Explicit Non-malleable Codes Against Bit-Wise Tampering and Permutations

Shashank Agrawal[1], Divya Gupta[2], Hemanta K. Maji[2,4], Omkant Pandey[3], and Manoj Prabhakaran[1(✉)]

[1] University of Illinois Urbana-Champaign, Champaign, USA
{sagrawl2,mmp}@illinois.edu
[2] Los Angeles and Center for Encrypted Functionalities, University of California, Los Angeles, USA
{divyag,hmaji}@cs.ucla.edu
[3] University of California, Berkeley, USA
omkant@gmail.com
[4] Purdue University, West Lafayette, USA

Abstract. A non-malleable code protects messages against various classes of tampering. Informally, a code is non-malleable if the message contained in a tampered codeword is either the original message, or a completely unrelated one. Although existence of such codes for various rich classes of tampering functions is known, *explicit* constructions exist only for "compartmentalized" tampering functions: i.e. the codeword is partitioned into *a priori fixed* blocks and each block can *only be tampered independently*. The prominent examples of this model are the family of bit-wise independent tampering functions and the split-state model.

In this paper, for the first time we construct explicit non-malleable codes against a natural class of non-compartmentalized tampering functions. We allow the tampering functions to *permute the bits* of the codeword and (optionally) perturb them by flipping or setting them to 0 or 1. We construct an explicit, efficient non-malleable code for arbitrarily long messages in this model (unconditionally).

We give an application of our construction to non-malleable commitments, as one of the first direct applications of non-malleable codes to computational cryptography. We show that non-malleable *string* commitments can be "entirely based on" non-malleable *bit* commitments.

S. Agrawal, D. Gupta, O. Pandey and M. Prabhakaran—Research supported in part by NSF grant 1228856.

D. Gupta, H.K. Maji and O. Pandey—Research supported in part from a DARPA/ ONR PROCEED award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11- 1-0389. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

R. Gennaro and M. Robshaw (Eds.): CRYPTO 2015, Part I, LNCS 9215, pp. 538–557, 2015.
DOI: 10.1007/978-3-662-47989-6_26

1 Introduction

Non-malleability is a cryptographic notion [16] which requires that an encoding (encryption, commitment etc.) of a message cannot be used to create a valid encoding of a "related" message by a (computationally) crippled adversary. Non-malleable codes [18] is a special case of this idea: here, the encoding is in the form of a single string (rather than an interactive protocol), but the attacker is heavily crippled in that the tampering function it can apply on a codeword must belong to very simple classes (e.g., bit-wise functions). Non-malleable codes are required to be secure without relying on any computational restriction on the adversary, but instead – as is the case for other information-theoretic cryptographic primitives like secret-sharing and information-theoretically secure multi-party computation – relies on limitations of the adversary's access[1] to information. The limited access is captured by limiting the class of tampering functions.

Ever since non-malleable codes were explicitly introduced, there has been a tremendous body of work on the topic [1,2,9–11,17,18]. Even when there are severe restrictions on the tampering class, it has been a challenge to obtain *explicit constructions* of non-malleable codes. All prior explicit constructions of non-malleable codes rely on the "compartmentalized" structure of the tampering function, i.e. the codeword is partitioned into *a priori fixed* blocks and *any tampering function should tamper with each block independently.* The prominent examples of this model are the family of bit-wise independent tampering functions and the split-state model.

In this work, we seek to build explicit non-malleable codes (with efficient encoding and decoding algorithms) for certain non-compartmentalized tampering functions. In particular, we consider bit-permutation attacks composed with arbitrary bit-wise functions. Our result could be seen as a first step towards one of the major problems in non-malleable codes: *to construct explicit codes that are non-malleable against low-complexity function classes like* NC^0 *and* AC^0. Indeed, the tampering functions we consider are very low-complexity circuits, with only unary gates and no fan-outs.

We point out that existential results and efficient randomized constructions are known for non-malleable codes against a very broad classes of tampering functions. However, given the theoretical importance of non-malleable codes (as evidenced by a deep and rich literature), explicit constructions are of great interest. Further, randomized constructions, even when they are efficient, are not suitable for some cryptographic applications. Indeed, in this work, we present a novel cryptographic application of non-malleable codes to non-malleable string commitments. Among other things, this is an instance of an application where neither party can be trusted to carry out the code construction honestly. We discuss this application further, later in this section.

