almost}}$. Recently, Cheraghchi and Guruswami [11] gave a construction with improved rate and efficiency than [18] for $\mathcal{F}_{\mathsf{bit}}$. On the other extreme is the situation when there are exactly two disjoint blocks of codewords, i.e, the split-state model. Dziembowski, Pietrzak, and Wichs [18] also gave a construction in this model under the random oracle assumption. Since then, there has been a series of work that proposed efficient construction of non-malleable code in the split-state model in both the computational setting [30] and in the information theoretic setting [3, 11, 17]. In a recent work, Coretti *et al.* [12] applied split-state non-malleable codes with n-states to get a weaker notion of multi-bit CCA security.

In a recent work, Faust *et al.* [21] showed an efficient code for a tampering function of size $2^{s(n)}$ for some polynomial function $s(n)$ in the information-theoretic setting. Concurrently, Cheraghchi and Guruswami [10] improved the probabilistic method construction of Dziembowski, Pietrzak, and

Wichs [18] to show that one can have some level of efficient encoding and decoding if we restrict the size of the tampering functions to a set of size at most $2^{s(n)}$ for some polynomial $s(n)$.

Apart from the split-state model and $\mathcal{F}_{\mathsf{bit}}$, many recent works have studied non-malleable code in various models. Faust *et al.* [19] studied non-malleable code when the tampering function is allowed to tamper codeword as long as it does not decodes to a special symbol $\perp$. They gave a necessary condition and a construction of such codes. This work was further improved by Jafarholi and Wichs [25]. Agarwal *et al.* [5] studied a class of tampering function that can permute the bits of the encoding and (optionally) perturb them. They proposed an efficient and explicit construction of non-malleable codes in the information theoretic setting. In the follow-up work, the authors [4] demonstrated a rate-optimized compiler for NMC against bit-wise tampering and permutations. Dachman-Soled *et al.* [13] initiated the study of locally decodable and updatable non-malleable codes. They gave two constructions of such codes that are secure against continual tampering, where their concept of continuity is different from Faust *et al.* [19] in the sense that they allow an updater that updates the codeword. Chattopadhyay and Zuckerman [9] showed a construction of non-malleable code in an extension of the split-state model, where codewords is partitioned in to $c = o(n)$ equal sized blocks.

The study of non-malleable commitments was initiated by Dolev, Dwork, and Naor [16]. They showed a n-round non-malleable commitment assuming the existence of one-way function and no trusted set up. Since then, many follow up works improved the round-complexity of the original construction with some trusted infrastructure. Damgard and Groth [15] showed non-interactive non-malleable commitments based on only one-way functions in presence of some trusted infrastructure. The work of Barak [6] was the first constant round non-malleable commitments; however, their security relied on existence of trapdoor permutations and collision resistant hash function against sub exponential size circuits and the proof is non-black box. Pandey, Pass, and Vaikuntanathan [32] were the first to prove a construction of a non-interactive non-malleable commitment with a black-box proof; however, their construction was based on a new hardness assumption with a strong non-malleable flavour. Lin and Pass [28] showed an almost constant round non-malleable commitment scheme based on one-way functions and had a black-box proof of security. Pass and Wee [34] gave a constant round non-malleable commitment using sub-exponential hard one-way function. Subsequently, Goyal [23] and Lin and Pass [29] concurrently showed a constant round non-malleable commitments assuming one-way functions using different techniques.

2 Preliminaries and Basic Primitives

2.1 Notations and Basic Definitions

Let $\mathbb{N} = \{1, 2, \ldots, \ldots\}$ be the set of natural numbers. For $n \in \mathbb{N}$, we write $[n] = \{1, 2, \cdots, n\}$. Given a set A, we write $a \leftarrow A$ to denote that element a is sampled from the set A. If A is an algorithm, $y \leftarrow A(x)$ denotes an execution of A with input x and output y. For a randomized algorithm $A(\cdot, \cdot)$, the output $y \leftarrow A(x; r)$ is a random variable when the input is x and randomness r. For a set X, we use the symbol $|X|$ to denote the size of the set X. When it is clear from the context, we only write $A(x)$ instead of $A(x; r)$. For a number $j \in \mathbb{N}$, we use the notation $\mathtt{BIT}(j)$ to denote the bit-wise representation of the number j. For a string s, we let $s[i]$ denote the i-th bit of s and $s[i....j]$ to be the bits of s starting from i-th index to the j-th index. A function $\delta(\cdot) : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every polynomial $p(\cdot)$ for all large enough n, it holds that $\delta(n) < 1/p(n)$. We generically denote any negligible function by $negl(\cdot)$.

In general, throughout the paper we denote the "standard" security parameter by κ (we use another one κ_s in Sec 5 for complexity levering). Let X be a random variable. Then we sometimes abuse notations and denote the corresponding probability distribution also by X. An ensemble of probability distributions is a sequence of $\{X_\kappa\}_{k\in\mathbb{N}}$ of probability distributions. For two probability ensembles $\{X\}_\kappa$ and $\{Y\}_\kappa$ defined over a finite support S, we use the notation $\{X\}_\kappa \approx \{Y\}_\kappa$ if the two distributions are *computationally indistinguishable*, i.e., for all probabilistic polynomial time distinguishers $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$,

$$|\mathsf{Pr}_{x\leftarrow X_\kappa}[\mathcal{D}(x)=1] - \mathsf{Pr}_{y\leftarrow Y_\kappa}[\mathcal{D}(y)=1]| \le \mathsf{negl}(\kappa).$$

We use the notation $X_\kappa \approx_c Y_\kappa$ as a shorthand for computationally indistinguishable ensembles.

Similarly, two probability ensembles $\{X_\kappa\}_\kappa$ and $\{Y\}_\kappa$, defined over a finite support S, are called *statistically indistinguishable* if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$,

$$\frac{1}{2}\sum_{s\in S}|\mathsf{Pr}[X_\kappa = s] - \mathsf{Pr}[Y_\kappa = s]| \le \mathsf{negl}(\kappa).$$

We use the notation $X_\kappa \approx_s Y_\kappa$ as a shorthand for statistically indistinguishable ensembles. In this paper, wherever the subscript under $\approx$ is not mentioned, it is implicit that the two distributions are computationally indistinguishable.

3 Building Blocks

In this section we provide definitions of a few well-known primitives which are used as building blocks.

3.1 One-time Signatures

One-time signatures are digital signature schemes that provide unforgeability guarantees when the signer signs at most *one* message with every signing key. More formally: A one-time signature scheme $\mathsf{Sig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ is a triple of algorithms defined below:

1. $\mathsf{KGen}(1^\kappa)$: A randomized algorithm, which on input a security parameter 1^κ, outputs a private signing key sk and a public verification key pk.

2. $\mathsf{Sign}(sk, m)$: A randomized algorithm which outputs a signature σ for the message $m \in \mathcal{M}$ under the signing key sk.

3. $\mathsf{Verify}(pk, \sigma, m)$: A deterministic algorithm which outputs 1 if and only if σ is a valid message on m under pk and 0 otherwise.

which satisfies the following properties:

1. **Correctness**: For all message $m \in \mathcal{M}$:

$$\Pr\left[\mathsf{Verify}(pk, \mathsf{Sign}(sk, m), m) \mid (pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)\right] = 1$$

2. **Unforgeability**: For any PPT adversary A which makes only one signing query on some message $m^\star$ to the signing oracle, the following holds.

$$\Pr\left[\mathsf{Verify}(pk, \sigma, m) = 1 \wedge (m \neq m^\star) \mid (\sigma, m) \leftarrow \mathsf{A}(pk) \wedge (sk, pk) \leftarrow \mathsf{KGen}(1^\kappa)\right] \leq negl(\kappa),$$

where the probability is taken over the coin toss of $\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}$, and A.

3.2 Commitment Schemes

A *commitment scheme* denoted by $\langle \mathsf{C}, \mathsf{R} \rangle$ is a executed by two parties, a committer C and a receiver R. C runs a randomized commitment algorithm Com on the messages $m \in \mathcal{M}$ and a randomness r to generate the commitment $\mathsf{cmt} \leftarrow \mathsf{Com}(m, r)$ and send cmt to R in the commitment phase. The commitment phase might be interactive and consists of several rounds. In decommitment phase C sends the decommitment opn to R and R checks if the opening is consistent by running a deterministic decommitment algorithm $\widetilde{m} \leftarrow \mathsf{Decom}(\mathsf{cmt}, \mathsf{opn})$. If $\widetilde{m} = \perp$, then R rejects, otherwise accepts $\widetilde{m}$ as the committed value. In this paper, we use computationally hiding and perfectly binding commitment schemes which are formally defined as follows:

- Computational hiding: For any two messages $m, m' \in \mathcal{M}$, the following holds:

$$\mathsf{Com}(m) \stackrel{c}{\approx} \mathsf{Com}(m')$$

- Perfect binding: For any message $m \in \mathcal{M}$, $\Pr\left[\mathsf{Decom}(\mathsf{Com}(m)) \notin \{\perp, m\}\right] = 0$

3.3 Non-malleable commitments

We follow the definition of non-malleable commitments introduced by Pass and Rosen [33] (these in turn are built on the original definition of Dolev *et al.* [16]). We formalize the definition by comparing a man-in-the-middle and a stand-alone execution. Consider a commitment scheme $\langle \mathsf{C}, \mathsf{R} \rangle$, and a polynomial-time relation $\mathfrak{R} \subseteq \{0,1\}^{k_m} \times \{0,1\}^{k_m}$. We consider man-in-the-middle adversaries that simultaneously participate in a left and a right interaction in which a commitment scheme is taking place. The adversary is said to succeed in mauling a left commitment to a value m if it is able to come up with a commitment $\widetilde{m}$ (and its opening) on the right such that $\mathfrak{R}(m, \widetilde{m}) = 1$. The man-in-the-middle and the stand-alone executions are defined belows.

Man-in-the-middle Execution. In the man-in-the-middle execution, the adversary M simultaneously participates in a left and a right interaction. In the left interaction, the man-in-the-middle adversary M interacts with C acting as a receiver to a commitment of m. In the right interaction, M interacts with R attempting to commit to a related value $\widetilde{m}$. We assume the man-in-the-middle adversary M is *synchronizing*,[9] which means that as soon as it receives a message from the committer in the left interaction, it sends a message immediately in the right interaction. Prior to the interaction, the value m is given to C as local input. M may also have an auxiliary input z, which in particular might contain a-priori information about m. The success of M is defined using the following boolean random variable:

[9] It is sufficient to consider synchronizing adversary as Wee [36] constructed a generic compiler which transforms any non-malleable commitment scheme against synchronizing adversaries to a non-malleable commitment against asynchronous adversaries. Though Wee only considered the stronger notion (non-malleability w.r.t. commitment), it won't be too hard to show that the same also works for non-malleability w.r.t. opening (which we consider) with necessary adjustments.

- $\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z) = 1$ if and only if A decommits to a value $\widetilde{m}$ such that $\mathfrak{R}(m, \widetilde{m}) = 1$.

The stand-alone execution. In the stand-alone execution only one interaction takes place. The stand-alone adversary S (a.k.a. simulator) directly interacts with R. As in the man-in-the-middle execution, the value m is chosen prior to the interaction and S receives some a-priori information about m as part of its an auxiliary input z. S first executes the commitment phase with R. Once the commitment phase has been completed, S receives the value m and attempts to decommit to a value $\widetilde{m}$. The success of S is defined using the following boolean random variables:

- $\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m, z) = 1$ if and only if S decommits to a value $\widetilde{m}$ such that $\mathfrak{R}(m, \widetilde{m}) = 1$.

Similar to earlier works, we shall work with the tag-based definition of non-malleable commitments, in which every interaction is associated with a tag tg. If the tag tg for the left interaction is equal to the tag $\widetilde{\mathsf{tg}}$ for the right interaction, the output is 0 in both the experiments. This is why we do allow $\mathfrak{R}$ to be reflexive i.e. $\mathfrak{R}(m, m) = 1$.

Definition 3.1 (Non-malleable commitment with respect to opening)**.** *A commitment scheme* $\langle \mathsf{C}, \mathsf{R} \rangle$ *is said to be* non-malleable w.r.t. opening *if for every probabilistic polynomial-time man-in-the-middle adversary* M*, there exists a (possibly expected)* PPT *stand-alone simulator* S *and a negligible function* $negl(\cdot) : \mathbb{N} \to \mathbb{N}$*, such that for every polynomial-time computable relation* $\mathfrak{R} \subseteq \{0,1\}^{k_m} \times \{0,1\}^{k_m}$*, every message* $m \in \{0,1\}^{k_m}$*, and every* $z \in \{0,1\}^{\star}$*, it holds that:*

$$\left| \Pr\left[\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z) = 1\right] - \Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m, z) = 1\right] \right| \leq negl(\kappa).$$

We remark that, in this section we consider strictly PPT simulators.

4 Definitions

A formal definition of non-malleable codes is provided in Appendix A. Below we first present our relaxed definition namely NMC with replacement (NMCwR). Finally, we present the concrete definition of *block-wise NMC* (BNMC) along with some other relevant definitions and some basic facts about BNMC.

4.1 Non-Malleable Codes with Replacement

First let us formally present the definition of an encoding scheme. Below the symbol $\perp$ denotes usual invalidity of a codeword.

Definition 4.1 (Encoding Scheme)**.** *An* (k, n)*-encoding scheme* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *consists of two functions: a randomized* encoding *function* $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ *and a deterministic decoding function* $\mathsf{Dec} : \{0,1\}^n \to \{0,1\}^{\ell} \cup \{\perp\}$*, such that, for every* $m \in \{0,1\}^k$*,* $\Pr[\mathsf{Dec}(\mathsf{Enc}(m)) = m] = 1$.

We present the indistinguishability-based definition of NMC i.e. so-called *strong non-malleable code* introduced in [18](see Def. 3.3 in that paper) as our definitions build up on this.

Definition 4.2 (Strong Non-malleable Codes)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (k, n)*-encoding scheme. Let* $\mathcal{F}$ *be some family of tampering functions. The* Code *is called* (k, n)*-strong non-malleable code if for every* $f \in \mathcal{F}$ *and any pair of messages* $m_0, m_1 \in \{0,1\}^k$*, the following holds:*

$$\mathsf{Tamper}^f_{m_0} \approx \mathsf{Tamper}^f_{m_1}$$

where for any $m \in \{0,1\}^k$*,* Tamper^f_m *is defined as*

$$\mathsf{Tamper}^f_m \equiv \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(m); c' \leftarrow f(c); \\ \textit{If } c' = c \textit{ set } m' := \mathsf{same}^\star \textit{ else } m' \leftarrow \mathsf{Dec}(c') \\ \textit{Output: } m' \end{array} \right\}$$

where the randomness is over the encoding function Enc*.*

From now on, by NMC we will refer to the above definition unless otherwise stated explicitly.

Remark 4.3. *Note that in the above definition, the tampering experiment is allowed to output a special symbol* $\mathsf{same}^\star$ *to indicate that the tampering function leaves the codeword c unchanged.*

We introduce the "relaxed" definition of non-malleable codes which is same as that of non-malleable codes except there is a *replacer* $\mathbf{R_f}$ which is an "all powerful" algorithm and comes into play only when the modified codeword is invalid (i.e. decodes to $\perp$). In that case, the replacer may replace the $\perp$ by any message in the message space or the symbol $\mathsf{same}^\star$.[10] Since the idea of replacer is similar in spirit with the notion of *non-malleable commitment with replacement* as introduced in [23] we call this relaxed version *non-malleable codes with replacement* (NMCwR in short). We present the formal definition below.

Definition 4.4 (Non-malleable codes with replacement)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (k, n)*-encoding scheme. Let* $\mathcal{F}$ *be some family of tampering functions. Then* Code *is called* (k, n)*-non-malleable code with replacement (NMCwR) if for every* $f \in \mathcal{F}$ *there exists an algorithm called the replacer* $\mathbf{R_f}$ *such that for any pair of messages* $m_0, m_1 \in \{0,1\}^k$*, the following holds:*

$$\mathsf{TampWR}^f_{m_0} \approx \mathsf{TampWR}^f_{m_1}$$

where for any $m \in \{0,1\}^k$*,* TampWR^f_m *is defined as*

$$\mathsf{TampWR}^f_m \equiv \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(m); c' \leftarrow f(c); \\ \textit{If } c' = c \textit{ set } m' := \mathsf{same}^\star \textit{ else } m' \leftarrow \mathsf{Dec}(c') \\ \textit{If } m' = \perp \textit{ then } m' := \mathbf{R_f}(c) \;;\; \textit{Output: } m' \end{array} \right\}$$

where the randomness is over the encoding function Enc*.*

Remark 4.5. *As usual the indistinguishability depends on the setting (information theoretic or computational). However, we emphasize that even if we are in the computationally bounded scenario, where the adversary is* PPT*, we do not restrict the replacer to be a* PPT *algorithm. This assumption is justified because the replacer is required only to establish the meaningfulness of the definition without affecting the natural intuition. Intuitively the purpose of the replacer is to relax the traditional notion in a way such that the tampering function is allowed to distinguish the tampering experiments, albeit only by making the codeword invalid. Nonetheless in the computational setting all the other algorithms involved as well as the the tampering functions are required to be* PPT*.*

[10] The replacer can also keep the $\perp$ in case when it not harmful (i.e. does not depend on the input) e.g. when the tampering function always tampers to something invalid

Some intuitions. Intuition behind why the above definition is meaningful can be understood in the following. For every adversary, there is guaranteed to exist another adversary which always tampers in the same way as the original adversary, except, when the original adversary were to output an invalid codeword. In that case, the new adversary may employ any other (PPT) strategy. However when the original adversary outputs an invalid codeword, (in many applications) it could be considered as aborting or failing in those cases. Hence, our new adversary could be seen as strictly more powerful than the original one. However as the definition guarantee, the new adversary actually obeys the standard non-malleable code guarantee. Thus, in many scenarios, we believe the above weaker notion may be sufficient. Indeed, as shown in [23], the corresponding weaker notion for non-malleable commitments (called non-malleability w.r.t. replacement) turns out to be sufficient for several applications including for obtaining constant round multi-party computation.

BNMC as NMCwR for $\mathcal{F}^{\ell}_{\mathsf{block}}$. In this paper we are mainly interested in achieving the above definition for the particular class $\mathcal{F}^{\ell}_{\mathsf{block}}$ for some $\ell \in \mathbb{N}$. To define this class first assume each n-bit codeword $\mathbf{c}$ is divided into ℓ blocks $(c_1, \ldots, c_\ell)$. Then $\mathcal{F}^{\ell}_{\mathsf{block}}$ contains ℓ-tuples of functions $\mathbf{f} = (f_1, \ldots, f_\ell)$ such that each f_i gets the first i blocks $(c_1, \ldots, c_i)$ as input and output the i-th tampered block c'_i. For concreteness we present the definition of NMCwR for $\mathcal{F}^{\ell}_{\mathsf{block}}$ explicitly and call this simply *block-wise non-malleable codes*.

4.2 Block-wise Non-Malleable Codes (BNMC)

We start with the syntactic definition of *block-wise encoding scheme*.

Definition 4.6 (Block-wise encoding scheme)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (k, n)*-encoding scheme. Then it is called an* (ℓ, k, n)-block-wise encoding scheme *if each string output by* Enc *is an* ℓ*-tuple:* $(c_1, \ldots, c_\ell)$ *where* $|c_i| = n_i$*, with* $\sum_{i=1}^{\ell} n_i = n$*. Also let* $\nu_i = \sum_{j=1}^{i} n_i$*.*

Next we define a property of such block-wise encoding scheme called *reveal index*, that will be useful later on.

Definition 4.7 (Reveal Index)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (ℓ, k, n)*-block-wise encoding scheme. Then* Code *is said to have reveal index* η *if* $\eta - 1 \in [\ell]$ *is the largest index for which the following condition holds:*

- *For all pair of messages* $m_0, m_1 \in \{0,1\}^k$ *if* $(c_1^{(0)}, \ldots, c_\ell^{(0)}) \leftarrow \mathsf{Enc}(m_0)$ *and* $(c_1^{(1)}, \ldots, c_\ell^{(1)}) \leftarrow \mathsf{Enc}(m_1)$ *then* $(c_1^{(1)}, \ldots, c_{\eta-1}^{(1)}) \approx (c_1^{(1)}, \ldots, c_{\eta-1}^{(1)})$*.*

Remark 4.8. *This definition formalizes the fact that, for any encoding scheme, there is an index* η *which reveals some information about the encoded message for the first time in the sequence and before that the sequence* $(c_1, \ldots, c_{\eta-1})$ *hides the encoded message. As usual the indistinguishability denoted by "*$\approx$*" in the above definition can refer to* computational indistinguishability *or* statistical indistinguishability *depending on whether we are in the computational or information-theoretic setting respectively. Obviously* $\eta \leq \ell$ *for any block-wise encoding scheme.*

Finally, we present our main definition of a *block-wise non-malleable encoding scheme* which is essentially an explicit presentation of NMCwR for the class $\mathcal{F}^{\ell}_{\mathsf{block}}$.

Definition 4.9 (Block-wise non-malleable codes)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (ℓ, k, n)*-block-wise encoding scheme. Let* $\mathbf{f} = (f_1, \ldots, f_\ell)$ *be any tuple of functions specified as follows:* $\forall i \in [\ell], f_i : \{0,1\}^{\nu_i} \to \{0,1\}^{n_i}$*. Then* Code *is called an* (ℓ, k, n)-block-wise non-malleable code *(*BNMC *in short) if, for any such tuple* $\mathbf{f}$*, there exists a replacer* $\mathbf{R_f}$*, such that, for any pair of messages* $(m_0, m_1) \in \{0,1\}^k$*, the following holds:*

$$\mathsf{BLTamp}^{\mathbf{f}}_{m_0} \approx \mathsf{BLTamp}^{\mathbf{f}}_{m_1}.$$

where $\mathsf{BLTamp}^{\mathbf{f}}_m$ *for any* $m \in \{0,1\}^k$ *is defined as:*

$$\mathsf{BLTamp}^{\mathbf{f}}_m = \left\{ \begin{array}{c} \mathbf{c} = (c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m);\ \forall i \in [\ell] : c'_i = f_i(c_1, \cdots, c_i); \\ \textit{Let } \mathbf{c}' = (c'_1, \ldots, c'_\ell);\ \textit{If } \mathbf{c}' = \mathbf{c} \textit{ then set } m' := \mathsf{same}^\star; \textit{Else decode } m' \leftarrow \mathsf{Dec}(c'_1, \ldots, c'_\ell); \\ \textit{If } m' = \perp \textit{ then } m' \leftarrow \mathbf{R_f}(c_1, \ldots, c_\ell);\ \textit{Output } m' \end{array} \right\}$$

Remark 4.10. *It is easy to see that any* BNMC *has reveal index* ≥ 2*.*

4.3 Uniqueness of BNMC

We now define an important parameter of BNMC called *uniqueness index* which is similar in spirit to the uniqueness defined in [20] in the context of continuous non-malleable codes.

Definition 4.11 (Uniqueness index)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (ℓ, k, n)*-block-wise non-malleable encoding scheme. Let* $\zeta \in [\ell]$ *be the minimum index such that there does not exist any pair of codewords* $\mathbf{c} = (c_1, \ldots, c_\ell)$ *and* $\mathbf{c}' = (c'_1, \ldots, c'_\ell)$ *for which the following holds:*

- $c_i = c'_i,\ \forall\ i \in \{1, \ldots, \zeta - 1\}$ *;*
- $\perp \neq \mathsf{Dec}(\mathbf{c}) \neq \mathsf{Dec}(\mathbf{c}') \neq \perp$.

Then we call ζ *the* uniqueness index *of* Code*. Alternatively we call that* Code *has* ζ*-uniqueness and also call such an encoding scheme a* ζ-unique code.

Remark 4.12. *From the correctness property of the code, it follows that* $\zeta \leq \ell$*. Also, note that, if an BNMC has* ζ*-uniqueness, then for any valid codeword, the first* $j \geq \zeta$ *blocks* uniquely *determine the encoded message.*

We now state the following lemma without proof that the uniqueness index of a block-wise non-malleable encoding scheme must always be strictly less than its reveal index.

Lemma 4.13. *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (ℓ, k, n)*-BNMC with reveal index* $j + 1$ *and uniqueness index* j'*. Then* $j' \leq j$*.*

Proof. The proof is by contradiction. Assume that $j' \geq j + 1$. This implies the following:

- From the definition of reveal index, we know that j is the maximum index for which the first j-blocks of codewords for any two messages are indistinguishable. In other words, there exists a pair of messages (m_0, m_1) and an *admissible*[11] adversary A that can distinguish between distributions $(c_1^{(0)}, \ldots, c_{j+1}^{(0)})$ and $(c_1^{(1)}, \ldots, c_{j+1}^{(1)})$ where $\mathbf{c}_0 = (c_1^{(0)}, \ldots, c_\ell^{(0)}) \leftarrow \mathsf{Enc}(m_0)$ and $\mathbf{c}_1 = (c_1^{(1)}, \ldots, c_\ell^{(1)}) \leftarrow \mathsf{Enc}(m_1)$. Without loss of generality assume that A outputs the bit $b \in \{0,1\}$ to signal the encoding is generated from m_b.

[11] An admissible adversary refers to a PPT algorithm in the computationally setting and unbounded in the information-theoretic setting.

- From the definition of uniqueness index, there exists a pair of codewords $\mathbf{c} = (c_1, \ldots, c_{j'}, c_{j'+1}, \ldots, c_\ell)$ and $\hat{\mathbf{c}} = (c_1, \ldots, c_{j'}, \hat{c}_{j'+1}, \ldots, \hat{c}_\ell)$ (for $j' \geq j+1$) such that $\mathsf{Dec}(\mathbf{c}) = m \neq \perp$, $\mathsf{Dec}(\hat{\mathbf{c}}) = \widehat{m} \neq \perp$ and $m \neq \widehat{m}$.

When the above two statements hold, we shall construct another admissible adversary B that can distinguish between any two tampering experiments $\mathsf{Tamper}^{\mathbf{f}}_{m_0}$ and $\mathsf{Tamper}^{\mathbf{f}}_{m_1}$ using A, thus violating the non-malleability of the code. The details follows.

Let $\mathbf{t} = (\tau_1, \ldots, \tau_\ell) \leftarrow \mathsf{Enc}(m_b)$ be the target codeword where $b \in \{0,1\}$.

<u>Description of $\mathsf{B}^{\mathsf{A}(\cdot,\cdot)}$:</u>

- Gets the pair $\mathbf{c}$ and $\hat{\mathbf{c}}$ as auxiliary inputs.
- Fix the random tape of $\mathsf{A}(\cdot,\cdot)$ to some randomness r. Now $\mathsf{A}(r,\cdot)$ becomes a deterministic algorithm.
- Design function tuple $\mathbf{f} = (f_1, \ldots, f_\ell)$ as follows:
  - Each function f_i is hard-wired with the pair $(\mathbf{c}, \hat{\mathbf{c}})$ and the adversary $\mathsf{A}(r,\cdot)$ as a subroutine.
  - For $i \in [j']$ each f_i is a constant function that disregards the input and always tampers the i^{th} codeword block to c_i.
  - For $i \in \{j'+1, \ldots, \ell\}$ each f_i runs $\mathsf{A}(r,\cdot)$ on the tuple $(\tau_1, \ldots, \tau_{j+1})$ (this is possible as $j' \geq j+1$, by assumption). If $\mathsf{A}(r, (\tau_1, \ldots, \tau_{j+1}))$ outputs 0, then f_i overwrites with c_i; otherwise it overwrites with $\hat{c}_i$.

Clearly for such functions $\mathbf{f}$, $\mathsf{Tamper}^{\mathbf{f}}_{m_0}$ would always output m and $\mathsf{Tamper}^{\mathbf{f}}_{m_1}$ would always output $\widehat{m}$ unless the tuple $(\tau_1, \ldots, \tau_j)$ is the same as one of the tuples $(c_1, \ldots, c_j)$ and $(\hat{c}_1, \ldots, \hat{c}_j)$. However, since the encoding procedure is randomized and the length of the first j-block is polynomial in the security parameter κ, this happens with negligible probability (in κ) and hence the above adversary can distinguish between $\mathsf{Tamper}^{\mathbf{f}}_{m_0}$ and $\mathsf{Tamper}^{\mathbf{f}}_{m_1}$, thus violating the non-malleability of the code. Hence $j' \leq j$. $\square$

Similar in spirit to [20], we state the following corollary that any BNMC has a uniqueness index of at most $\ell - 1$.

Corollary 4.14. *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (ℓ, k, n)*-block-wise non-malleable code having* ζ*-one-sided uniqueness. Then* $\zeta \leq \ell - 1$.

4.4 Impossibility of Information-theoretic BNMC

We now show that it is impossible to construct BNMC against unbounded (block-wise) adversaries (i.e. in the information-theoretic setting).

Lemma 4.15. *It is impossible to construct an information-theoretic block-wise non-malleable code.*

Proof. Assume for the sake of contradiction that Code is an information-theoretically secure (ℓ, k, n)-block-wise non-malleable code. From Corollary 4.14, we can assume that Code has j-uniqueness for some $j \leq \ell - 1$. This implies that there must exist a pair of codewords $\mathbf{c} = (c_1, \ldots, c_{j-1}, c_j, \ldots, c_\ell)$ and $\hat{\mathbf{c}} = (c_1, \ldots, c_{j-1}, \hat{c}_j, \ldots, \hat{c}_\ell)$ such that they are valid and decode to different messages $\perp \neq m \leftarrow \mathsf{Dec}(\mathbf{c})$, $\perp \neq \hat{m} \leftarrow \mathsf{Dec}(\hat{\mathbf{c}})$[12]

Consider the experiments $\mathsf{Tamper}^{\mathbf{f}}_{m_0}$ and $\mathsf{Tamper}^{\mathbf{f}}_{m_1}$ for a pair of messages $m_0, m_1 \in \{0,1\}^k$ such that $m_0, m_1 \notin \{m, \widehat{m}\}$[13]. The unbounded adversary, finds the pair $(\mathbf{c}, \hat{\mathbf{c}})$ by brute force. Let $\mathbf{t} = (\tau_1, \ldots, \tau_\ell) \leftarrow \mathsf{Enc}(m)$ be the target codeword. The adversary's set of tampering functions $\mathbf{f} = (f_1, \ldots, f_\ell)$ are described as follows:

1. For $i \in [j-1]$, f_i overwrites τ_i to c_i.
2. For $i \in \{j, \ldots, \ell\}$, f_i first determine the *unique* encoded message $\widetilde{m}$ by trying all possibilities. Note that this is indeed possible as the target codeword is valid (encodes one of m_0, m_1) and by j-uniqueness, the message $\widetilde{m}$ is uniquely determined by the first j blocks of the target codeword. If $\widetilde{m} = m_0$, then it tampers to m; otherwise, if $\widetilde{m} = m_1$, then it tampers to $\widehat{m}$.

Clearly the experiment $\mathsf{Tamper}^{\mathbf{f}}_{m_0}$ would always outputs m whereas $\mathsf{Tamper}^{\mathbf{f}}_{m_1}$ would always outputs $\widehat{m}$; hence they can be easily distinguished. This shows the impossibility of information-theoretic block-wise non-malleable encoding schemes. □

Henceforth, from now on we focus only on computationally bounded scenario where the adversaries are PPT and the functions are efficient; however, as mentioned in Remark 4.5, we do not put any restriction on the efficiency of the replacer, in particular it is allowed to run in super-poly (or even exponential) time. In fact, later in this paper, we often encounter a replacer which is running in exponential time. Nonetheless, since we are in computationally bounded scenario we must restrict the reduction to be PPT. We are indeed able to overcome this technical hurdle by constructing such "efficient" reductions which can correctly simulate behavior of "highly inefficient" replacers.

5 Our Construction

In this section, we provide our main construction of a BNMC based on *sub-exponentially hard one-way permutations*. We construct the encoding scheme in three steps:

1. In Sec 5.1 we begin by constructing a weaker BNMC that we call Tag-based block-wise non-malleable encoding scheme (TBNMC). In such a code, every codeword has a *tag* associated with it and the tampering function must change the tag of a codeword in order to successfully maul a codeword. In other words, we allow an adversary to create a related codeword only when the tag remains the same. The tag used here is an index of the block and hence is only of size $\log(\kappa)$.
2. Then in Sec. 5.2 we use a technique, commonly known as the DDN-XOR trick [16], to construct a TBNMC with tags of length $poly(\kappa)$.

[12]In particular here we use the fact (see Def. B.2) that j is the minimum such index.
[13]This is in order to avoid any possibility of getting $\mathsf{same}^\star$.

3. Finally in Sec. 5.3 we construct an BNMC which achieves Def. 4.9, by using the public key of a one-time signature scheme as the tag of the above code, and by signing the entire codeword using the corresponding signing key.

5.1 Tag-based non-malleability

In this section we diverge from our original definition and construct an encoding scheme which meets a weaker definition of non-malleability. Although the concept of tag (or identity) is well-established in non-malleable commitment literature, it is not clear how that can be extended to the non-malleable code scenario due to its inherent non-interactive nature. Here we import the concept of tags in non-malleable code as well, albeit in a very particular and construction-specific way only for better modularity and simplicity. As a first step we provide a construction satisfying this weaker notion.

We define the tag to be always the first block of any codeword. A tag-based BNMC or TBNMC is defined exactly as the same way as BNMC with the only difference that whenever the tag of the tampered codeword is equal to the tag of the original codeword the tampering experiment outputs $\mathsf{same}^\star$ even if there is any other modification. Clearly this is strictly weaker than BNMC. Below we present the formal definitions.

Definition 5.1 (Tag of a codeword)**.** *Let* Code *be an* (ℓ, k, n)*-block-wise encoding scheme. Then for any codeword* $\mathbf{c} = (c_1, \ldots, c_\ell)$*, the tag of the codeword, denoted by* $\mathrm{Tag}(\mathbf{c})$ *is defined to be the first block* $\mathrm{Tag}(\mathbf{c}) = c_1$*.*

Now we define *Tag-based block-wise code* which is defined for a fixed tag, in that the encoding algorithm always outputs a codeword with the tag (i.e. the first block) is equal to that fixed tag.

Definition 5.2 (Tag-based block-wise code)**.** *For any tag* $\mathsf{tg} \in \mathbb{N}$*, a* (ℓ, k, n)*-block-wise encoding scheme* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *is called a* $(\mathsf{tg}, \ell, k, n)$-tag-based block-wise encoding scheme *if for all messages* $m \in \{0,1\}^k$*, for any codeword generated by the encoding algorithm,* $\mathbf{c} \leftarrow \mathsf{Enc}(m)$ *we have* $\mathrm{Tag}(\mathbf{c}) = \mathsf{tg}$

Definition 5.3 (Tag-based block-wise non-malleable codes)**.** *Let* $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ *be a* $(\mathsf{tg}, \ell, k, n)$*-tag-based block-wise encoding scheme. Let* $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$ *be any tuple of functions such that* $\forall i \in [\ell], \widehat{f}_i : \{0,1\}^{\nu_i} \to \{0,1\}^{n_i}$*. Then* TCode *is called a* $(\mathsf{tg}, \ell, k, n)$-tag-based-block-wise non-malleable code *(TBNMC) if for any such tuple* $\widehat{\mathbf{f}}$ *there exists a replacer* $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ *such that for any pair of messages* $(m_0, m_1) \in \{0,1\}^k$*, the following holds:*

$$\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0} \approx \mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$$

where $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m}$ *for any* $m \in \{0,1\}^k$ *is defined as:*

$$\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m} = \left\{ \begin{array}{c} \mathbf{c} = (c_1, \ldots, c_\ell) \leftarrow \mathsf{TEnc}(m);\ \forall i \in [\ell] : c'_i = \widehat{f}_i(c_1, \cdots, c_i); \\ \textit{Let } \mathbf{c}' = (c'_1, \ldots, c'_\ell); \textit{If } \mathrm{Tag}(\mathbf{c}') = \mathsf{tg} \textit{ then set } m' := \mathsf{same}^\star \\ \textit{Else decode } m' \leftarrow \mathsf{TDec}(c'_1, \ldots, c'_\ell);\ \textit{If } m' = \perp \textit{ then } m' \leftarrow \widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}(c_1, \ldots, c_\ell);\ \textit{Output } m' \end{array} \right\}$$

Remark 5.4. *Note that this definition is strictly weaker than* BNMC *(Def. 4.9) as it does not allow tampering of any other part of the codeword when the tag (i.e. the first block) is unchanged.*

Now we construct an encoding scheme which satisfies this weaker definition Def. 5.3 based on sub-exponentially hard OWP. The proof uses complexity leveraging which essentially forces us to assume sub-exponential hardness as opposed to standard (super-poly) hardness.

Using complexity leveraging. We assume that sub-exponentially hard one-way permutations (OWP for short) exist that are considered to be hard to break even if the adversary is allowed to run in sub-exponential time, namely in $O(2^{\kappa_{\mathsf{s}}})$ such that $\kappa_{\mathsf{s}} = \kappa^{\varepsilon}/2$ (recall that κ is the security parameter) for some constant $\varepsilon \in (0, 1)$. The proof crucially relies on this as it uses one level of complexity leveraging. In particular, while reducing to such OWP, we assume that the adversary (the reduction in this case) is unable to break the one-way permutation (the hiding of a commitment scheme in this case) even when it is allowed to run in time $O(2^{\kappa_{\mathsf{s}}})$ (but in time $o(2^{\kappa})$).

Our construction. We use a non-interactive commitment Com that is perfectly binding. We write $\mathsf{Com}_{\kappa_{\mathsf{s}}}$ and Com_{κ} to denote the commitment scheme has computational hiding with the security parameters κ_{s} and κ, respectively. In particular, Com_{κ} is a computationally hiding commitment scheme even against an adversary running in $O(2^{\kappa_{\mathsf{s}}})$ time. Suppose that such commitment scheme, on input some bit-string of length $k \in \mathbb{N}$, outputs commitments of length $\mathsf{p}(\kappa, k)$ where $\mathsf{p}(\cdot) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a fixed polynomial (determined by the specifications of the commitment scheme) in security parameter. We stress that such commitments can be constructed from *sub-exponentially hard one-way permutations.*

First we give a brief overview of the construction. Let $\mu \in \mathbb{N}$ be a parameter. We will now construct a TBNMC with ℓ blocks where $\ell = 2\mu + 2$. For now, assume ℓ to be a even number. Now for any tag $\mathsf{tg} \in [\mu]$ we construct the encoding scheme as follows: we put strings of 0 in all the blocks except the four "special" blocks: the first block is set to tg, the $(\mathsf{tg}+1)$-th block is set to the "bigger" commitment $\mathsf{Com}_{\kappa}(m)$, the $(\ell - \mathsf{tg})$-th block is set to the "smaller" commitment $\mathsf{Com}_{\kappa_{\mathsf{s}}}(m)$ and the ℓ-th (and final) block is set to the openings of the commitments. Now, for odd ℓ, one can just append one dummy block (string of 0's) right before the final block. So, without loss of generality we would assume ℓ to be even in this section. The detail construction is presented in Fig. 1. Note that here the blocks are of different length. However, it is easy to convert the code with equal block-length by padding additional zeros. We keep it without such padding for simplicity.

Remark 5.5. *From the computational hiding property of the commitment scheme, it follows that the construction has reveal index $\ell = 2\mu + 2$ for any* PPT *adversary.*

Now we prove that the construction is a TBNMC.

Theorem 5.6. *Let $\mu \in \mathbb{N}$ be some parameter. Assume that sub-exponentially hard one-way-permutations exists. Then, for any tag $\mathsf{tg} \in [\mu]$ and any $k \in \mathbb{N}$, the encoding scheme $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ described in Fig. 1 is a $(\mathsf{tg}, \ell, k, n)$-TBNMC against all* PPT *adversary such that $n = O(k + \mu \cdot \mathsf{p})$ and $\ell = 2\mu + 2$.*

Proof. Fix a function tuple $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$ and a pair of message $(m_0, m_1) \in \{0,1\}^k$. To prove the theorem we need to show the existence of a replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ such that no PPT adversary can distinguish between the experiments $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0}$ and $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$.

Constructing the replacer: We construct the replacer as follows:

$\underline{\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}(c_1, \ldots, c_\ell)\text{:}}$

Parameters: Let $\mathsf{Com}_{\kappa_\mathsf{s}}$ takes a k-bit message as input and u_s-bit randomness to produce a v_s-bit commitment and Com_κ takes a message of the same length, but randomness of u-bit to produce a v-bit commitment[a]. Let $\mathsf{tg} \in [\mu]$ be the tag of the encoding scheme for some $\mu \in \mathbb{N}$. We define a $(\mathsf{tg}, \ell, k, n)$-block-wise encoding scheme where $\ell = 2\mu+2$ and $n = k + u_\mathsf{s} + u + \mu(v_\mathsf{s} + v) + \lfloor \log \mu \rfloor + 1$ as follows:

Encoding $\mathsf{TEnc}(m)$: The encoder gets a message $m \in \{0,1\}^k$ as input and do as follows:

1. Initialize: Choose randomnesses $r_\mathsf{s} \xleftarrow{\$} \{0,1\}^{u_\mathsf{s}}$ and $r \xleftarrow{\$} \{0,1\}^u$ for commitment scheme. Set the first block $c_1 := \mathsf{tg}$.
2. Stage-1: For all $i \in \{2, \ldots, \mu+1\}$, define the i-th block of codeword c_i as follows:
$$c_i := \begin{cases} 0^v & i \neq \mathsf{tg}+1 \\ \mathsf{Com}_\kappa(m, r) & i = \mathsf{tg}+1 \end{cases}$$
3. Stage-2: For all $i \in \{\mu+2, \ldots, 2\mu+1\}$, define the i-th block of codeword c_i as follows:
$$c_i := \begin{cases} 0^{v_\mathsf{s}} & i \neq 2\mu+2-\mathsf{tg} \\ \mathsf{Com}_{\kappa_\mathsf{s}}(m, r_\mathsf{s}) & i = 2\mu+2-\mathsf{tg} \end{cases}$$
4. Final stage: Define the last block as the decommitments i.e. the message and the randomnesses in the order of commitments are sent:
$$c_{2\mu+1} := (m, r, r_\mathsf{s})$$

Decoding $\mathsf{TDec}(\mathbf{c})$: On receiving a codeword $\mathbf{c}$ parse it as $\mathbf{c} = (c_1, \ldots, c_{2\mu+2})$ such that $|c_1| = \lfloor \mu \rfloor + 1$, for $i \in \{2, \ldots, \mu+1\}$, $|c_i| = v$, for $i \in \{\mu+2, \ldots, 2\mu+1\}$, $|c_i| = v_\mathsf{s}$ and for $i = 2\mu+2$, $|c_i| = k + u_\mathsf{s} + u$. Then do as follows:

1. Correctness of Structure: First check if the structure is correct: that is if $c_1 \neq 0$ and there are exactly two indexes $i_1 \in \{2, \ldots, \mu+1\}$, $i_2 \in \{\mu+2, 2\mu+1\}$ such that:
   (a) $c_{i_1} \neq 0^v$ and $c_{i_2} \neq 0^{v_\mathsf{s}}$.
   (b) for all other indexes $i \in \{2, \ldots, \mu+1\} \setminus \{i_1\}$, $c_i = 0^v$ and $i \in \{\mu+2, \ldots, 2\mu+1\} \setminus \{i_2\}$, $c_i = 0^{v_\mathsf{s}}$.
   (c) $i_1 + i_2 = 2\mu + 1$.

   if any of them fails, then the structure of the tampered codeword is incorrect and therefore output $\perp$, else go to the next step.
2. Consistency of commitment: Parse $c_{2\mu+2}$ as $(m, r, r_\mathsf{s}) := c_{2\mu+2}$ such that $|m| = k$, $|r| = u$ and $|r_\mathsf{s}| = u_\mathsf{s}$. Then check the validity of the commitment-decommitment pair $(c_{i_1}, (m, r))$ and $(c_{i_2}, (m, r_\mathsf{s}))$, if any of them are invalid output $\perp$, otherwise output the committed message m.

[a] We assume $|v_\mathsf{s}|, |v| = \mathit{poly}(\kappa)$

Figure 1: The construction of $(\mathsf{tg}, \ell, k, n)$-TBNMC for tag size $\log \kappa$.

On input a tuple $\mathbf{c} = (c_1, \ldots, c_\ell)$, the replacer first generates the tampered codeword $\mathbf{c}' = (c'_1, \ldots, c'_\ell)$ as in the real experiment. Let $\mathsf{tg}(\mathbf{c}) = \mathsf{tg}$ and $\mathsf{tg}(\mathbf{c}') = \widetilde{\mathsf{tg}}$. Then, depending on the values of $\widetilde{\mathsf{tg}}$ it works as follows:

1. If $\widetilde{\mathsf{tg}} = \mathsf{tg}$, then output $\mathsf{same}^\star$.
2. Otherwise first check if the structure is correct (Step-1 of decoding). If not, then it outputs $\perp$.
3. If the structure is correct and $\widetilde{\mathsf{tg}} \neq \mathsf{tg}$, then perform the following checks:
   (a) If $\widetilde{\mathsf{tg}} < \mathsf{tg}$, then compute the message committed in the first stage of the tampered codeword by brute-force and output it. Note that, this message is unique by perfect binding of Com.
   (b) If $\widetilde{\mathsf{tg}} > \mathsf{tg}$, then compute the message committed in the second stage of the tampered codeword by brute-force and output it.

The reduction using one-level complexity leveraging. Our aim is to prove that, for the above replacer, the distributions $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0}$ and $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$ are computationally indistinguishable. The key idea is to reduce to the hiding property of the commitment with respect to the bigger security parameter κ and allow the reduction to run in time $O(2^{\kappa_\mathsf{s}})$ hence relying crucially on complexity leveraging.

Assume, for the sake of contradiction, that there exists a PPT adversary A which can distinguish between experiments $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0}$ and $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$ while running in $o(2^{\kappa_\mathsf{s}})$-time. We say that A outputs a bit b while it detects the experiment to be $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$. Therefore, following holds for a randomly chosen $b \in \{0, 1\}$:

$$\Pr\left[(\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{A}) = b\right] > 1/2 + \varepsilon(\kappa_\mathsf{s}) \tag{1}$$

for some non-negligible function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{N}$ of the security parameter κ_s.

Denote the encoding of m_b in experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ by $\mathbf{c}^{(b)}$, the tampered codeword by $\widetilde{\mathbf{c}}^{(b)}$. The i-th block of any codeword $\mathbf{c}^{(b)}$ is denoted by $c_i^{(b)}$.

Formally we prove the following claim.

Lemma 5.7. *If* $\Pr\left[(\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{A}) = b\right] > 1/2 + \varepsilon(\kappa)$ *for some non-negligible function* $\varepsilon : \mathbb{N} \to \mathbb{N}$ *then there exists a* PPT *adversary* B *which can break hiding of the commitment scheme* Com_κ *(with probability at least* $\varepsilon(\kappa)$*) if* B *is allowed to run in* $O(2^{\kappa_\mathsf{s}})$ *(but* $o(2^\kappa)$*) time.*

Proof. We start with the observation that, for any tuple of functions $\widehat{\mathbf{f}}$, the tampered tags are the same in both the experiments since they are deterministically computed as a function of the original tag tg as $\widetilde{\mathsf{tg}} = f_1(\mathsf{tg})$. Now we describe the reduction B: B receives a commitment $\mathsf{cmt}^\star = \mathsf{Com}_\kappa(m_b)$ for some randomly chosen bit $b \in \{0, 1\}$ and some auxiliary input z. It will run the tampering adversary A, hence the main task of B is to simulate the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ correctly which it does as follows:

- B creates a dummy commitment $\mathsf{Com}_{\kappa_\mathsf{s}}(0^k)$ and defines the first $\ell - 1$ blocks of the input codeword as follows:

- $c_1 := \mathsf{tg}$.
- For all $i \in \{2, \ldots, \mu+1\}$, define the i-th block of codeword c_i as follows:

$$c_i := \begin{cases} 0^v & i \neq \mathsf{tg}+1 \\ \mathsf{cmt}^\star & i = \mathsf{tg}+1 \end{cases}$$

- For all $i \in \{\mu+2, \ldots, 2\mu+1\}$, define the i-th block of codeword c_i as follows:

$$c_i := \begin{cases} 0^v & i \neq 2\mu+2-\mathsf{tg} \\ \mathsf{Com}_{\kappa_\mathsf{s}}(0^k) & i = 2\mu+2-\mathsf{tg} \end{cases}$$

- Then it runs the adversary A to receive the tampering function tuple $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$. Using $\widehat{\mathbf{f}}$, it computes the first $\ell-1$ tampered blocks $(\widetilde{c}_1^{(b)}, \ldots, \widetilde{c}_{\ell-1}^{(b)})$ where $\widetilde{c}_1^{(b)} = \widetilde{\mathsf{tg}} = f_1(\mathsf{tg})$ is the tag of the tampered code.
- Depending on the value of $\widetilde{\mathsf{tg}}$, B proceeds as follows:
  - If $\widetilde{\mathsf{tg}} = \mathsf{tg}$, then return $\mathsf{same}^\star$ to A.
  - Otherwise, B checks if the structure of $\widetilde{\mathbf{c}}^{(b)}$ is correct (Note that the structure of any codeword is determined by the first $\ell-1$ blocks). If not, then return $\perp$ to A. Otherwise, B checks if $\widetilde{\mathsf{tg}} < \mathsf{tg}$.
    * If it is, then B returns the auxiliary input z.
    * If it is not, then B runs in $O(2^{\kappa_\mathsf{s}})$ time to compute the committed messages m' inside the block $\widetilde{c}^{(b)}_{2\mu+1-\widetilde{\mathsf{tg}}}$ by brute force, and return m' to A. This is the part of the proof where we use complexity leveraging.
- Finally it outputs the decision bit returned by A.

In order to proceed with the proof, we need to argue that B correctly simulates the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ to A. We analyze this case by case.

1. If $\widetilde{\mathsf{tg}} = \mathsf{tg}$, then the replacer would also output $\mathsf{same}^\star$. Hence the simulation is correct.
2. If $\widetilde{\mathsf{tg}} \neq \mathsf{tg}$, then we split into the following sub-cases.
   (a) *When the structure of $\widetilde{\mathbf{c}}^{(b)}$ is incorrect.* It is easy to see that the simulation is correct in this case. This is because if the replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ is invoked in either $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0}$ or $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$, then it would output $\perp$. On the other hand, note that the structure of $\widetilde{\mathbf{c}}^{(b)}$ is determined entirely by three values: the tag and the two commitments; all the other values are set to be string of 0. However, B replaces the second commitment with a dummy commitment. Here the hiding property of $\mathsf{Com}_{\kappa_\mathsf{s}}$ comes to our rescue. Due to the hiding property of the scheme $\mathsf{Com}_{\kappa_\mathsf{s}}$, the PPT adversary A can not distinguish this change from the actual experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$.
   (b) *When the structure of $\widetilde{\mathbf{c}}^{(b)}$ is correct.* This can be further split into following two sub-cases according to the value of the tag.

i. $\widetilde{\mathsf{tg}} < \mathsf{tg}$. In this case, the tampering function puts the first-stage commitment in the $\widetilde{\mathsf{tg}}$-th block $\widetilde{c}^{(0)}_{\widetilde{\mathsf{tg}}}$. Now in the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0}$, $\widetilde{c}^{(0)}_{\widetilde{\mathsf{tg}}} = f_{\widetilde{\mathsf{tg}}}(\mathsf{tg}\|0^{\nu_{\widetilde{\mathsf{tg}}}})$ where $\nu_{\widetilde{\mathsf{tg}}} = \sum_{i=1}^{\widetilde{\mathsf{tg}}} n_i$. Therefore, in the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$, it deterministically use exactly the same value as the committed value in the $\widetilde{\mathsf{tg}}$-th block since the input $\mathsf{tg}\|0^{\nu_{\widetilde{\mathsf{tg}}}}$ to the $\widetilde{\mathsf{tg}}$-th tampering function is the same. In other words, we would have $\widetilde{c}^{(1)}_{\widetilde{\mathsf{tg}}} = f_{\widetilde{\mathsf{tg}}}(\mathsf{tg}\|0^{\nu_{\widetilde{\mathsf{tg}}}})$. In this case, B returns the auxiliary input z. Now, it is possible to fix the auxiliary input z to a value such that $\mathsf{Com}_\kappa(z) = f_{\widetilde{\mathsf{tg}}}(\mathsf{tg}\|0^{\nu_{\widetilde{\mathsf{tg}}}})$. This is possible as it depends only on tg which is also fixed a priori. Moreover since the structure is correct, there are two possibilities: (i) either the codeword is valid – in that case the output would be the message committed in $\widetilde{c}^{(b)}_{\widetilde{\mathsf{tg}}}$ ($b \in \{0,1\}$); (ii) or the codeword is invalid (possibly dependent on the input) – in that case, the replacer would output that message. Hence in this case, the simulation is correct.

ii. $\widetilde{\mathsf{tg}} > \mathsf{tg}$. This implies that $2\mu+2-\widetilde{\mathsf{tg}} < 2\mu+2-\mathsf{tg}$, which, in particular, implies that the $(2\mu+2-\widetilde{\mathsf{tg}})$-th tampered block is not dependent on the $(2\mu+2-\mathsf{tg})$-th input block and all the input blocks $(c_1, \ldots, c_{2\mu+2-\widetilde{\mathsf{tg}}})$ are correctly defined at this stage. Recall that B defined the $(2\mu+2-\mathsf{tg})$-th input block to a dummy commitment which does not affect the $(2\mu+2-\widetilde{\mathsf{tg}})$-th tampered block in this case. There are two possible sub-cases:

Case 1: (When the tampered codeword $\widetilde{\mathbf{c}}^{(b)}$ is valid). This implies that the committed values are consistent with the openings contained in the final block $\widetilde{c}^{(b)}_{\ell}$. So, clearly the value will be the same as the value committed in the block $\widetilde{c}^{(b)}_{2\mu+2-\widetilde{\mathsf{tg}}}$, which B returns. Hence in this case the simulation is perfect.

Case 2: (When tampered codeword $\widetilde{\mathbf{c}}^{(b)}$ is invalid). In this case the replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ will be invoked. However, since the structure is correct, we get (from the description of the replacer) that the output of the tampering experiment is equal to the value committed in the block $\widetilde{c}^{(b)}_{2\mu+2-\widetilde{\mathsf{tg}}}$, which is what B returns. Hence, the simulation is perfect in this case as well.

Since the above cases are exhaustive we can conclude that B runs in time $O(2^{\kappa_{\mathsf{s}}})$ and simulate the view of experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ correctly; thereby, breaking the hiding of the commitment Com_κ with probability at least $\varepsilon(\kappa)$.

□

This concludes the proof of the theorem. □

Problem of applying signature directly. Now, with a construction of TBNMC in hand the natural intention is to build a BNMC applying a "standard" trick: namely, use a one-time signature and sign the entire codeword with respect to the tag as the verification-key. Notice that, for this we do *not* need any additional assumption as Lamport [27] showed that one-time signatures can be built from any one-way function and therefore can be already built from our current assumption (sub-exponentially hard OWP). However, for the security of the signature scheme (against PPT adversary), the size of such verification-key must be at least $\Omega(\kappa)$. Notice that, in the above construction tag-size is bounded by $|\mathsf{tg}| = O(\log(\mu))$. Moreover, the number of blocks ℓ is linearly related to μ as $\ell = 2\mu + 2$. Evidently, setting the tag-size $|\mathsf{tg}| = \Omega(\kappa)$ would result in a code with exponentially many blocks as $\ell = 2^{O(|\mathsf{tg}|)} = 2^{\Omega(\kappa)}$ rendering the construction *inefficient*.

Therefore, in order to apply the "signature trick", we need to build a code which supports (i) "larger " tag (ii) has at most polynomially many blocks. In the next section we attempt to "amplify" exponentially the tag-size without blowing up the block-size with a technique known as DDN-XOR trick [16].

5.2 Non-malleability amplification

In this section we extend our construction to an efficient construction which can support larger tags. This extension is similar to a well-known phenomenon, namely *non-malleability amplification* [28] in the non-malleable commitment literature. The key-idea is to use the "so-called" DDN-XOR trick, introduced in [16].

5.2.1 One-many non-malleability.

Towards that, we first show that the construction given in Fig. 1 already satisfies a stronger notion, which we call *one-many* tag-based non-malleability (OMTBC). This definition, informally states that an adversary that is able to tamper a single codeword of m, cannot even come up with a set of codewords such that one of them is related to m. In particular, each function f_i in the tuple $\mathbf{f} = (f_1, \ldots, f_\ell)$ has much larger range than the domain and produces many c_i's together with the knowledge of the first i blocks of the input codeword[14].

Definition 5.8 (One-many tag-based BNMC)**.** *Let* $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ *be a* $(\mathsf{tg}, \ell, k, n)$*-tag-based block-wise encoding scheme. Let* $t \in \mathbb{N}$ *be a parameter and* $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$ *be any tuple of functions such that* $\forall i \in [\ell], \widehat{f}_i : \{0,1\}^{\nu_i} \to \{0,1\}^{tn_i}$ *where* $\nu_i = \sum_{j=1}^{i} n_j$*. Then* TCode *is called an* $(t, \mathsf{tg}, \ell, k, n)$-one-many tag-based-block-wise non-malleable code *(*OMTBC *in short) if for any such tuple* $\widehat{\mathbf{f}}$ *there exists a replacer* $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ *such that for any pair of messages* $(m_0, m_1) \in \{0,1\}^k$*, the following holds:*

$$\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_0} \approx \mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_1}$$

where $\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m}$ *for any* $m \in \{0,1\}^k$ *is defined as:*

$$\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m} = \left\{ \begin{array}{c} \mathbf{c} = (c_1, \ldots, c_\ell) \leftarrow \mathsf{TEnc}(m);\ \forall i \in [\ell] : (c'_{i,1}, \ldots, c'_{i,t}) = \widehat{f}_i(c_1, \cdots, c_i); \\ \underline{\forall\ j \in [t]\ \textit{do as follows}:} \\ \left\{ \begin{array}{c} \textit{Let } \mathbf{c}'_j = (c'_{1,j}, \ldots, c'_{\ell,j});\ \textit{If } \mathsf{tg}(\mathbf{c}'_j) = \mathsf{tg} \textit{ then set } m'_j := \mathsf{same}^\star; \\ \textit{else decode } m'_j \leftarrow \mathsf{TDec}(\mathbf{c}'_j); \textit{If } m'_j = \perp \textit{ then } m'_j \leftarrow \widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}(j, c_1, \ldots, c_\ell); \end{array} \right\}; \\ \textit{Output } \mathbf{m}' = (m'_1, \ldots, m'_t) \end{array} \right\}.$$

Remark 5.9. *Note that this definition is similar to one-many non-malleable commitments [33]. In this definition the i-th tampering function's range is t times the size of the i-th block. In other words, we allow the tampering function to output t codewords. Also note that the replacer, which*

[14] In [8] Chattopadhyay et al. introduced the notion of one-many non-malleable code which is in turn built on continuous non-malleable code [20](CNMC). It is important not to confuse this notion with CNMC where the adversary chooses each subsequent tampering function after observing the result of the previous tamperings.

can be called t times, gets as input the index of the invalid codeword, and it outputs the replaced value for that codeword.

The proof is a straightforward extension of the proof of Theorem 5.6, so we omit many details.

Theorem 5.10. *Let $\mu, t \in \mathbb{N}$ be some parameter. Assume that sub-exponentially hard one-way-permutations exists. Then, for any tag $\mathsf{tg} \in [\mu]$ and any $k \in \mathbb{N}$ the $(\mathsf{tg}, \ell, k, n)$-TBC $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ described in Fig. 1 is an $(t, \mathsf{tg}, \ell, k, n)$-one-many tag-based BNMC against all PPT adversary such that $n = O(k + \mu \cdot \mathsf{p})$ and $\ell = 2\mu + 2$.*

Proof. The central ideas used in this proof are similar to that in the proof of Theorem 5.6. Again we start with description of the replacer.

$\underline{\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}(j, c_1, \ldots, c_\ell)}$:
On input an index j and a tuple $\mathbf{c} = (c_1, \ldots, c_\ell)$, the replacer first generates the t-tuple of the tampered codeword. Let $\mathbf{c}'_j = (c'_{1,j}, \ldots, c'_{\ell,j})$ be the j-th such codeword. Let $\mathsf{tg}(\mathbf{c}') = \widetilde{\mathsf{tg}}$. Then it works as follows:

1. If $\widetilde{\mathsf{tg}} = \mathsf{tg}$ then output $\mathsf{same}^\star$.
2. Otherwise first check if the structure is correct (Step-1 of decoding). If not then it outputs $\perp$.
3. Otherwise do as follows:
   (a) If $\widetilde{\mathsf{tg}} < \mathsf{tg}$, then output the message (this message is unique by perfect binding of Com) committed in the first stage of the tampered codeword by brute-force.
   (b) If $\widetilde{\mathsf{tg}} > \mathsf{tg}$, then output the message committed in the second stage of the tampered codeword by brute-force.

Now, assume that there exists a PPT adversary A which can distinguish among experiments $\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_0}$ and $\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_1}$ while running in $o(\kappa_\mathsf{s})$-time. Further, assume that A outputs a bit b while it detects the experiment to be $\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_b}$. Therefore, for a randomly chosen $b \in \{0,1\}$,

$$\Pr\left[(\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{A}) = b\right] > 1/2 + \varepsilon(\kappa_\mathsf{s}) \tag{2}$$

for some non-negligible function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{N}$ of the security parameter κ.

We prove a lemma similar to Lemma 5.7

Lemma 5.11. *If $\Pr\left[(\mathsf{OMTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{A}) = b\right] > 1/2 + \varepsilon(\kappa)$ for some non-negligible function $\varepsilon : \mathbb{N} \to \mathbb{N}$ then there exists a PPT adversary B which can break hiding of the commitment scheme Com_κ (with probability at least $\varepsilon(\kappa)$) if B is allowed to run in $O(2^{\kappa_\mathsf{s}})$ (but in $o(2^\kappa)$) time.*

Proof (Sketch). We only provide a sketch here as the proof idea is exactly the same as that of Lemma 5.7. The only difference here is that the reduction B has to simulate experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}$ which outputs a vector of t values as opposed to the single value in the earlier case. However, it is straightforward to extend the simulation from single value to a vector by treating each value in the vector individually. So, the adversary simulates the tampering experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}$ correctly, albeit using single-lvel complexity leveraging (to simulate values encoded in a codeword with larger tag) and non-uniform reduction (to simulate the value encoded in a codeword with smaller tag). $\square$

This concludes the proof of the theorem. $\square$

5.2.2 Using DDN-XOR trick

Some intuitions. In this section we use the DDN-XOR trick to construct an "efficient" TB-NMC with "large" tags. Let us start with some intuitions. The construction uses any OMTBC (called "inner code" in the following) with "small" tag in a black-box way. The basic idea is as follows: let the "big" tag TG be t-bit long. Then compute t shares of message m just using XOR's i.e. $(m_1, \ldots, m_t)$ which is nothing but a t-out-of-t secret sharing. Then encode each m_j with the inner code using $j \| \mathsf{TG}[j]$ (which is of $O(\log(t))$-size) as tag. Finally put the encodings in increasing order of j (from 1 to t). The first block of the final codeword is, by definition the tag TG. the second block would consist of the first t blocks of inner codes in order and so on. The key-intuitions why the construction works are as follows. In order to break the tag-based non-malleability (Def. 5.3) of the final encoding (called "outer code" within this sub-section), the adversary must produce a valid codeword with different"big" tag $\widetilde{\mathsf{TG}} \neq \mathsf{TG}$. In that case, evidently, there must exist at least one index $j \in [t]$ where the "small" tags differ $\widetilde{\mathsf{tg}}_j \neq \mathsf{tg}_j$. Moreover notice that, the adversary can't copy tg_j to any other position than j as that would result in an invalid codeword. Therefore $\mathsf{tg}_j = j \| \mathsf{TG}[j]$ is different from *all* the "small tags" of the tampered inner codewords. Then we reduce to the one-many non-malleability of the inner code in first such position (say $j^\star$). In particular, if the adversary tampers with the $j^\star$-th inner code, then by one-many non-malleability of the "inner code" no tampering function would not be able to succeed in producing any valid inner codeword that encodes a value which is "related" to the $j^\star$-th original share. Clearly, this implies the entire tampered outer codeword would have no information about $j^\star$-th share which makes the encoded massage (if valid) completely unrelated to the original message by the property of secret sharing.

The construction. For any tag $\mathsf{TG} \in \{0,1\}^t$ we construct a $(\mathsf{TG}, \ell', k', n')$-TBNMC $\mathsf{LCode} = (\mathsf{LEnc}, \mathsf{LDec})$ from a $(t, \mathsf{tg}, \ell, k, n)$-OMTBC $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ for any $\mathsf{tg} \in \{0,1\}^\alpha$ such that $t = 2^{\alpha-1} - 1$, $\ell' = \ell + 1$, $k' = k$ and $n' = nt$ as follows.

- **Encode** $\mathsf{LEnc}(m)$:
  1. Secret-sharing: On receiving an input message $m \in \{0,1\}^{k'}$, first choose $(t-1)$ random k'-bit strings $(m_1, \ldots, m_{t-1})$ and then compute $m_t = m \oplus m_1 \oplus \cdots \oplus m_{t-1}$. Note that the tuple $(m_1, \ldots, m_t)$ represents a (t,t)-secret sharing of m.
  2. Encode using smaller tag: Then for each $j \in t$, let the j-th "smaller" tag be $\mathsf{tg}_j = \mathtt{BIT}(j) \| \mathsf{TG}[j]$. Then compute the encoding of m_j as: $(c_{1,j}, \ldots, c_{\ell,j}) \leftarrow \mathsf{TEnc}_{\mathsf{tg}_j}(m_j)$.
  3. Constructing blocks: Define the tag-block $c_0 := \mathsf{TG}$. For all $i \in [\ell]$ define the i-th block as $c_i := (c_{i,1}, \ldots, c_{i,t})$. Output the codeword $\mathbf{c} = (c_0, \ldots, c_\ell)$.
- **Decode** $\mathsf{LDec}(\mathbf{c})$:
  1. Parsing: On receiving a codeword $\mathbf{c}$, parse it as $(c_0, \ldots, c_\ell) := \mathbf{c}$ such that $|c_0| = t$ and for all $i \in [\ell]$ $|c_i| = tn_i$. Then, for all $i \in [\ell]$ parse c_i as $(c_{i,1}, \ldots, c_{i,t})$ such that for all $j \in [t]$, $|c_{i,j}| = n_i$.
  2. Checking Tag consistency: Check if the "bigger" tag is consistent with the "smaller" tag: $c_0 = c_{1,1}[\alpha] \| c_{1,2}[\alpha] \| \cdots \| c_{1,t}[\alpha]$. Also check if the positions of the smaller tags are correct: $\forall\, j \in [t]$, $c_{1,j}[1 \ldots (\alpha - 1)] = \mathtt{BIT}(j)$. If any of these fail output $\perp$, otherwise go to the next step.

3. Decoding with smaller tag: For each $j \in [t]$ decode each value $v_j \leftarrow \mathsf{TDec}_{\mathsf{tg}_j}(c_{1,j}, \ldots, c_{\ell,j})$. If any of them is $\perp$ then output $\perp$. Otherwise, parse each v_j as m_j and finally output $m = m_1 \oplus \cdots \oplus m_t$.

Theorem 5.12. *Let* $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ *be a* $(t, \mathsf{tg}, \ell, k, n)$*-OMTBC for any tag* $\mathsf{tg} \in \{0,1\}^\alpha$*,* $t = 2^{\alpha-1} - 1$ *and* $k \in \mathbb{N}$*. Then for any tag* $\mathsf{TG} \in \{0,1\}^t$ *the above construction* $\mathsf{LCode} = (\mathsf{LEnc}, \mathsf{LDec})$ *is a* $(\mathsf{TG}, \ell', k', n')$*-TBNMC for* $\ell' = \ell + 1$*,* $k' = k$ *and* $n' = nt$

Proof. To show that LCode is a TBNMC, for any tampering function tuple $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_{\ell'})$ such that $\forall i \in [\ell']$, $\widehat{f}_i : \{0,1\}^{\nu'_i} \to \{0,1\}^{n'_i}$ where $\nu'_i = \sum_{j=1}^{i} n'_j$, we need to show the existence of a replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ such that, for any pair of messages (m_0, m_1), the experiments $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_0}$ and $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_1}$ are indistinguishable for any PPT adversary. Below we start with the description of the replacer. Note that $n_1 = \alpha$.

$\underline{\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}(c_0, \ldots, c_\ell)\text{:}}$ The replacer takes the following steps in order.

1. Set $\mathsf{TG} := c_0$. Compute $c'_0 = \widetilde{\mathsf{TG}} = \widehat{f}_1(\mathsf{TG})$. If $\widetilde{\mathsf{TG}} = \mathsf{TG}$, then output $\mathsf{same}^\star$. Otherwise go to the next step.
2. Check if all the "smaller" tags are consistent with the "big" tag post tampering of the first block. In other words, compute $c'_1 = \widehat{f}_2(c_1)$. Parse $(c'_{1,1}, \ldots, c'_{1,t}) := c'_1$ such that, for all $j \in t$, $|c_{1,j}| = \alpha + 1$. Set $\widetilde{\mathsf{tg}}_j := c'_{1,j}$ for all $j \in [t]$. Now make the following two checks:
   (a) If $\forall\, j \in [t]$; $\widetilde{\mathsf{tg}}_j[1 \ldots \alpha - 1] = \mathtt{BIT}(j)$.
   (b) If $\widetilde{\mathsf{TG}} = \widetilde{\mathsf{tg}}_1[\alpha] \| \widetilde{\mathsf{tg}}_2[\alpha] \| \cdots \| \widetilde{\mathsf{tg}}_t[\alpha]$.

   If any of them fails, then output $\perp$. Otherwise, go to the next step.
3. Find the minimum index $j^\star$ for which $\widetilde{\mathsf{TG}}[j^\star] \neq \mathsf{TG}[j^\star]$.
4. Construct the tuple functions $\widetilde{\mathbf{f}} = (\widetilde{f}_1, \ldots, \widetilde{f}_\ell)$ such that $\forall i \in [\ell]$, $\widetilde{f}_i : \{0,1\}^{\nu_i} \to \{0,1\}^{n_i t}$ and each function $\widetilde{f}_i$ is defined to work as follows:
   - Has the "big" codeword $(c_0, \ldots, c_\ell)$ hardwired. Parse $(c_{i,1}, \ldots, c_{i,t}) := c_i$ for all $i \in [\ell]$ such that $|c_{i,j}| = n_i$.
   - On input a partial encoding $(\gamma_1, \ldots, \gamma_i)$, set $c_{i',j^\star} := \gamma'_i$ for all $i' \in [i]$.
   - Apply $\widehat{f}_{i+1}$ to $((c_{1,1}, \ldots, c_{1,t}), \ldots, (c_{i,1}, \ldots, c_{i,t}))$ to produce c'_i.
   - Output c'_i.
5. For all $i \in [\ell]$, parse each c_i and c'_i as $(c_{i,1}, \ldots, c_{i,t}) := c_i$ and $(c'_{i,1}, \ldots, c'_{i,t}) := c'_i$ such that for all $j \in [t]$, $|c_{i,j}| = |c'_{i,j}| = n_i$, respectively.
6. Decode $v_j := \mathsf{Dec}(c'_{1,j}, \ldots, c'_{\ell,j})$ for all $j \in [t]$. If $v_j = \perp$ run the one-many replacer $v_j \leftarrow \widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}(j, c_{1,j^\star}, \ldots, c_{\ell,j^\star})$. Here, we use the fact that the underlying code is one-many tag-based BNMC and hence there exists such a replacer.
7. If $\exists j \in [t]$ such that $v_j = \perp/\mathsf{same}^\star$, then output $\perp$.

8. Output $v_1 \oplus \cdots \oplus v_t$.

Next we will prove that, for the above replacer, the experiments are indistinguishable. In particular we reduce to the one-many non-malleability of TCode. Formally, we prove the following lemma.

Lemma 5.13. *Assume that there exists a* PPT *adversary* A*, a pair of messages* (m_0, m_1) *and a tuple of functions* $\widehat{\mathbf{f}}$ *for which we have, for a random bit* $b \in \{0,1\}$,

$$\Pr\left[(\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{A})\right] > 1/2 + \varepsilon(\kappa) \tag{3}$$

for some non-negligible function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{N}$*. Then there exists a pair of messages* (m'_0, m'_1)*, a function tuple* $\widehat{\mathbf{f}}$ *and a PPT adversary* B *such that the following holds for a random bit* b*:*

$$\Pr\left[(\mathsf{OMTamper}^{\widetilde{\mathbf{f}}}_{m'_b} \leftrightarrows \mathsf{B}^{\mathsf{A}})\right] > 1/2 + \varepsilon(\kappa) \tag{4}$$

Proof. We describe the adversary B as follows:

<u>Adversary B^{A}.</u>

On receiving the message pair (m_0, m_1) and the tuple of tampering functions $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_{\ell'})$ from A, the adversary B^{A} takes the following steps in order.

1. Computes the tampered tag $\widetilde{\mathsf{TG}}$ as $\widetilde{\mathsf{TG}} = \widehat{f}_1(\mathsf{TG})$ If $\widetilde{\mathsf{TG}} = \mathsf{TG}$, then return $\mathsf{same}^\star$ to A. Otherwise go to the next step.

2. Check if the smaller tags are consistent after tampering in exactly the same way as the replacer does:

   (a) Construct the second block as $c_1 := (\mathsf{tg}_1, \ldots, \mathsf{tg}_t)$ where $\mathsf{tg}_j = \mathtt{BIT}(j) \| \mathsf{TG}[j]$ for all $j \in [t]$. It computes the second tampered block $c'_1 = \widehat{f}_2(c_0, c_1)$.

   (b) Parse $(c'_{1,1}, \ldots, c'_{1,t}) := c'_1$ such that for all $j \in t$, $|c_{1,j}| = \alpha$.

   (c) For all $j \in [t]$, set $\widetilde{\mathsf{tg}}_j := c'_{1,j}$. Now check if $\forall\, j \in [t]$; $\widetilde{\mathsf{tg}}_j[1 \ldots (\alpha - 1)] = j$. If any of them fails then return $\perp$ to A. Otherwise go to the next step.

3. Find the minimum index $j^\star$ for which $\widetilde{\mathsf{TG}}[j^\star] \neq \mathsf{TG}[j^\star]$, then follows the following steps:

   (a) Choose $t-1$ random values $m^{(j)} \in \{0,1\}^{k'}$ for all $j \in [t] \setminus \{j^\star\}$. Compute the messages (m'_0, m'_1) as $m'_b := m^{(1)} \oplus \cdots \oplus m^{(j^\star-1)} \oplus m_b \oplus m^{(j^\star+1)} \cdots m^{(t)}$ $(b \in \{0,1\})$.

   (b) For all $j \in [t] \setminus \{j^\star\}$, encodes $m^{(j)}$ to produce encodings $\mathbf{c}_j = (c_{1,j}, \ldots, c_{\ell,j}) \leftarrow \mathsf{Enc}(m^{(j)})$ with tags $\mathtt{BIT}(j) \| \mathsf{TG}[j]$.

4. Define the tampering function tuple $(\widetilde{f}_1, \ldots, \widetilde{f}_\ell)$ as follows:

   - Each $\widetilde{f}_i : \{0,1\}^{\nu_i} \to \{0,1\}^{tn_i}$ is hardwired with the values $(\mathbf{c}_1, \ldots, \mathbf{c}_{j^\star-1}, \mathbf{c}_{j^\star+1}, \ldots, \mathbf{c}_t)$ and the tag TG.
   - On input $(c_1, \ldots, c_i)$, set $c_{i',j^\star} := c_{i'}$ for all $i' \in [i]$.
   - Then apply the function $\widehat{f}_{i+1} : \{0,1\}^{\nu'_i} \to \{0,1\}^{n'_i}$ on the tuple $(\mathsf{TG}, (c_{1,1}, \ldots, c_{1,t}), \ldots, (c_{i,1}, \ldots, c_{i,t}))$ to produce the tampered codeword $(c'_{i,1}, \ldots, c'_{i,t})$

- Output the tuple $(c'_{i,1}, \ldots, c'_{i,t})$

5. B outputs the pair (m_0, m_1) as messages to be challenged upon by the challenger of experiment $\mathsf{OMTamper}^{\widetilde{\mathbf{f}}}$ with respect to the tag $\mathsf{tg} = j^\star \| \mathsf{TG}[j^\star]$ with the tampering functions $\widetilde{\mathbf{f}} = (\widetilde{f}_1, \ldots, \widetilde{f}_\ell)$ described above.

6. On receiving a tuple $(v_1, \ldots, v_t)$ as the response from the experiment $\mathsf{OMTamper}^{\widetilde{\mathbf{f}}}_{m_b}$, B executes the following steps similar to the replacer.

   (a) If there exists a $v_j = \perp/\mathsf{same}^\star$, then return $\perp$ to A.

   (b) Otherwise, set $\widetilde{m}_j := v_j$ then return $\widetilde{m}_1 \oplus \ldots \oplus \widetilde{m}_t$ to A.

7. Finally output whatever A outputs as its decision.

In order to complete the proof, we need to argue that the above reduction perfectly simulates the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ to A. To do this, we split the analysis into several cases.

- $\mathsf{TG} = \widetilde{\mathsf{TG}}$*:* Here the simulation is trivially perfect because A expects $\mathsf{same}^\star$ irrespective of anything.

- $\mathsf{TG} \neq \widetilde{\mathsf{TG}}$*:* This case is more involved and we split again in the following sub-cases:

  – *Tag consistency fails:* This is a structural inconsistency. In this case A decides to tamper to something invalid as soon as in the second tampering even without having any information about the input. Clearly, in this case, the decoder would output $\perp$ which can not depend on the input. So, B returns $\perp$. Note that also the replacer does the same.

  – *Tag consistency succeeds:* This case is more involved. We present the steps the reduction follows in this case below:

    1. B first chooses its own challenge messages (m_0, m_1) just by forwarding the challenge messages output by A and tampering functions $\widetilde{\mathbf{f}}$ (one-many) depending on the tampering functions (one-one) chosen by A, respectively. Importantly, it chooses the tag to be the tag $\mathsf{tg}_{j^\star}$ because we want this to different from all the possible tampered tags. This will be helpful later. It is easy to see why this is the case: (i) Note that $j^\star$ is the index where $\widetilde{\mathsf{TG}}[j^\star] \neq \mathsf{TG}[j^\star]$, whence clearly $\mathsf{tg}_{j^\star} = \mathtt{BIT}(j^\star) \| \mathsf{TG}[j^\star] \neq \mathtt{BIT}(j^\star) \| \widetilde{\mathsf{TG}}[j^\star] = \widetilde{\mathsf{tg}}_{j^\star}$; and (ii) Since we are already in the case where the tags are consistent, and each of the tag $\widetilde{\mathsf{tg}}_j$ has their corresponding position j as a prefix.
    2. Next note that, the one-many challenger here receives two messages (m_0, m_1) as the challenge messages. Then it picks a bit $b \in \{0,1\}$ randomly and encodes m_b and tamper with functions $\widetilde{\mathbf{f}} = (\widetilde{f}_1, \ldots, \widetilde{f}_\ell)$. Each function $\widetilde{f}_i$ is hardwired with the encodings of all shares except the $j^\star$-th one which it gets as input. Then it "simulates" an partial encoding $\mathsf{LEnc}(m_b)$ of m_b with respect to tag TG, feed that to the tampering function $\widehat{f}_{i+1}$ and outputs whatever it outputs. Eventually, a tuple of tampered codeword is generated by such tampering. Let $(v_1, \ldots, v_t)$ be the decodings of the tampered codewords. Now, recall that all the tampered tags are different from the input tag $j^\star$. Hence no v_j will be equal to $\mathsf{same}^\star$. At this point there are two possible scenarios:

* $\forall\ j\ \in [t],\ v_j \neq \perp$*:* In this case, the replacer $\widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}$ won't be invoked in the experiment $\mathsf{OMTamper}^{\widetilde{\mathbf{f}}}_{m_b}$. Therefore, the experiment just outputs these values. B on receiving them can easily finish the rest of decoding process itself. Clearly B perfectly simulates the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ to A.
* $\exists\ j\ \in [t]$ *such that* $v_j \neq \perp$*:* In this case, the one-many replacer $\widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}$ would come into play. First note that the decoding for the code corresponding to the "big" tag would also result in $\perp$; thereby, invoking the replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ in the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$. Now, the job of the reduction is to simulate the behaviour of $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ consistently when we are in this case. To see this, recall the construction of $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$. The replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ is constructed in a manner that it uses the one-many replacer $\widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}$ internally. This is the key-fact that allows the successful simulation. First note that we are already in the case where the tag-consistency succeeds during Step-3 in the description of $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$. So, at this stage $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ constructs the function-tuple $\widetilde{\mathbf{f}}$ which outputs the tampered "big" encoding and run the one-many replacer $\widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}$ with that with the $j^{\star}$-th encoding as input. Now once $\widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}$ replaces any value with $\perp$, $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ also outputs $\perp$; otherwise, it finishes the rest of the decoding. On the other hand, in the experiment $\mathsf{OMTamper}^{\widetilde{\mathbf{f}}}_{m_b}$, the replacer gets the encoding $\mathsf{TEnc}(m_b)$ as input and then replaces the $\perp$ with some value. Now B gets a tuple of values which are possibly replaced by $\widetilde{\mathbf{R}}_{\widetilde{\mathbf{f}}}$. Again, if one of them is $\perp$ B outputs $\perp$ and otherwise finishes the rest of the decoding. Hence, clearly B simulates the environment of $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ even when replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ is invoked.

Since the above cases are exhaustive and in all of them the adversary B can simulate the view of A in experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ perfectly for a random b, we can conclude that the success probability of B is at least equal to the success probability of A which concludes the proof.

□

This concludes the proof of the theorem.

□

5.3 The full construction by removing tags

Finally we present a transformation to remove tags using one-time signature scheme and a tag-based code with "large tag" (will be referred to as "inner code" in this section). This is similar to a standard trick [16] used in the area of non-malleable commitment for the same purpose. The main idea is to sign the entire codeword and set the public-key as the tag. This forces the tampering function either to keep the tag same and forge the signature in order to tamper, otherwise change the tag by producing its own key-pairs and then tamper. But the "inner code" guarantees that whenever the tag is changed, the tampering would result in an "unrelated" codeword.

The Transformation. Let $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ be an $(\mathsf{tg}, \ell, k, n)$-TBNMC for any tag $\mathsf{tg} \in \{0,1\}^t$. Let $\mathsf{OTSig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ be a one-time signature scheme with public key $pk \in$

$\{0,1\}^t$ which takes any $k_m = n - t$-bit message to produce a n_{s}-bit signature. Then we construct an $(\ell, k, n + n_{\mathsf{s}})$-BNMC $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ as follows:

- **Encode** $\mathsf{Enc}(m)$:
  1. Generate signature keys: On input message $m \in \{0,1\}^k$ first run the key-generation algorithm of the signature scheme OTSig to generate a key pair: $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$.
  2. Encode with tag: Run the tag-based encoding scheme with pk as the tag on the input message m to produce the codeword $(\tilde{c}_1, \ldots, \tilde{c}_\ell) \leftarrow \mathsf{TEnc}(m)$. Note that $\tilde{c}_1 = pk$.
  3. Sign the codeword: Sign the codeword (except the tag) $(\tilde{c}_2, \ldots, \tilde{c}_\ell)$ to compute the signature $\sigma \leftarrow \mathsf{Sign}(sk, (\tilde{c}_2, \ldots, \tilde{c}_\ell))$.
  4. Output: Set for all $i \in [\ell - 1]$, $c_i = \tilde{c}_i$ and $c_\ell = \tilde{c}_\ell \| \sigma$. Output the codeword $\mathbf{c} = (c_1, \ldots, c_\ell)$
- **Decode** $\mathsf{Dec}(c_1, \ldots, c_\ell)$:
  1. Parse: On input the codeword $(c_1, \ldots, c_\ell)$, set $\forall i \in [\ell - 1]$, $\tilde{c}_i := c_i$ and parse c_ℓ as $(\tilde{c}_\ell \| \sigma) := c_\ell$ such that $|\tilde{c}_\ell| = n_\ell$ and $|\sigma| = n_{\mathsf{s}}$.
  2. Verify signature: Then verify the signature $d \leftarrow \mathsf{Verify}(\tilde{c}_1, (\tilde{c}_2, \ldots, \tilde{c}_\ell), \sigma)$. If $d = 0$ (i.e. verification fails) then output $\perp$. Otherwise go to the next step.
  3. Decode with tag: Decode the codeword as $\widetilde{m} \leftarrow \mathsf{TDec}(\tilde{c}_1, \ldots, \tilde{c}_\ell)$. Output $\widetilde{m}$.

Next we prove that the above construction is a BNMC.

Theorem 5.14. *Let* $\mathsf{TCode} = (\mathsf{TEnc}, \mathsf{TDec})$ *be a* $(\mathsf{tg}, \ell, k, n)$*-TBNMC for any tag* $\mathsf{tg} \in \{0,1\}^t$ *and* $\mathsf{OTSig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a one-time signature scheme with public key* $pk \in \{0,1\}^t$ *which takes any* $k_m = n - t$*-bit message to produce a* n_{s}*-bit signature. Then the above construction* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *is a* (ℓ', k', n')*-BNMC for* $\ell' = \ell$, $k' = k$ *and* $n' = n + n_{\mathsf{s}}$

Proof. Without loss of generality assume that, for all valid tag-based codeword $(\tilde{c}_1, \ldots, \tilde{c}_\ell)$, $\tilde{c}_\ell \neq 1^{n_\ell}$. For any given tampering function tuple $\mathbf{f} = (f_1, \ldots, f_\ell)$ for experiment $\mathsf{BLTamp}^{\mathbf{f}}$ we construct a corresponding function-tuple $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$, such that, for all $i \in [\ell]$ $\widehat{f}_i : \{0,1\}^{\nu_i} \to \{0,1\}^{n_i}$ and each such $\widehat{f}_i$ is same as f_i, except the last function f_ℓ. The function $\widehat{f}_\ell$ is hardwired with the signing key sk. On input $(\tilde{c}_1, \ldots, \tilde{c}_\ell)$, $\widehat{f}_\ell$, executes the following steps:

- First compute the signature $\sigma \leftarrow \mathsf{Sign}(sk, (\tilde{c}_2, \ldots, \tilde{c}_\ell))$ and then concatenate σ with the input to produce $(c_1, \ldots, c_\ell)$ where $\forall\ i \in [\ell - 1]$, $c_i = \tilde{c}_i$ and $c_\ell = \tilde{c}_\ell \| \sigma$.
- Then run f_ℓ on $(c_1, \ldots, c_\ell)$ to produce $c'_\ell \in \{0,1\}^{n_\ell + t}$.
- Then it checks if that verifies by running $\mathsf{Verify}(c'_1, (c'_2, \ldots, c'_\ell), \sigma')$.
- If that fails then it outputs 1^{n_ℓ} (trigger an invalid tag-based codeword); otherwise, it outputs $c'_\ell[1 \ldots n_\ell]$.

For any given pair of messages (m_0, m_1) and a function tuple $\mathbf{f} = (f_1, \ldots, f_\ell)$ we construct the replacer for experiment $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$ $(b \in \{0,1\})$ as follows:

<u>Replacer $\mathbf{R}_{\mathbf{f}}(c_1, \ldots, c_\ell)$:</u>

1. On receiving a codeword, it first computes the tampered codeword $(c'_1, \ldots, c'_\ell)$ by applying the tampering functions $(f_1, \ldots, f_\ell)$ on $(c_1, \ldots, c_\ell)$.

2. Set $\forall\ i \in [\ell - 1]$, $\tilde{c}_i := c_i$ and $\tilde{c}'_i := c'_i$. Notice that, $pk = c_1$ and $pk' = c'_1$. Then parse c_ℓ as $\tilde{c}_\ell \| \sigma := c_\ell$ and c'_ℓ as $\tilde{c}'_\ell \| \sigma' := c'_\ell$ such that $|\tilde{c}_\ell| = |\tilde{c}'_\ell| = n_\ell$ and $|\sigma| = |\sigma'| = n_{\mathsf{s}}$.

3. If $pk = pk'$ then output $\mathsf{same}^\star$.

4. Otherwise, run the tag-based replacer $\widetilde{m} \leftarrow \widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}(\tilde{c}_1, \ldots, \tilde{c}_\ell)$ where the functions $\widehat{\mathbf{f}}$ constructed above and outputs $\widetilde{m}$.

Next we prove that for the above replacer the experiments $\mathsf{BLTamp}^{\mathbf{f}}_{m_0}$ and $\mathsf{BLTamp}^{\mathbf{f}}_{m_1}$ are computationally close. Let us first present the experiment $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$ in detail ($b \in \{0, 1\}$) adjusted to our construction.

$\underline{\mathsf{BLTamp}^{\mathbf{f}}_{m_b}}$

1. Encode:

   (a) Generate the signing keys: $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$.

   (b) Apply the tag-based code with pk as the tag: $(\tilde{c}_1, \ldots, \tilde{c}_\ell) \leftarrow \mathsf{TEnc}(m_b)$. Note that, $\tilde{c}_1 = pk$.

   (c) Compute the signature: $\sigma \leftarrow \mathsf{Sign}(pk, (\tilde{c}_2, \ldots, \tilde{c}_\ell))$.

   (d) Form the codeword by appending the signature: $\forall\ i \in [\ell - 1]$ $c_i := \tilde{c}_i$ and $c_\ell := \tilde{c}_\ell \| \sigma$

2. Tamper: $\forall i \in [\ell] : c'_i = f_i(c_1, \ldots, c_i)$. Set $pk' := c'_1$

3. Decode:

   (a) If $(c'_1, \ldots, c'_\ell) = (c_1, \ldots, c_\ell)$ then set $m' := \mathsf{same}^\star$.

   (b) Else parse $\forall\ i \in [\ell - 1]$, $\tilde{c}'_i := c'_i$ and $\tilde{c}'_\ell := c_\ell[1 \ldots n_\ell]$, $\sigma' := c_\ell[n_\ell + 1 \ldots n_\ell + n_{\mathsf{s}}]$. Verify the signature: $d \leftarrow \mathsf{Verify}(pk', (\tilde{c}'_2, \ldots, \tilde{c}'_\ell), \sigma')$ if $d = 0$ set $m' := \perp$.

   (c) If verification fails, then decode: $m' \leftarrow \mathsf{TDec}(\tilde{c}'_1, \ldots, \tilde{c}'_\ell)$.

   (d) If $m' = \perp$ then call the replacer $m' \leftarrow \mathbf{R}_{\mathbf{f}}(c_1, \ldots, c_\ell)$.

   (e) Output m'.

Let Forge be the event defined below for which the simulation will not be correct.

- Forge happens whenever the following happens in $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$:

  1. The public key is not changed: $pk' = pk$.
  2. The codeword is not copied: $\mathbf{c}' \neq \mathbf{c}$
  3. The signature verifies in Step 3b while decoding: $\mathsf{Verify}(pk', (\tilde{c}'_2, \ldots, \tilde{c}'_\ell), \sigma') = 1$

First, assume for the sake of contradiction that there is a PPT adversary A, a pair of messages (m_0, m_1), and a tuple of functions $\mathbf{f} = (f_1, \ldots, f_\ell)$ such that the following holds for a randomly chosen $b \in \{0, 1\}$:

$$\Pr\left[(\mathsf{BLTamp}^{\mathbf{f}}_{m_b} \leftrightarrows \mathsf{A}) = b\right] > 1/2 + \varepsilon(\kappa) \tag{5}$$

for some non-negligible functions $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{N}$. Now we describe a PPT adversary (reduction) B^{A} for the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ as follows:

Reduction B^{A}

1. Receive the messages (m_0, m_1) and the tampering functions $\mathbf{f} = (f_1, \ldots, f_\ell)$ from A.
2. Sample a pair of signature keys $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$.
3. Check if $f_1(pk) = pk$.
   (a) If yes then return $\mathsf{same}^\star$ to A.
   (b) Otherwise construct the function tuple $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$ as described above. Send the messages (m_0, m_1) and the tampering functions $\widehat{\mathbf{f}}$ to its challenger.
   (c) Receive a value $\widetilde{m}$ from the challenger. Return $\widetilde{m}$ to A
4. Receive the decision bit from A and output that bit as its decision.

In order to succeed in experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$, B needs to simulate the view of A in the experiment perfectly. However, if FORGE happens, then B would return $\mathsf{same}^\star$ to A, whereas the experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$ would return the decoding of $\mathbf{c}'$. But in that case A produces an existential forgery of a new value $\mathbf{c}'$ without knowing the secret-key. Hence, by the unforgeability of the signature scheme we have that $\Pr[\textsc{Forge}] \leq negl(\kappa)$, and using Eq. 5, we have:

$$\Pr\left[(\mathsf{BLTamp}^{\mathbf{f}}_{m_b} \leftrightarrows \mathsf{A}) = b\right] \leq \Pr\left[(\mathsf{BLTamp}^{\mathbf{f}}_{m_b} \leftrightarrows \mathsf{A}) = b | \neg\textsc{Forge}\right] + \Pr\left[\textsc{Forge}\right].$$

Clearly,

$$\Pr\left[(\mathsf{BLTamp}^{\mathbf{f}}_{m_b} \leftrightarrows \mathsf{A}) = b | \neg\textsc{Forge}\right] > 1/2 + \varepsilon'(\kappa) \tag{6}$$

for some non-negligible function $\varepsilon'(\cdot) : \mathbb{N} \to \mathbb{N}$.

Next we argue that, when FORGE does not happen, then B is able to simulate the view of A perfectly. We argue case-by-case as follows:

1. $pk' = pk$: In this case, B returns $\mathsf{same}^\star$. Now, since the event FORGE does not happen, we must have one of the following:
   (a) $\mathbf{c}' = \mathbf{c}$: In this case, $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$ would have returned $\mathsf{same}^\star$.
   (b) $\mathbf{c}' \neq \mathbf{c}$: In this case, the verification would fail, which implies that the replacer $\mathbf{R_f}$ would be invoked in the experiment $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$. Notice that in this case, $\mathbf{R_f}$ would output $\mathsf{same}^\star$.
2. $pk' \neq pk$: In this case B outputs whatever the challenger returns in experiment $\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b}$. The challenger runs the set of functions $\widehat{\mathbf{f}} = (\widehat{f}_1, \ldots, \widehat{f}_\ell)$. From the description of functions, it is easy to see that it produces exactly the same codeword as $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$ until $\ell - 1$ blocks. Depending on the tampering of the last block, we have the following two scenarios.

(a) $\widehat{f}_\ell$ checks the validity of the signature. If it fails, then it outputs all 1 string, triggering an invalid codeword for TCode. In this case, the replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ would be invoked, However, recall the construction of $\mathbf{R}_{\mathbf{f}}$, which in this case also invokes the replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$. Hence the output returned by the challenger would be identically distributed with the output of $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$.

(b) On the other hand, if the signature remains valid, then there are two more cases:

Case 1: The inner encoding (tag-based) is valid. In this case the decoding of that inner codeword will be received by B. From the decoding algorithm Dec it is easy to see that the experiment $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$ would also respond with the decoded value of the inner-encoding.

Case 2: The inner encoding is invalid. In this case, the challenger calls the replacer $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$ and return the possibly replaced value to B. On the other hand in $\mathsf{BLTamp}^{\mathbf{f}}_{m_b}$ the replacer $\mathbf{R}_{\mathbf{f}}$ would be invoked and then this replacer will in turn call $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$, and return the value output by $\widehat{\mathbf{R}}_{\widehat{\mathbf{f}}}$.

Hence in all the cases when $pk \neq pk'$ it is fine to return the value returned by the challenger

So, we have,

$$\begin{aligned}\Pr\left[(\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{B}) = b\right] &\geq \Pr\left[(\mathsf{TBTamper}^{\widehat{\mathbf{f}}}_{m_b} \leftrightarrows \mathsf{B}) = b | \neg\textsc{Forge}\right] \Pr[\neg\textsc{Forge}] \\ &= \Pr\left[(\mathsf{BLTamp}^{\mathbf{f}}_{m_b} \leftrightarrows \mathsf{A}) = b | \neg\textsc{Forge}\right] \Pr[\neg\textsc{Forge}] \quad (7) \\ &> \varepsilon'(\kappa)(1 - negl(\kappa)) = \varepsilon''(\kappa) \quad (8)\end{aligned}$$

for some non-negligible function $\varepsilon''(\cdot) : \mathbb{N} \to \mathbb{N}$. In the above set of inequalities, (7) follows from the above argument that when $\textsc{Forge}$ does not happen then B can simulate the view of A perfectly and Eq. 8 follows from Eq. 6 and the fact that $\Pr[\textsc{Forge}] = negl(\kappa)$ for some negligible function.

This concludes the proof.

□

5.4 Putting things together with instantiations and parameters

Finally we put everything together with concrete instantiations. Recall that κ is the security parameter and k, n, ℓ denotes message-length, codeword-length and number of blocks respectively. Also, recall the fixed polynomial $\mathsf{p}(\cdot) : \mathbb{N} \to \mathbb{N}$ which specifies the output length of the commitment scheme. First fix the parameters of Theorem 5.10 by setting $t = O(\kappa^{2+\varphi})$ and $\mu = 2t + 1$ for any arbitrary constant $\varphi > 0$ of our choice. Then by Theorem 5.10, our construction (Fig. 1) is a $(t, \mathsf{tg}, \ell, k, n)$-OMTBC for any $k \in \mathbb{N}$ and any tag $\mathsf{tg} \in [\mu]$ such that $\ell = 2\mu + 2 = O(\kappa^{2+\varphi})$ and $n = O(k + \mu\mathsf{p})$. Now if we use the "generic" perfectly binding commitment scheme from OWP (via hardcore-bit) then we get $\mathsf{p}(\kappa) = \kappa k$. Putting that we get $n = O(k(1 + \kappa^{3+\varphi}))$. Now, based on that, by Theorem 5.12 we obtain an explicit construction of $(\mathsf{TG}, \ell', k', n')$-TBNMC for any tag $\mathsf{TG} \in \{0,1\}^t$ of size $t = O(\kappa^{2+\varphi})$ such that $k' = k \in \mathbb{N}$, $\ell' = \ell + 1 = O(\kappa^{2+\varphi})$ and $n' = nt = O(k\kappa^{6+\varphi})$.

In the final construction we use a one-time signature scheme with verification key of size $|pk| = t = O(\kappa^{2+\varphi})$. In particular, we can use Lamport's signature [27] with *hash list* (in order to make the public-key short) using a *universal one-way hash function* (UOWHF). Naor and Yung [31] showed that such UOWHF can be built from any OWP. Using parameters from [31] we get that $|pk|$ of such OTS must be $\Omega(\kappa^2 \log(|m|))$ (which is essentially the size for a succinct description of such hash function)[15] where $|m|$ is the length of message to sign. In our case, from Theorem 5.14, we get the message (to be signed) is of size $O(n') = O(k\kappa^{6+\varphi})$. Hence, we would need $|pk| = \Omega(\kappa^2 \log(k\kappa^{6+\varphi}))$. Therefore, our setting of parameters which resulted $|pk| = O(\kappa^{2+\varphi})$ suffices for Theorem 5.14 to hold.

Finally by Theorem 5.14 we can construct a (ℓ'', k'', n'')-BNMC for $k'' = k' \in \mathbb{N}$, $\ell'' = \ell' = O(\kappa^{2+\varphi})$ and $n'' = n' + n_{\mathsf{s}} = O(k\kappa^{6+\varphi})$, where n_{s} is the bit-length of signature produced which will be of the order $O(k\kappa)$ (again according to parameters from [27]).

Combining Theorem 5.10, Theorem 5.12 and Theorem 5.14 we can state the following theorem which is our main result.

Theorem 5.15. *Assume the existence of sub-exponentially hard one-way permutations. Then for any $\varphi > 0$ of our choice, and any $k \in \mathbb{N}$ there exists an explicit construction of (ℓ, k, n)*-BNMC *such that $\ell = O(\kappa^{2+\varphi}), n = O(k\kappa^{6+\varphi})$.*

More generically we can state the following

Corollary 5.16. *Assume the existence of sub-exponentially hard OWP. Then for any arbitrary constant $\varphi > 0$ of our choice there exists an explicit construction of NMCwR for class $\mathcal{F}^{\ell}_{\mathsf{block}}$ for any $\ell = \Omega(\kappa^{2+\varphi})$.*

Moreover, we can conclude the following corollary about the rate of our codes.

Corollary 5.17. *One can observe that the rate of our constructions are (inverse of) polynomial in security parameter, in particular the* BNMC *construction has rate $\approx O(1/\kappa^6)$.*

6 Connection to Non-malleable Commitment

In this section, we discuss connections between BNMC and a well-known non-malleability notion namely Non-malleable Commitment. In particular we show that given a BNMC, it is possible to construct a non-malleable commitment scheme with respect to opening (NMCom) in a black-box manner. Moreover, the case of it is possible to construct a 2-block BNMC from any non-interactive NMCom scheme. Combining the above two it can be concluded that when $\ell = 2$ then BMNC is actually equivalent to perfectly binding commitments that are non-malleable w.r.t. opening.

6.1 Non-malleable Commitment from BNMC

We provide a simple construction of a perfectly binding non-malleable commitment scheme (w.r.t. opening) against synchronizing adversary solely from a BNMC. More concretely, given an (ℓ, k, n)-block-wise non-malleable encoding scheme $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ with reveal index ℓ, we design a commitment scheme $\langle \mathsf{C}, \mathsf{R} \rangle$ as follows: C encodes the input message to generate the codeword $\mathbf{c} = (c_1, \ldots, c_\ell)$ and sends each block c_i in the i-th round for all $i \in [\ell - 1]$ in the commitment

[15] Note that the public key actually consists of the top hash, using UOWHF, which consists of the description of hash function as well as the output of that. However, we can set the output length to be $O(\kappa^2)$ which implies that $|pk| = \Omega(\kappa^2 \log(|m|))$

Block-wise NMC: Let $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ be an (ℓ, k, n)-block-wise non-malleable encoding scheme with the reveal index ℓ.

Tag: Let $\mathsf{tg} \in \{0,1\}^{k_t}$ be the tag of the interaction.

Secret input to the committer: Message $m \in \{0,1\}^{k_m}$ such that $k_m + k_t = k$.

<u>Protocol:</u>

- **Initialize:** The committer C encodes the message concatenated with the tag:

$$(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(\mathsf{tg} \| m)$$

- **Commit:** The commitment consists of $\ell - 1$ rounds where in the i-th round C sends c_i for all $i \in [\ell - 1]$.
- **Decommit:** C sends the last block c_ℓ as decommitment all at once. The receiver R decodes the codeword $\widetilde{m} \leftarrow \mathsf{Dec}(c_1, \ldots, c_\ell)$ and output $\widetilde{m}$ as the committed value.

Figure 2: Non-malleable Commitment from BNMC.

phase. Finally, C sends the final block c_ℓ as decommitment. On receiving the final block R decodes $\mathbf{c}$. If the decoder outputs $\perp$ then R rejects, otherwise accepts the decoded message. The scheme is described in Figure 2. Note that, there is no message from R except some "acknowledgement" after each new message received.

We formally prove the following theorem:

Theorem 6.1. *Suppose there is a* BNMC *with reveal index ℓ. Then the protocol described in Fig. 2 is a $(\ell-1)$-round perfectly binding non-malleable commitment scheme with respect to opening against a synchronizing man-in-the-middle adversary.*

Proof. In order to prove the theorem we need to show three properties:

1. Perfect binding.
2. Computational hiding.
3. Non-malleability against synchronizing adversary.

Perfect binding. By Lemma 4.13 we have that Code has j-uniqueness where $j \leq \ell - 1$. Perfect binding follows in a straightforward manner from that, which guarantees that the encoded message is uniquely defined by the first $\ell - 1$ blocks of any codeword.

Computational hiding. This follows easily from the fact that Code has reveal index j which intuitively says that for any codeword $\mathbf{c} = (c_1, \ldots, c_\ell)$, the first $\ell - 1$ blocks $(c_1, \ldots, c_{\ell-1})$ reveal no information to a computationally bounded adversary about the message encoded by $\mathbf{c}$.

Non-malleability. Without any loss of generality we can assume the man-in-the middle M to be deterministic. Let tg be the tag of the commitment and z be the auxiliary input. Now for all $i \in [\ell]$, $f_i : \{0,1\}^{\nu_i} \to \{0,1\}^{n_i}$ has the tag tg, the auxiliary input z, and the code of M hardwired, and works as follows:

- Parse the input as a tuple $(c_1, \ldots, c_i)$ where $|c_j| = n_j$ for all $j' \in [i]$.
- Run $\mathsf{M}(\cdot)$ on z and $(c_1, \ldots, c_i)$ to generate the tampered value $c'_i \leftarrow \mathsf{M}\left(r_{\mathsf{M}}; (c_1, \ldots, c_i)\right)$, where r_{M} is the internal randomness of M.
- Output c'_i.

To show non-malleability, we need to show the existence of an PPT simulator S for any M. We explicitly construct such a simulator as follows:

$\underline{\mathsf{S}^{\mathsf{M}}(\mathsf{tg}, z)}$:

1. Start with committing to the message 0^k by first encoding $\mathbf{c} = (c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(\mathsf{tg} \| 0^k)$, and then setting the left commitment to $\mathsf{cmt} = (c_1, \ldots, c_{j-1})$.
2. Apply the functions $(f_1, \ldots, f_{\ell-1})$ defined above to generate the messages (in the right interaction) for the commitment $\mathsf{cmt}' = (c'_1, \ldots, c'_{\ell-1})$.
3. Finally decommit in the right by sending $c'_\ell \leftarrow f_\ell(c_1, \ldots, c_\ell)$.

Now we prove that this simulation satisfies Definition A.2. We reduce our problem to the underlying block-wise non-malleable encoding scheme. Assume for the sake of contradiction that there exists a man-in-the-middle M, an auxiliary input z and a value m such that the following holds for some non-negligible function $\varepsilon(\kappa) : \mathbb{N} \to \mathbb{N}$.

$$\Pr\left[\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z) = 1\right] - \Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m, z) = 1\right] > \varepsilon. \tag{9}$$

We describe the executions in detail below:

$\underline{\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z)}$. Let the tag of committer be tg. We split the execution into two phases: (i) commitment phase and (ii) decommitment phase.

1. Commitment Phase.
   - The committer C generates the encoding $(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(\mathsf{tg} \| m)$ and set the commitment to be $\mathsf{cmt} = (c_1, \ldots, c_{\ell-1})$. In round-$i$ of the commitment phase, it sends block c_i.
   - M has as an input the auxiliary value z. Then in the $i \in [\ell - 1]$-th round, it receives a block c_i and sends a block c'_i in the right to R.
2. Decommitment Phase
   - C sends the final block c_ℓ as decommitment. M, on receiving c_ℓ, sends c'_ℓ to R.
   - R decodes $v \leftarrow \mathsf{Dec}(c'_1, \ldots, c'_\ell)$. If $v \neq \perp$ then it parses $\widetilde{\mathsf{tg}} \| \widetilde{m} := v$ and sets $\theta_{\mathsf{R}} := 1$ if and only if $\widetilde{\mathsf{tg}} \neq \mathsf{tg}$ and $\mathfrak{R}(m, \widetilde{m}) = 1$. In all other cases, it sets $\theta_{\mathsf{R}} := 0$.

Finally output θ_{R}.

$\underline{\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m, z)}$. Let the tag of committer be tg. Like above, we again split the execution into two phases: (i) commitment phase and (ii) decommitment phase.

1. Commitment Phase.
   - The simulator $\mathsf{S}^{\mathsf{M}}(\mathsf{tg}, z)$, described above produces the transcript of the right interaction $(c'_1, \ldots, c'_\ell)$ by first encoding $(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(\mathsf{tg} \| 0^{k_m})$ and applying functions $\mathbf{f}$, described above, on them accordingly. In round-i of the commitment phase, it sends a c'_i to R.
2. Decommitment Phase
   - S gets the message m. However, without using m, it sends the final block c'_ℓ produced earlier to R.
   - R decodes $v \leftarrow \mathsf{Dec}(c'_1, \ldots, c'_\ell)$. If $v = \perp$ then the replacer is invoked $v \leftarrow \mathbf{R}_{\mathbf{f}}(c_1, \ldots, c_\ell)$. Now if $v = \perp$ then R sets $\theta_{\mathsf{R}} := 0$. Otherwise it parses $\widetilde{\mathsf{tg}} \| \widetilde{m} := v$. It sets $\theta_{\mathsf{R}} := 1$ if and only if $\widetilde{\mathsf{tg}} \neq \mathsf{tg}$ and $\mathfrak{R}(m, \widetilde{m}) = 1$ and $\theta_{\mathsf{R}} := 0$ otherwise.

Finally output θ_{R}.

We will now construct a PPT adversary A which can distinguish between the distribution $\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|m}$ and $\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|0^{k_m}}$ using the above man-in-the-middle M, where $\mathbf{f}$ is the function tuple defined above, with probability at least ε. Without loss of generality, we can assume that A outputs 1 when it detects the experiment to be $\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|m}$ and 0 otherwise. The description of A follows:

- It tampers the experiment with functions $\mathbf{f}$ and receives the response v from the experiment $\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|m^\star}$ (say) where $m^\star \in \{m, 0^{k_m}\}$.
- It parses v as $\widetilde{\mathsf{tg}} \| \widetilde{m}$. It outputs 1 if $\widetilde{\mathsf{tg}} \neq \mathsf{tg}$ and $\mathfrak{R}(m, \widetilde{m}) = 1$, and a random bit otherwise.

Now clearly,

$$\begin{aligned}
& \Pr\left[\mathsf{A} \text{ outputs } 1 \mid\ m^\star = m\right] - \Pr\left[\mathsf{A} \text{ outputs } 1 \mid\ m^\star = 0^{k_m}\right] \\
= \; & \Pr\left[\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|m} = \widetilde{\mathsf{tg}} \| \widetilde{m} \wedge (\widetilde{\mathsf{tg}} \neq \mathsf{tg}) \wedge (\mathfrak{R}(m, \widetilde{m}) = 1)\right] \\
& - \Pr\left[\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|0^{k_m}} = \widetilde{\mathsf{tg}} \| \widetilde{m} \wedge (\widetilde{\mathsf{tg}} \neq \mathsf{tg}) \wedge (\mathfrak{R}(m, \widetilde{m}) = 1)\right] \qquad (10) \\
\geq \; & \Pr\left[\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z) = 1\right] - \Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m, z) = 1\right] > \varepsilon. \qquad (11)
\end{aligned}$$

Eq. 10 follows from the description of adversary A in a straightforward manner since for all the other cases A outputs a random bit in both the experiments. The first inequality in Eq. 11 is more tricky. Note that, if the replacer is not invoked in the experiment $\mathsf{BLTamp}^{\mathbf{f}}_{m}$, then the equality holds clearly from the description of the executions. However, consider the case when replacer is invoked in the experiment $\mathsf{BLTamp}^{\mathbf{f}} m$ and it replaces the $\perp$ with some valid value $\widetilde{\mathsf{tg}} \| \widetilde{m}$ such that

$\mathsf{tg} \neq \widetilde{\mathsf{tg}}$ and $\mathfrak{R}(m, \widetilde{m}) = 1$. In that case, $\mathsf{BLTamp}^{\mathbf{f}}_m$ would output 1 whereas $\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z)$ would output 0, since there is no replacer in $\mathsf{Mim}^{\mathsf{M}}$. Clearly, in this case we would have

$$\begin{aligned} &\Pr\left[\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|m} = \widetilde{\mathsf{tg}}\|\widetilde{m} \wedge (\widetilde{\mathsf{tg}} \neq \mathsf{tg}) \wedge (\mathfrak{R}(m, \widetilde{m}) = 1)\right] \\ &\quad \geq \Pr\left[\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m, z) = 1\right]. \end{aligned}$$

On the other hand, since in the execution $\mathsf{Sta}^{\mathsf{S}}$ the replacer is also used the probabilities for the second quantities are always the same, i.e.,

$$\begin{aligned} &\Pr\left[\mathsf{BLTamp}^{\mathbf{f}}_{\mathsf{tg}\|0^{km}} = \widetilde{\mathsf{tg}}\|\widetilde{m} \wedge (\widetilde{\mathsf{tg}} \neq \mathsf{tg}) \wedge (\mathfrak{R}(m, \widetilde{m}) = 1)\right] \\ &\quad = \Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m, z) = 1\right]. \end{aligned}$$

The final inequality uses the assumption Eq. 9. This completes the proof.

□

Remark 6.2. *Note that our construction provided in Sec. 5 has reveal index ℓ and hence can be used here to construct non-malleable commitment scheme in the way described above. However due to the restriction on the number of blocks it will require $O(\kappa^{6+\varphi})$ rounds for any arbitrary constant $\varphi > 0$.*

6.2 BNMC from Non-malleable Commitment

In this section we show that it is possible to construct a $(2, k, n)$-BNMC from a perfectly binding *non-interactive* non-malleable commitment with respect to opening.

The construction is given below:

The construction: Let Com be a perfectly binding non-interactive non-malleable commitment scheme (w.r.t. opening) whose input is a k-bit message and output is an n-bit commitment. Let $\mathsf{OTSig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ be a one-time signature scheme which produces a signature of n_{s} bits while applied on any $(k + n_r + n)$-bit message, where n_r is the number of random bits used to generate the commitment. Then the encoding scheme is defined as follows:

- **Encode:** First generate the signing and public-key pair for the one-time signature scheme: $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$. Let $pk \in \{0,1\}^{n_{\mathsf{p}}}$ be the tag of the commitment scheme. On input message m, run the commitment algorithm with random coins $r \leftarrow \{0,1\}^{n_r}$ to produce the commitment $\mathsf{cmt} = \mathsf{Com}(m, r)$, where r is an n_r bits random number. Then produce the signature $\sigma \leftarrow \mathsf{Sign}(sk, (m, r, \mathsf{cmt}))$ of length n_{s}. The codeword consists of two parts $(c_1, c_2) = ((\mathsf{cmt}, pk), (m, r, \sigma))$. The length of the codeword is $n' = n + n_{\mathsf{s}} + n_{\mathsf{p}} + k + n_r$.

- **Decode:** On input $c \in \{0,1\}^{n'}$ parse c as a tuple $(\mathsf{cmt}, pk, m, r, \sigma)$ such that $|\mathsf{cmt}| = n$, $|pk| = n_{\mathsf{p}}$, $|m| = k$, $|r| = n_r$ and $|\sigma| = n_{\mathsf{s}}$. Then check if σ verifies as a signature of (m, r) w.r.t. the public key pk; i.e., $\mathsf{Verify}(pk, (m, r, \mathsf{cmt}), \sigma) = 1$, and the commitment and decommitment are consistent. Output $\perp$ if either of them fails and output m otherwise.

Theorem 6.3. *The above encoding scheme is a $(2, k, n')$-BNMC.*

Proof. To prove the theorem, we need to show that, for any two messages $m_0, m_1 \in \{0,1\}^k$ and any pair of tampering functions $\mathbf{f} := (f_1, f_2)$ with $f_1 : \{0,1\}^{n_1} \to \{0,1\}^{n_1}$ and $f_2 : \{0,1\}^{n'} \to \{0,1\}^{n_2}$ (let $n_1 = n + n_{\mathsf{p}}$ and $n_2 = k + n_r + n_{\mathsf{s}}$), there exists a replacer $\mathbf{R}_{f_1,f_2}$ such that the experiments defined in Def. 4.9 are computationally indistinguishable:

$$\mathsf{BLTamp}_{m_0}^{f_1,f_2} \approx \mathsf{BLTamp}_{m_1}^{f_1,f_2}.$$

The replacer $\mathbf{R}_{f_1,f_2}$ can be constructed as follows: on receiving the codeword $\mathbf{c} = (c_1, c_2) = ((\mathsf{cmt}, pk), (m, r, \sigma))$, it works as follows:

Replacer $\mathbf{R}_{f_1,f_2}$:

- First generate the tampered codeword $c_1' = f_1(c_1)$ and $c_2' = f_2(c_1, c_2)$. Parse c_1' as (cmt', pk') and c_2' as (m', r', σ'). If $c_1' = c_1$ output $\mathsf{same}^\star$. Otherwise go to the next step.
- Check if the signature verifies: $\mathsf{Verify}(pk, (m', r', \mathsf{cmt}'), \sigma) = 1$ and the commitment cmt' is consistent with the opening (m', r'). If either of them fails, then "brute-force" the unique message $\hat{m}$ corresponding to the commitment cmt'. If there is no such message, then output $\perp$; otherwise output $\hat{m}$.

We next show that for the above replacer, we can make a reduction to the non-malleability of the underlying commitment scheme. Assume for the sake of contradiction that there exists a PPT adversary A who specifies f_1 and f_2 such that A can distinguish $\mathsf{BLTamp}_{m_0}^{f_1,f_2}$ from $\mathsf{BLTamp}_{m_1}^{f_1,f_2}$ with probability more than ε for some non-negligible function $\varepsilon : \mathbb{N} \to \mathbb{N}$. Further, assume that A outputs b when it detects the message to be m_b. So formally we have for any random bit $b \in \{0,1\}$:

$$\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b\right] > 1/2 + \varepsilon. \tag{12}$$

We shall construct a man-in-the-middle M that can break the non-malleability of the commitment scheme Com for either m_0 or m_1 with respect to a polynomial relation $\mathfrak{R}$ to be defined later.

For some message $m_b \in \{m_0, m_1\}$ we recall the original tampering experiment from Def. 4.9 for this particular scheme and the above replacer:

$\underline{\mathsf{BLTamp}_{m_b}^{f_1,f_2}:}$

1. *Encode:* Generate the signing key-pair $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$. Produce the commitment: $\mathsf{cmt} \leftarrow \mathsf{Com}(m, r)$ with pk as the tag. Sign with the signing key $\sigma \leftarrow \mathsf{Sign}(sk, (m, r, \mathsf{cmt}))$. Let the codeword be $\mathbf{c} = (c_1, c_2) = ((\mathsf{cmt}, pk), (m, r, \mathsf{cmt}))$.
2. *Tamper:* Apply the tampering functions: $c_1' \leftarrow f_1(c_1)$ and $c_2' \leftarrow f_2(c_1, c_2)$. Let $\mathbf{c}' = (c_1', c_2')$.
3. *Decode:* Parse $(\mathsf{cmt}', pk') := c_1'$ and $(m', r, \sigma') := c_2'$. If $\mathbf{c}' = \mathbf{c}$ set $\widetilde{m} := \mathsf{same}^\star$,
   - Otherwise, check if the signature verifies $\mathsf{Verify}(pk', (m', r'\mathsf{cmt}'), \sigma') = 1$ and also if the commitment in cmt' and decommitment (m', r') are consistent. If not then set $\widetilde{m} := \perp$.
   - Otherwise, if both the check succeeds then set $\widetilde{m} = m'$.
4. *Replace:* If $\widetilde{m} = \perp$ call the replacer $\widetilde{m} \leftarrow \mathbf{R}_{f_1,f_2}(c_1, c_2)$.

5. *Output:* Finally output $\widetilde{m}$.

Before going into the reduction, first consider the event when f_2 tampers to some c_2' such that either of the checks fail. Let us call this event $\textsc{Fail}$. Note, that this event is simply the event that the decoder outputs $\perp$ because in no other way the codeword can be made invalid. So, in this case the replacer is invoked which then extracts the unique message $\widetilde{m}$ (uniqueness comes from the perfect binding) only from cmt' (which is independent of c_2'). On the other hand, for the same cmt' if $\textsc{Fail}$ does not happen then the experiment will the same value $\widetilde{m}$. This observation implies that $\Pr\left[\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \widetilde{m}\right]$ is completely independent of c_2' in particular whether or not $\textsc{Fail}$ happens. Hence we can have for any $\widetilde{m} \in \{0,1\}^{k_m} \cup \{\mathsf{same}^\star, \perp\}$ and any random $b \in \{0,1\}$,

$$\Pr\left[\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \widetilde{m} \mid \textsc{Fail}\right] = \Pr\left[\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \widetilde{m} \mid \neg\textsc{Fail}\right]$$

and hence we can have:

$$\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \textsc{Fail}\right] = \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail}\right].$$

So combining this with Eq. 12 we get:

$$\begin{aligned}
&\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b\right] \\
&\quad= \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \textsc{Fail}\right] \Pr\left[\textsc{Fail}\right] \\
&\qquad+ \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail}\right] \Pr\left[\neg\textsc{Fail}\right] \\
&\quad= \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail}\right] \Big(\Pr\left[\textsc{Fail}\right] + \Pr\left[\neg\textsc{Fail}\right] \Big) \\
&\quad= \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail}\right] \\
&\qquad> 1/2 + \varepsilon. \qquad (13)
\end{aligned}$$

Next consider the event when $pk' = pk$. Call this event $\textsc{TagEq}$. Then continuing from Eq. 13 we can get:

$$\begin{aligned}
&\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail}\right] \\
&= \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail} \wedge \textsc{TagEq}\right] \Pr\left[\textsc{TagEq} \mid \neg\textsc{Fail}\right] \\
&\quad+ \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail} \wedge \neg\textsc{TagEq}\right] \Pr\left[\neg\textsc{TagEq} \mid \neg\textsc{Fail}\right] \\
&> 1/2 + \varepsilon.
\end{aligned}$$

By averaging argument we can have at least one of the following two equations must hold:

$$\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail} \wedge \textsc{TagEq}\right] > 1/2 + \varepsilon \qquad (14)$$

or

$$\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail} \wedge \neg\textsc{TagEq}\right] > 1/2 + \varepsilon. \qquad (15)$$

Now, when $\textsc{TagEq}$ happens, then the entire codeword must be the same, otherwise the functions (f_1, f_2) can be used to forge the signature. This is formalized in Claim 6.4.

Claim 6.4. *If Eq. 14 holds, then there exists a* PPT *adversary* B *which can forge the underlying one-time signature scheme* OTSig*.*

Proof. Suppose the above equation holds then we construct the forger B as follows:

- Run the tampering adversary A to receive the function pairs (f_1, f_2).
- Generate the commitment $\mathsf{cmt} = \mathsf{Com}(m_b, r)$. Query the signing oracle to receive the signature $\sigma \leftarrow \mathsf{Sign}(pk, (m_b, r, \mathsf{cmt}))$. Set the codeword to:

$$\mathbf{c} = (c_1, c_2) := ((\mathsf{cmt}, pk,), (m_b, r, \sigma))\,.$$

- Apply the tampering functions to generate the tampered codeword $c_1' = f_1(c_1)$ and $c_2' = f_2(c_1, c_2)$.
- Parse $(\mathsf{cmt}', pk') := c_1'$ and $(m', r', \sigma') := c_2'$. Let $\mathbf{c}' = (c_1', c_2')$. Now, if $\mathbf{c} = \mathbf{c}'$, then return $\mathsf{same}^\star$ to A and abort. Otherwise output $((m', r', \mathsf{cmt}'), \sigma')$ as the forgery.

From above, it is clear that B succeeds whenever $\mathbf{c}' \neq \mathbf{c}$. However, when $\mathbf{c}' = \mathbf{c}$, then (f_1, f_2) are identity functions, and, for such functions, it is obviously impossible for A to distinguish the experiments $\mathsf{BLTamp}_{m_0}^{f_1,f_2}$ and $\mathsf{BLTamp}_{m_1}^{f_1,f_2}$. So, A must make $\mathbf{c}' \neq \mathbf{c}$ in order to win the experiment. This implies that B can forge with probability at least ε. Notice that we implicitly use the fact that FAIL does not happen; otherwise the tampering might result in an invalid signature leaving B unsuccessful.

□

Hence, we conclude that Eq. 15 holds. Finally we prove that if Eq. 15 holds then it is possible to contradict the non-malleability of the underlying commitment scheme Com. Formally we prove the following claim.

Claim 6.5. *If Eq. 15 holds, then there exists a man-in-the-middle adversary* M*, a relation* $\mathfrak{R} \subseteq \{0,1\}^k \times \{0,1\}^k$*, an auxiliary input* z*, and a message* m_b *for which the following holds:*

$$\Pr\left[\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m_b, z) = 1\right] - \Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m_b, z) = 1\right] \geq \varepsilon.$$

Proof. Set the auxiliary input $z = (pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$. For a message $\widetilde{m} \in \{0,1\}^k$, let us denote by

$$p_{\widetilde{m}} := \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \;\middle|\; \begin{array}{c}\neg\textsc{Fail} \wedge \neg\textsc{TagEq} \\ \wedge(\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \widetilde{m})\end{array}\right].$$

Now from Eq. 15 we get:

$$\begin{aligned}
&\Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail} \wedge \neg\textsc{TagEq}\right] \\
&= \sum_{\widetilde{m}\in\{0,1\}^k} p_{\widetilde{m}} \cdot \Pr\left[\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \widetilde{m} \mid \neg\textsc{Fail} \wedge \neg\textsc{TagEq}\right] \\
&> 1/2 + \varepsilon.
\end{aligned} \tag{16}$$

Hence by the averaging argument, there must exists a $\widetilde{m} \in \{0,1\}^k$ such that[16],

$$\sum_{\widetilde{m} \in \{0,1\}^k} \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \;\middle|\; \begin{array}{c} \neg\textsc{Fail} \wedge \neg\textsc{TagEq} \\ \wedge(\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \widetilde{m}) \end{array} \right] > 1/2 + \varepsilon. \tag{17}$$

The relation $\mathfrak{R}$ is defined as follows: $\mathfrak{R}(x,y) = 1$ if and only if $x = m_b$ and $y = \widetilde{m}$.
The construction of M is straightforward and given below:

- On receiving the commitment cmt from the left, it first applies f_1 on (cmt, pk) to generate (cmt', pk'). It sends cmt' as the right commitment.
- On receiving the opening (m_b, r) from the left, it applies f_2 on $\big((\mathsf{cmt}, pk), (m_b, r, \sigma)\big)$ where $\sigma \leftarrow \mathsf{Sign}(sk, (m_b, r, \mathsf{cmt}))$ and then gets (m', r', σ') as a output. It sends the values (m', r') in the right as opening.

It is clear from the description that,

$$\Pr\left[\mathsf{Mim}^{\mathsf{M}}(\mathfrak{R}, m_b, z) = 1\right] = \Pr\left[(\mathsf{BLTamp}_{m_b}^{f_1,f_2} \leftrightarrows \mathsf{A}) = b \mid \neg\textsc{Fail} \wedge \neg\textsc{TagEq}\right]. \tag{18}$$

Also note that, by perfect binding property,

$$\Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m_0, z) = 1\right] = \Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m_1, z) = 1\right].$$

This follows because the distributions of the simulator output is identical for both messages in the commitment phase, since S gets exactly the same information in both cases. However, in the decommitment phase it gets the actual message, but in that phase it can not open to another messages due to perfect binding of the commitment scheme.

So, we have for a random b,

$$\Pr\left[\mathsf{Sta}^{\mathsf{S}}(\mathfrak{R}, m_b, z) = 1\right] = 1/2. \tag{19}$$

Combining Eq. 18 and Eq. 19 we have the claim.

□

This concludes the proof of the theorem. □

7 Acknowledgment

The authors thank Yevgeniy Dodis for useful comments on the initial draft of this work.

[16]Note that even if we condition on $\neg\textsc{Fail}$ it is possible that f_1 tampers in such a way that the output of $\mathsf{BLTamp}_{m_b}^{f_1,f_2}$ becomes $\perp$, e.g. f_1 outputs a c_1' containing string of 0. However, in that case $\Pr\left[\mathsf{BLTamp}_{m_b}^{f_1,f_2} = \perp\right]$ is independent of b otherwise the computational hiding property of Com would be violated. This implies that for $\widetilde{m} = \perp$, the adversary A can not be able to distinguish $\mathsf{BLTamp}_{m_0}^{f_1,f_2}$ and $\mathsf{BLTamp}_{m_1}^{f_1,f_2}$ with probability $> 1/2 + \varepsilon$. Therefore we do not include $\perp$ in the domain of $\widetilde{m}$

References

[1] Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. Via Personal Communication in February, 2014.

[2] Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 459–468, 2015.

[3] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 774–783. ACM, 2014.

[4] Shashank Agrawal, Divya Gupta, Hemanta K Maji, Omkant Pandey, and Manoj Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations.

[5] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Explicit non-malleable codes resistant to permutations. *IACR Cryptology ePrint Archive*, 2014:316, 2014.

[6] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 345–355. IEEE, 2002.

[7] Nishanth Chandran, Vipul Goyal, Pratyay Mukherjee, Omkant Pandey, and Jalaj Upadhyay. Block-wise non-malleable codes. *IACR Cryptology ePrint Archive*, 2015:129, 2015.

[8] Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. *arXiv preprint arXiv:1505.00107*, 2015.

[9] Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 306–315. IEEE, 2014.

[10] Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 155–168. ACM, 2014.

[11] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In *Theory of Cryptography*, pages 440–464. Springer, 2014.

[12] Sandro Coretti, Ueli Maurer, Björn Tackmann, and Daniele Venturi. Domain-extension for public-key encryption via non-malleable codes. 2014.

[13] Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications, 2014. Manuscript.

[14] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. In *ASIACRYPT (2)*, pages 140–160, 2013.

[15] Ivan Damgard and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 426–437. ACM, 2003.

[16] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM review*, 45(4):727–784, 2003.

[17] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *CRYPTO (2)*, pages 239–257, 2013.

[18] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

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

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

[21] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In *Advances in Cryptology–EUROCRYPT 2014*, pages 111–128. Springer, 2014.

[22] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *Theory of Cryptography*, pages 258–277. Springer, 2004.

[23] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 695–704. ACM, 2011.

[24] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. *IACR Cryptology ePrint Archive*, 2015:1178, To appear at STOC 2016.

[25] Zahra Jafargholi and Daniel Wichs. Tamper detection and continuous non-malleable codes. Technical report, Cryptology ePrint Archive, Report 2014/956, 2014. http://eprint. iacr. org, 2014.

[26] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.

[27] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[28] Huijia Lin and Rafael Pass. Non-malleability amplification. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 189–198. ACM, 2009.

[29] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 705–714. ACM, 2011.

[30] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In *CRYPTO*, pages 517–532, 2012.

[31] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43. ACM, 1989.

[32] Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. Adaptive one-way functions and applications. In *Advances in Cryptology–CRYPTO 2008*, pages 57–74. Springer, 2008.

[33] Rafael Pass and Alon Rosen. New and improved constructions of nonmalleable cryptographic protocols. *SIAM Journal on Computing*, 38(2):702–752, 2008.

[34] Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from subexponential one-way functions. In *Advances in Cryptology–EUROCRYPT 2010*, pages 638–655. Springer, 2010.

[35] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[36] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 531–540. IEEE, 2010.

A Definitions of Non-malleable Codes

In this section we provide some supplementary definitions for completeness.

A.1 Non-malleable Codes

Definition A.1 (Encoding Scheme)**.** *An (k,n)-encoding scheme* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *consists of two functions: a randomized* encoding *function* $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ *and a deterministic decoding function* $\mathsf{Dec} : \{0,1\}^n \to \{0,1\}^\ell \cup \{\perp\}$*, such that, for every* $m \in \{0,1\}^k$, $\Pr[\mathsf{Dec}(\mathsf{Enc}(m)) = m] = 1$.

We present the indistinguishability-based definition of NMC introduced in [18], where the authors originally called this notion *strong non-malleable code* (see Def. 3.3 in [18]). For simplicity we just call it non-malleable codes.

Definition A.2 (Non-malleable Codes)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be a (k,n)-encoding scheme. Let $\mathcal{F}$ be some family of tampering functions. The* Code *is called (k,n)-non-malleable code if for every $f \in \mathcal{F}$ and any pair of messages $m_0, m_1 \in \{0,1\}^k$, the following holds:*

$$\mathsf{Tamper}^f_{m_0} \approx \mathsf{Tamper}^f_{m_1}$$

where for any $m \in \{0,1\}^k$, Tamper_m^f *is defined as*

$$\mathsf{Tamper}_m^f \equiv \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(m); c' \leftarrow f(c); \\ \textit{If } c' = c \textit{ set } m' := \mathsf{same}^\star \textit{ else } m' \leftarrow \mathsf{Dec}(c') \\ \textit{Output: } m' \end{array} \right\}$$

where the randomness is over the encoding function Enc.

Throughout, by NMC we refer to the above definition unless otherwise stated explicitly.

B Strong BNMCs

In this section, we introduce a stronger definition of block-wise non-malleable code, in which the adversary can tamper the blocks in any order of its choice. We call this notion *strong block-wise non-malleable code* (SBNMC in short) and show how to build such codes *generically* based on a weaker BNMC (here weaker refers to a code satisfying Def. 4.9) and a secret-sharing scheme in a black-box manner without any additional assumptions. Note that, since the transformation is generic, any result which we obtain for BNMC can be extended to SBNMC with a (quadratic) blow up in the size of the codeword. In particular, our construction presented in Section 5 can be extended to a SBNMC using the generic transformation provided in this section.

We formalize this notion by a permutation (mapping within the set of block indexes) controlled by the adversary along with the tampering functions.

Definition B.1 (Strong block-wise non-malleable codes)**.** *Let* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *be an* (ℓ, k, n)*-block-wise encoding scheme. Let* $\overline{\mathbf{f}} = (\overline{f}_1, \ldots, \overline{f}_\ell)$ *be any tuple of functions and* $\pi : [\ell] \to [\ell]$ *be any permutation such that* $\forall i \in [\ell]$, $\overline{f}_{\pi(i)} : \{0,1\}^{\nu_i} \to \{0,1\}^{n_{\pi(i)}}$ *where* $\nu_i = \sum_{j=1}^{i} n_{\pi(j)}$*. Then* Code *is called an* (ℓ, k, n)-strong-block-wise non-malleable code *if, for any such tuple* $\overline{\mathbf{f}}$ *and any permutation* π*, there exists an algorithm* $\overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}$ *with output domain* $\{\perp, \mathsf{same}^\star\} \cup \{0,1\}^k$ *such that, for any pair of messages* $m_0, m_1 \in \{0,1\}^k$*, the following holds:*

$$\mathsf{STamper}_{m_0}^{\overline{\mathbf{f}},\pi} \approx \mathsf{STamper}_{m_1}^{\overline{\mathbf{f}},\pi}.$$

where $\mathsf{STamper}_m^{\overline{\mathbf{f}},\pi}$ *is defined as:*

$$\mathsf{STamper}_m^{\overline{\mathbf{f}},\pi} = \left\{ \begin{array}{c} \mathbf{c} = (c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m); \\ \forall i \in [\ell] : c'_{\pi(i)} = \overline{f}_{\pi(i)}(c_{\pi(1)}, \cdots, c_{\pi(i)}); \\ \textit{Let } \mathbf{c}' = (c'_1, \ldots, c'_\ell);\ \textit{If } \mathbf{c}' = \mathbf{c} \textit{ then set } m' := \mathsf{same}^\star; \\ \textit{Else } m' \leftarrow \mathsf{Dec}(c'_1, \ldots, c'_\ell); \\ \textit{If } m' = \perp \textit{ then } m' \leftarrow \overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}(m, c_1, \ldots, c_\ell); \\ \textit{Output } m' \end{array} \right\}.$$

It is not hard to see that, in order to achieve such strong non-malleability, a block-wise code must satisfy a stronger version of uniqueness which we call *strong uniqueness*.

Definition B.2. *(Strong uniqueness) Let* $\mathsf{SCode} = (\mathsf{SEnc}, \mathsf{SDec})$ *be an* (ℓ, k, n)-SBNMC*. Let* $\zeta \in [\ell]$ *be the minimum index such that there does not exist a pair of codewords* $\mathbf{c} = (c_1, \ldots, c_\ell)$ *and* $\mathbf{c}' = (c'_1, \ldots, c'_\ell)$ *and a permutation* $\pi : [\ell] \to [\ell]$ *for which the following holds:*

- $c_{\pi(i)} = c'_{\pi(i)}$, $\forall\ i \in \{1, \ldots, \zeta - 1\}$;
- $\perp \neq \mathsf{Dec}(\mathbf{c}) \neq \mathsf{Dec}(\mathbf{c}') \neq \perp$.

Then, we call ζ *the* strong uniqueness index *of* SCode*. Alternatively we call that* SCode *has* ζ*-strong-uniqueness and also call such an encoding scheme a* ζ-strong-unique code

Remark B.3. *Similar to BNMC from the property of correctness of the code, it follows that* $\zeta \leq \ell$*. Also, note that, if a* SBNMC *has* ζ*-strong-uniqueness, then for any valid codeword,* any $j \geq \zeta$ *blocks* uniquely *determine the encoded message.*

The following corollary is a straightforward extension of Corollary 4.14.

Lemma B.4. *Let* $\mathsf{SCode} = (\mathsf{SEnc}, \mathsf{SDec})$ *be an* (ℓ, k, n)-SBNMC *which is* ζ*-strong-unique. Then* $\zeta \leq \ell - 1$*.*

Proof. Assume for the sake of contradiction that $\zeta = \ell$. This implies that there is an adversary which outputs two valid codewords $\mathbf{c} = (c_1, \ldots, c_{\ell-1}, c_\ell)$, $\mathbf{c}' = (c_1, \ldots, c_{\ell-1}, c'_\ell)$ and a permutation $\pi : [\ell] \to [\ell]$ such that

- $c_{\pi(i)} = c'_{\pi(i)}$, $\forall\ i \in \{1, \ldots, \ell - 1\}$;
- $\mathsf{Dec}(\mathbf{c}) \neq \mathsf{Dec}(\mathbf{c}')$.

so that $\mathsf{Dec}(\mathbf{c}) \neq \mathsf{Dec}(\mathbf{c})$. Let $\mathsf{Dec}(\mathbf{c}) = m$ and $\mathsf{Dec}(\hat{\mathbf{c}}) = \hat{m}$. Then the adversary can execute the following attack for any pair of messages (m_0, m_1) on the target codeword $\mathbf{t} = (\tau_1, \ldots, \tau_\ell)$ (which is encoding of either m_0 or m_1):

1. For all $i \in [\ell - 1]$, $f_{\pi(i)}$ are constant functions, each of which overwrites τ_i to c_i disregarding the input.
2. Note that $f_{\pi(\ell)}$ gets the entire codeword $\mathbf{t}$ as input. It first decodes the codeword $\widetilde{m} \leftarrow \mathsf{Dec}(\tau_1, \ldots, \tau_\ell)$. If $\widetilde{m} = m_0$, then it overwrites to c_ℓ; else, if $\widetilde{m} = m_1$, it overwrites to $\hat{c}_\ell$.

Clearly, in the above case, $\mathsf{STamper}^{\bar{\mathbf{f}},\pi}_{m_0}$ will always output m whereas $\mathsf{STamper}^{\bar{\mathbf{f}},\pi}_{m_1}$ will output m_1 which makes the experiments $\mathsf{STamper}^{\bar{\mathbf{f}},\pi}_{m_0}$ and $\mathsf{STamper}^{\bar{\mathbf{f}},\pi}_{m_1}$ easily distinguishable which is a contradiction. $\square$

Now we present a general transformation from any block-wise non-malleable code to a strong block-wise non-malleable code.

The transformation: Let $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ be a block-wise encoding scheme. Let $\mathsf{SSH}_{i,\ell}$ be an p-out-of-ℓ secret-sharing scheme which takes any λ-bit secret as input to produce shares each of size $O(\lambda)$-bit[17]. It consists of three efficient algorithms: (i) a randomized algorithm $\mathsf{Share}_{p,\ell}$ which takes any secret s as input and outputs ℓ shares $\mathbf{sh} = (sh_1, \ldots, sh_\ell)$; (ii) a deterministic algorithm $\mathsf{Recon}_{i,\ell}$ which takes any p shares from the set of all shares $\mathbf{sh}$ as input and outputs the secret s and (iii) a deterministic algorithm $\mathsf{Verify}_{p,\ell}$ which takes at least p shares (it can take more shares, basically any number between p and ℓ) from $\mathbf{sh}$ as input, checks if they form a "valid" secret-sharing and outputs 1 if and only if the check succeeds and 0 otherwise. Let $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ be an (ℓ, k, n)-BNMC. We build an (ℓ, k, n)-SBNMC SCode using Code and $\mathsf{SSH}_{i,\ell}$ for all $i \in [\ell]$ (ℓ instances of the secret-sharing scheme) as follows:

[17] Concretely using Shamir's secret sharing [35] would give a 2λ-bit share.

1. SEnc. Start with encoding the message $m \in \{0,1\}^k$ with the underlying code Code. Let $(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m)$. For each $i \in [\ell]$, secret-share the i-th block using $\mathsf{SSH}_{i,\ell}$ as follows: $(sh_1^i, \ldots, sh_\ell^i) \leftarrow \mathsf{Share}_{i,\ell}(c_i)$. Then construct the i-th block of SCode as follows: $sc_i = (sh_i^1, \ldots, sh_i^\ell)$.
2. SDec. On input a codeword $(sc_1, \ldots, sc_\ell)$, parse each sc_i as $(sh_i^1, \ldots, sh_i^\ell)$. Check if the secret shares form a valid secret-sharing by running $\mathsf{Verify}_{i,\ell}(sh_1^i, \ldots, sh_\ell^i)$ for each $i \in [\ell]$. If any of them outputs 0, then output $\perp$. Otherwise, reconstruct the shares as follows: recover c_i by running $\mathsf{Recon}_{i,\ell}$ for each $i \in [\ell]$ on any i shares among $(sc_1^i, \ldots, sc_\ell^1)$. Then decode with the decoding process of the underlying code: $m \leftarrow \mathsf{Dec}(c_1, \ldots, c_\ell)$ and output m.

Theorem B.5. *If the underlying block-wise encoding scheme* Code *is an* (ℓ, k, n)-BNMC*, then* $\mathsf{SCode} = (\mathsf{SEnc}, \mathsf{SDec})$ *is an* (ℓ, k, n')-SBNMC *where* $n' = \Theta(\ell n)$*.*

Proof. Intuitively there are two key reasons why the above transformation work: (i) the tampering function $\overline{\mathbf{f}}_{\pi(i)}$ can only re-construct just i-blocks of the underlying weaker code $(c_1, \ldots, c_i)$ and "does not know anything" about the remaining blocks; thus tampering with them would result in values independent of the original values; (ii) moreover, at this point, it has already "committed" to tampering with the first $i-1$ blocks $(c_1, \ldots, c_{i-1})$ [18] and trying to change any of them would result in an invalid secret-sharing and outputting $\perp$. So, the only thing it can do is to tamper with c_i, i.e. the i-th block of the original codeword (of the underlying weaker code) with the knowledge of the first i blocks which eventually reduces the tampering in this model to the tampering in the weaker model. The detailed proof is provided in below.

Without loss of generality, we assume that the underlying code $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ has the following property:

1. For all valid codewords $(c_1, \ldots, c_\ell)$, $c_i \neq 1^{n_i}, \forall i \in [\ell]$.
2. Let the weaker code Code have reveal index ζ. For any function $\mathbf{f}$, the replacer $\mathbf{R_f}$ for such code (by definition, there must exist a replacer $\mathbf{R_f}$) has the following property: if $c_i = 1^{n_i}$ for any $1 \leq i \leq [\zeta - 1]$ then it outputs $\perp$. Intuitively, this means that whenever the tampered codeword is invalid due to the blocks which do not reveal any information about the encoded message, i.e., the first $\zeta - 1$ blocks, then we can assume that there is no necessity of a replacer (as any such invalidity can not depend on the message). The replacer's job is to ensure that the adversary can not make the output of the experiment $\perp$ to *trivially depend* on the input i.e. by *only* tampering the last $\ell - \zeta + 1$ blocks (this can, for example, overwriting to 1^{n_ℓ} – see the discussion on the replacer after Def 4.9). So any such replacer with this additional property should work for the underlying BNMC.

Formally we make a reduction to the non-malleability of the weaker code. For any set of functions $\overline{\mathbf{f}} = (\overline{f}_1, \ldots, \overline{f}_\ell)$, any permutation $\pi : [\ell] \to [\ell]$ and any message $m \in \{0,1\}^k$ which breaks the stronger non-malleability (Def. B.1), we can construct a tuple of functions $\mathbf{f} = (f_1, \ldots, f_\ell)$ which can break Def. 4.9 as follows:

1. We start with sampling ℓ uniform random values $r_1, r_2, \ldots, r_\ell$ such that $r_i \in \{0,1\}^{n_i}$ for $i \in [\ell]$ and we hardwire these values into each f_i (for all $i \in [\ell]$). Assume that each f_i consists of two sub-functions g_i and h_i that basically transform the input/output between f_i and $\overline{f}_i$.

[18] Since for j-th block, any j shares determine the block, when $j \leq i-1$ the first $i-1$ blocks are already determined at this stage.

2. Each f_i works as follows:

   (a) It starts with executing the input transformation g_i on its own input $(c_1, \ldots, c_i)$, and produces the secret shares as follows. For the "past shares," it computes the correct shares, i.e., for $1 \leq j \leq i, (sh^j_1, \ldots, sh^j_\ell) \leftarrow \mathsf{Share}_{j,\ell}(c_j)$, and, for "future shares," it computes the shares using the random values i.e. for $i+1 \leq j \leq \ell, (sh^j_1, \ldots, sh^j_\ell) \leftarrow \mathsf{Share}_{j,\ell}(r_j)$. At the end, it outputs $(sc_{\pi(1)}, \ldots, sc_{\pi(i)})$, where for each $j \in [i]$, we have $sc_{\pi(j)} = (sh^1_{\pi(j)}, sh^2_{\pi(j)}, \ldots, sh^\ell_{\pi(j)})$. We remark that although $\mathsf{Share}_{j,\ell}()$ is a randomized algorithm, every f_i uses the same randomness (hardwired into the functions) to compute the shares. This is done so that the shares are consistent with each other across the various blocks.

   (b) In the next step each f_i applies corresponding $\overline{f}_{\pi(i)}$ on $(sc_{\pi(1)}, \ldots, sc_{\pi(i)})$ to produce the tampered block $sc'_{\pi(i)}$.

   (c) Then it runs the output transformation function h_i, which takes the entire output of g_i but the $\pi(i)$-th block $sc_{\pi(i)}$ which is replaced by the tampered block $sc'_{\pi(i)}$. For notational convenience, let us denote the whole input of h_i as $(sc'_{\pi(1)}, \ldots, sc'_{\pi(i)})$ where $\forall j \in [i-1], sc'_{\pi(j)} = sc_{\pi(j)}$ and $sc'_{\pi(i)} = \overline{f}_{\pi(i)}(sc_{\pi(1)}, \ldots, sc_{\pi(i)})$. It parses each $sc'_{\pi(j)}$ as a tuple $(sh'^1_{\pi(j)}, \ldots, sh'^\ell_{\pi(j)})$ and first checks for all $k \in [i]$ if $\mathsf{Verify}_{k,\ell}(sh'^k_{\pi(1)}, \ldots, sh'^k_{\pi(i)}) = 1$. If there exists an index $k \in [i]$ which outputs 0, this implies that the function $\overline{\mathbf{f}}_{\pi(i)}$ tampers to some invalid share(s). In that case, the corresponding f_i also tampers to some invalid codeword. In particular, h overwrites the i-th block to 1^{n_i}. Otherwise, h re-constructs the modified i-th block by running $c'_i \leftarrow \mathsf{Recon}_{i,\ell}(sh'^i_{\pi(1)}, \ldots, sh'^i_{\pi(i)})$ and outputs c'_i.

   (d) Finally f_i outputs c'_i.

For any pair of messages $m_0, m_1 \in \{0,1\}^k$ we use the hybrid argument, starting from the experiment $\mathsf{STamper}^{\overline{\mathbf{f}},\pi}_{m_0}$ and through several hybrid experiments reaching the experiment $\mathsf{STamper}^{\overline{\mathbf{f}},\pi}_{m_1}$ using the above transformation as follows:

$$\mathsf{STamper}^{\overline{\mathbf{f}},\pi}_{m_0} = \left\{ \begin{array}{c} (sc_1,\ldots,sc_\ell) \leftarrow \mathsf{SEnc}(m_0); \\ \forall i \in [\ell] : sc'_{\pi(i)} = \overline{f}_{\pi(i)}(sc_{\pi(1)},\cdots,sc_{\pi(i)}); \\ \text{If } (sc'_1,\ldots,sc'_\ell) = (sc_1,\ldots,sc_\ell), \text{ then set } m' := \mathsf{same}^\star; \\ \text{Else } m' \leftarrow \mathsf{SDec}(sc'_1,\ldots,sc'_\ell); \\ \text{If } m' = \perp \text{ then } m' \leftarrow \overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}(sc'_1,\ldots,sc'_\ell); \\ \text{Output } m' \end{array} \right\} \tag{20}$$

$$\approx \left\{ \begin{array}{c} \text{Sample uniform values} : \forall\ i \in [\ell]\ r_i \leftarrow \{0,1\}^{n_i}; \\ (c_1,\ldots,c_\ell) \leftarrow \mathsf{Enc}(m_0); \forall i \in [\ell] : c'_i = f_i(c_1,\cdots,c_i); \\ \text{If } (c'_1,\ldots,c'_\ell) = (c_1,\ldots,c_\ell), \text{ then } m' := \mathsf{same}^\star; \\ \text{Else } m' \leftarrow \mathsf{Dec}(c'_1,\ldots,c'_\ell); \\ \text{If } m' = \perp \text{ then } m' \leftarrow \mathbf{R}_{\mathbf{f}}(c'_1,\ldots,c'_\ell); \\ \text{Output } m' \end{array} \right\} \tag{21}$$

$$\equiv \left\{ \begin{array}{c} \text{Sample uniform values} : \forall\ i \in [\ell]\ r_i \leftarrow \{0,1\}^{n_i}; \\ m' \leftarrow \mathsf{Tamper}^{\mathbf{f}}_{m_0};\ \text{Output } m' \end{array} \right\}$$

$$\approx \left\{ \begin{array}{c} \text{Sample uniform values} : \forall\ i \in [\ell]\ r_i \leftarrow \{0,1\}^{n_i}; \\ m' \leftarrow \mathsf{Tamper}^{\mathbf{f}}_{m_1};\ \text{Output } m' \end{array} \right\} \tag{22}$$

$$\equiv \left\{ \begin{array}{c} \text{Sample uniform values} : \forall\ i \in [\ell]\ r_i \leftarrow \{0,1\}^{n_i}; \\ (c_1,\ldots,c_\ell) \leftarrow \mathsf{Enc}(m_1); \forall i \in [\ell] : c'_i = f_i(c_1,\cdots,c_i); \\ \text{If } (c'_1,\ldots,c'_\ell) = (c_1,\ldots,c_\ell), \text{ then set } m' := \mathsf{same}^\star; \\ \text{Else } m' \leftarrow \mathsf{Dec}(c'_1,\ldots,c'_\ell); \\ \text{If } m' = \perp \text{ then } m' \leftarrow \mathbf{R}_{\mathbf{f}}(c'_1,\ldots,c'_\ell); \\ \text{Output } m' \end{array} \right\}$$

$$\approx \left\{ \begin{array}{c} (sc_1,\ldots,sc_\ell) \leftarrow \mathsf{SEnc}(m_1); \\ \forall i \in [\ell] : sc'_{\pi(i)} = \overline{f}_{\pi(i)}(sc_{\pi(1)},\cdots,sc_{\pi(i)}); \\ \text{If } (sc'_1,\ldots,sc'_\ell) = (sc_1,\ldots,sc_\ell), \text{ then } m' := \mathsf{same}^\star; \\ \text{Else } m' \leftarrow \mathsf{SDec}(sc'_1,\ldots,sc'_\ell); \\ \text{If } m' = \perp \text{ then } m' \leftarrow \overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}(sc'_1,\ldots,sc'_\ell); \\ \text{Output } m' \end{array} \right\} \tag{23}$$

$$\equiv \mathsf{STamper}^{\overline{\mathbf{f}},\pi}_{m_1}. \tag{24}$$

Eq. (20) and Eq. (24) follow from the definition of SBNMC (see Def. B.1) except the description of the replacer $\overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}(sc'_1,\ldots,sc'_\ell)$, which can be constructed as follows. The replacer first make the consistency check: for all $i \in [\ell]$ if $\mathsf{Verify}_{i,\ell}(sh^i_1,\ldots,sh^i_\ell) = 0$ for any $i \in [\zeta-1]$, then output $\perp$ [19]. Otherwise, reconstruct the secrets from the shares: $\forall\ i\ \in [\ell]$, $c'_i \leftarrow \mathsf{Recon}_{i,\ell}(sh^i_1,\ldots,sh^i_\ell)$ and use the replacer of the weaker code $\mathbf{R}_{\mathbf{f}}$ to get $m' \leftarrow \mathbf{R}_{\mathbf{f}}(c'_1,\ldots,c'_\ell)$. and output m' where the tuple of functions $\mathbf{f}$ are described as above.

Eq. (21) and Eq. (23) follow from the security of the underlying secret sharing scheme SSH. In Eq. 21, some shares (referred as "future shares" in the above transformation) are computed using random values instead of the actual values. Intuitively, the key-fact is that any such replacement

[19] It is worth noting that the replacer $\overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}$ does not check consistency for the last $\ell-\zeta+1$ blocks. This is justified as the ζ-th block reveals some information about the message, so those inconsistencies might have been provoked depending on the message which should be essentially replaced by a valid message.

only takes place within that particular tampering function which does not have enough shares to reconstruct the secret (see the above transformation for details). By the property of secret-sharing schemes, any adversary that gets less than the threshold number of shares, cannot distinguish between the shares of two different secrets. This informal argument is not hard to formalize. We first give a sketch and the detail proof follows later. If there is a PPT adversary which can distinguish between the two tampering experiments (applying some tampering functions $\overline{\mathbf{f}}$), we can construct another PPT adversary which uses the former to distinguish shares of the actual value and a random value even without getting sufficient shares. This leads to a contradiction to the secrecy of the secret sharing scheme. Using ℓ hybrid steps, where in each step an actual value is replaced by a random value, we can complete the reduction. Another change among these two experiments is in using different replacer. However, notice that, basically the replacer $\overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}$ uses $\mathbf{R}_{\mathbf{f}}$ only in the case when there is an inconsistent secret-sharing found among first $\zeta - 1$ blocks and in which case $\overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}$ outputs $\perp$. In that case, by the above transformation, the corresponding block c_i will be overwritten to 1^{n_i}. By our assumption regarding $\mathbf{R}_{\mathbf{f}}$, we know that such a codeword must be invalid and for such invalidity the replacer $\mathbf{R}_{\mathbf{f}}$ always outputs $\perp$. We now present the reduction more formally below.

First note that Eq, 20 can be written as below:

$$\left\{\begin{array}{l}
(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m_0); \forall i \in [\ell] : (sh_1^i, \ldots, sh_\ell^i) \leftarrow \mathsf{Share}_{i,\ell}(c_i); \\
\forall i \in [\ell] : sc_i := (sh_i^1, \cdots, sc_i^\ell); \\
\forall i \in [\ell] : sc'_{\pi(i)} = \overline{f}_{\pi(i)}(sc_{\pi(1)}, \cdots, sc_{\pi(i)}); \\
\text{If } (sc'_1, \ldots, sc'_\ell) = (sc_1, \ldots, sc_\ell), \text{ then set } m' := \mathsf{same}^\star; \\
\quad \text{Else } m' \leftarrow \mathsf{SDec}(sc'_1, \ldots, sc'_\ell); \\
\quad\quad \text{If } m' = \perp \text{ then } m' \leftarrow \overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}(sc'_1, \ldots, sc'_\ell); \\
\text{Output } m'
\end{array}\right\}.$$

Also, Eq, 21 can be written as below:

$$\left\{\begin{array}{l}
\text{Sample uniform values} : \forall\ i \in [\ell]\ r_i \leftarrow \{0,1\}^{n_i}; \\
(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m_0); \\
\text{For } i \in [\ell] : \\
\quad \forall j \le i : (sh_1^i, \ldots, sh_\ell^i) \leftarrow \mathsf{Share}_{i,\ell}(c_j) \\
\quad \forall j > i : (sh_1^i, \ldots, sh_\ell^i) \leftarrow \mathsf{Share}_{i,\ell}(r_j) \\
\forall i \in [n] : sc_i := (sh_i^1, \ldots, sh_i^\ell) c'_i = f_i(c_1, \cdots, c_i); \\
\text{If } (c'_1, \ldots, c'_\ell) = (c_1, \ldots, c_\ell), \text{ then set } m' := \mathsf{same}^\star; \\
\quad \text{Else } m' \leftarrow \mathsf{Dec}(c'_1, \ldots, c'_\ell); \\
\quad\quad \text{If } m' = \perp \text{ then } m' \leftarrow \mathbf{R}_{\mathbf{f}}(c'_1, \ldots, c'_\ell); \\
\text{Output } m'
\end{array}\right\}.$$

We now give the series of hybrids. For any $0 \le i \le \ell$, we have

Hyb_i**:** In this experiment, we first compute $(c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m_0)$. Then for $i \le \ell - i$, compute the shares of $(sh_1^i, \ldots, sh_\ell^i) \leftarrow \mathsf{Share}_{i,\ell}(c_i)$ and for $i > \ell - i$, compute the shares of random value r_i as $(sh_1^i, \ldots, sh_\ell^i) \leftarrow \mathsf{Share}_{i,\ell}(r_i)$. It is clear that Hyb_0 corresponds to the case when we are in the setting of eq. 20 and Hyb_ℓ corresponds to the case when we are in the setting of eq. 21. We now show that $\mathsf{Hyb}_i \approx \mathsf{Hyb}_{i+1}$ for $0 \le i \le \ell - 1$. Since there are total ℓ hybrids, this would conclude the proof.

We break our analysis into two cases: when $i = 0$ and when $i \geq 1$. We first consider the case when $i \geq 1$.

Case $i \geq 1$**:** Let $\mathcal{A}_i$ be a distinguisher that distinguishes Hyb_i from Hyb_{i+1}. We will build a distinguishing adversary $\mathcal{B}$ that breaks the secret sharing scheme. The adversary $\mathcal{B}$ gets $(\tilde{sc}_1, \ldots, \tilde{sc}_\ell)$ as inputs which is either created by shares of $\{c_1, \cdots, c_i, r_{i+1}, \cdots, r_\ell, \ldots,\}$ or by shares of $\{c_1, \cdots, c_{i+1}, r_{i+2}, \cdots, r_\ell, \ldots,\}$. $\mathcal{B}$ then does the following:

1. $\mathcal{B}$ calls the tampering adversary to compute the tamper codewords $\tilde{c}'_1, \cdots, \tilde{c}'_\ell$.
2. $\mathcal{B}$ checks whether $(\tilde{c}'_1, \ldots, \tilde{c}'_\ell) = (\tilde{c}_1, \ldots, \tilde{c}_\ell)$. If it does, it outputs $\perp$; else it decodes using Dec. It sets this decoded value to be $\tilde{m}$.
3. If Dec in the above step outputs $\perp$, it calls $\mathbf{R_f}$ and set $\tilde{m}$ to be the output of $\mathbf{R_f}$.
4. Call $\mathcal{A}_i$ with this value of $\tilde{m}$ and outputs whatever $\mathcal{A}_i$ outputs.

It is easy to see that if $(\tilde{sc}_1, \ldots, \tilde{sc}_\ell)$ is created as the share of $\{c_1, \cdots, c_i, r_{i+1}, \cdots, r_\ell\}$, then $\mathcal{B}$ emulates the distribution of Hyb_i else it emulates the distribution of Hyb_{i+1}. Therefore, if $\mathcal{A}_i$ distinguishes between Hyb_i and Hyb_{i+1} with some non-negligible probability, then we can distinguish the random shares of r_i with the random shares of c_i with the same probabilty, ariving at a contradiction. Since this hold true for all $i \geq 1$, $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_\ell$.

Case $i = 0$**:** In order to complete the proof, we have to show $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$. We have

$$\mathsf{Hyb}_0 = \left\{ \begin{array}{l} (c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m_0); \forall i \in [\ell] : (sh^i_1, \ldots, sh^i_\ell) \leftarrow \mathsf{Share}_{i,\ell}(c_i); \\ \forall i \in [\ell] : sc_i := (sh^1_i, \cdots, sc^\ell_i); \\ \forall i \in [\ell] : sc'_{\pi(i)} = \overline{f}_{\pi(i)}(sc_{\pi(1)}, \cdots, sc_{\pi(i)}); \\ \text{If } (sc'_1, \ldots, sc'_\ell) = (sc_1, \ldots, sc_\ell), \text{ then set } m' := \mathsf{same}^\star; \\ \quad \text{Else parse } sc'_i \text{ to get } (sh^i_1, \ldots, sh^i_\ell) \text{ and } m' \leftarrow \mathsf{Dec}(sc'_1, \ldots, sc'_\ell); \\ \quad\quad \text{If } m' = \perp \text{ then } m' \leftarrow \overline{\mathbf{R}}_{\overline{\mathbf{f}},\pi}(sc'_1, \ldots, sc'_\ell); \\ \text{Output } m' \end{array} \right\}$$

while we can write

$$\mathsf{Hyb}_1 = \left\{ \begin{array}{l} \text{Sample a random } r_\ell \leftarrow \{0,1\}^{n_\ell}; \\ (c_1, \ldots, c_\ell) \leftarrow \mathsf{Enc}(m_0); \\ \text{For } i \in [\ell - 1] \\ \quad (sh^i_1, \ldots, sh^i_\ell) \leftarrow \mathsf{Share}_{i,\ell}(c_j); (sh^\ell_1, \ldots, sh^\ell_\ell) \leftarrow \mathsf{Share}_{\ell,\ell}(r_\ell) \\ (\tilde{sc}_1, \ldots, \tilde{sc}_\ell) = \pi(sc_1, \ldots, sc_\ell); sc'_i = f_{\pi(i)}(\tilde{sc}_i); \\ \text{Parse } sc_i \text{ as } (sh^i_1, \ldots, sh^i_\ell); \forall i \in [\ell] \\ \ \text{Run } \mathsf{Verify}_{i,\ell}(sh^i_1, \ldots, sh^i_\ell) \text{ and compute } (c'_1, \ldots, c'_\ell); \\ \text{If } (c'_1, \ldots, c'_\ell) = (c_1, \ldots, c_\ell), \text{ then set } m' := \mathsf{same}^\star; \\ \quad \text{Else } m' \leftarrow \mathsf{Dec}(c'_1, \ldots, c'_\ell); \\ \quad\quad \text{If } m' = \perp \text{ then } m' \leftarrow \mathbf{R_f}(c'_1, \ldots, c'_\ell); \\ \text{Output } m' \end{array} \right\}.$$

We have to consider two events depending on whether m equals $\perp$ or not. For the latter case, the proof is exactly as before for the case when $i > 0$. Therefore, conditional on the event that Dec does not output $\perp$, $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$.

In the event of $m' = \perp$, note that, the first if condition would fail in both the cases and we have to only consider the difference in the replacer in the two hybrids; in Hyb_0 we have $\overline{\mathbf{R}}_{\bar{\mathbf{f}},\pi}$ while in Hyb_1 we have $\mathbf{R_f}$. Recall that the replacer $\overline{\mathbf{R}}_{\bar{\mathbf{f}},\pi}$ uses $\mathbf{R_f}$ only in the case when there is an inconsistent secret-sharing found among first $\zeta - 1$ blocks. In this case, $\overline{\mathbf{R}}_{\bar{\mathbf{f}},\pi}$ outputs $\perp$, and, by the above transformation, the corresponding block c_i will be overwritten to 1^{n_i}. In the case of Hyb_1, at least one of the Verify calls will fail and the tampered codeword would not be equal to the original codeword. Therefore, the first conditional statement would not hold. By our assumption regarding $\mathbf{R_f}$, we know that such codewords must be invalid and for such invalidity the replacer $\mathbf{R_f}$ always outputs $\perp$. Therefore, in the event when SDec or Dec outputs $\perp$, both the distributions are identical. This completes the proof that $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$.

Finally Eq. (22) follows from the fact that the underlying code Code is a BNMC(according to Def. 4.9).

□

Instantiation. Combining Theorem B.5 and Theorem 5.15 we get a (ℓ''', k''', n''')-strong block-wise non-malleable encoding scheme such that $\ell''' = \ell'' = O(\kappa^{2+\varphi})$, $k''' = k'' \in \mathbb{N}$ and $n''' = O(n''\ell'') = O(k\kappa^{8+2\varphi})$. Formally we can get the following theorem

Theorem B.6. *Assume the existence of sub-exponentially hard one-way permutations. Then for any $\varphi > 0$ of our choice, and any $k \in \mathbb{N}$ there exists an explicit construction of (ℓ, k, n)-*SBNMC *such that $\ell = O(\kappa^{2+\varphi}), n = O(k\kappa^{8+2\varphi})$.*

and more generically,

Corollary B.7. *Assume the existence of sub-exponentially hard OWP. Then for any arbitrary constant $\varphi > 0$ of our choice there exists an explicit construction of NMCwR for class $\mathcal{F}^{\ell}_{\mathsf{s\text{-}block}}$ where ℓ is at least $O(\kappa^{2+\varphi})$.*

Moreover, we get the following corollary:

Corollary B.8. *One can observe that the rate of our construction is (inverse of) polynomial in security parameter, in particular the* SBNMC *construction has rate $\approx O(1/\kappa^8)$.*