Construction Sketch. Our non-malleable code construction consists of four steps, that are sketched below. We present a more detailed overview and further motivation behind these steps in the full version [3].

[1] Here access refers to both the ability to read and write the information in the system.

- We start with a large-alphabet randomized encoding which has a large enough distance and whose positions are t-wise independent for a large enough t (e.g., a "packed secret-sharing scheme" based on the Reed-Solomon code suffices), and make it resistant to permutations by incorporating into each character its position value; i.e., the character at the i^{th} position in a codeword x_i is re-encoded as $\langle i, x_i \rangle$, and allowed to occur at any position in the new codeword.
- The above code uses a large alphabet. It is concatenated with a binary inner code that is also resistant to permutations: each character in the outer code's alphabet is mapped to a positive integer (in a certain range) and is encoded by a block of bits whose weight (number of positions with a 1) equals this integer. Note that a permutation may move bits across the different blocks. To resist such attacks, we keep the bits within each block randomly permuted, and also, ensure that a good fraction of the weights do not correspond to a valid block (achieved, for instance, by requiring that the weight of each block is a multiple of 3[2]), so that blindly mixing together bits from different blocks has *some* probability of creating an invalid block. A careful combinatorial argument can be used to show that, despite dependencies among the blocks caused by a permutation attack, the probability of having all attacked blocks remaining valid decreases multiplicatively with the number of blocks being attacked thus. This, combined with the fact that the outer code has a large distance, ensures that the probability of creating a different valid codeword by this attack is negligible. However, we need to ensure not only that the attack has negligible chance of modifying one codeword into a different valid codeword, but also that the probability of creating an invalid codeword is (almost) independent of the actual message. Roughly, this is based on the large independence of the outer code.
- The resulting code is not necessarily resistant to attacks which can set/reset several bits. Towards achieving resistance to such attacks as well, we consider an intermediate 2-phase attack family: here the adversary can set/reset bits at *random positions*, learn which positions were subjected to this attack, and then specify a permutation attack.[3] To resist such attacks, we encode each bit in the above codeword into a bundle, using an additive secret-sharing. Then, if one or more bits in a bundle are set/reset, all the other bits in the bundle turn uniformly random. Hence, unless the adversary chooses to set/reset a very large number of positions (in which case almost every bundle is touched, and all information about the original message is lost), for every bit which has been set/reset, there will be several that are uniformly random. Now, even though the adversary can apply a permutation to rearrange these random bits (into as few bundles as possible), to ensure that there are only

[2] In our actual analysis, we also allow the attacker to flip any subset of bits. This prevents us from having valid weights to be 0 modulo 2, as flipping an even number of positions preserves this parity.

[3] In the actual analysis, we need to consider a slightly stronger 2-phase attack, in which the adversary can also learn the values of the bits in a small number of positions before specifying a permutation (and flipping a subset of bits).

a few bundles with a random bit, the adversary is forced to set/reset at most a few bundles' worth of bits. We note that our actual analysis follows a somewhat different argument, but fits the above intuition.

- Finally, the above code is modified as follows: a random permutation over the bits of the code is applied to a codeword; the permutation itself is encoded using a code of large distance, and appended to the above (permuted) codeword. Then it can be shown that a full-fledged attack (involving arbitrary set/reset and permutations) on such a codeword translates to a 2-phase attack of the above kind. Note that we do *not* rely on the permutation itself to be encoded in a non-malleable fashion. Indeed, the adversary can be allowed to learn and modify the encoded permutation *after* it has committed to the set/reset part of its attack on the rest of the codeword; in the 2-phase attack, this is modeled by the fact that the adversary can learn which positions in the codeword were set and reset, before deciding on the permutation attack.

  As sketched above, the rate of our code is zero, as the codewords are polynomially longer than the message. However, the generic compiler provided in [4] when instantiated with our code yields a rate 1 code (i.e., the codewords are only marginally longer than the messages, with the increase being sub-linear in the length of the message).

An Application. One motivation behind the class of attacks considered in this work comes from the following intriguing question:

> *Can non-malleable string-commitments be "entirely based" on non-malleable bit-commitments?*

To formalize this problem, we may consider an idealized model of bit commitments using physical tokens: to commit a bit to Bob, Alice can create a small physical token which has the bit "locked" inside (and later, she can send him a "key" to open the token). This completely hides the bit from Bob until Alice reveals it to him; on the other hand, Alice cannot change the bit inside the token once she has sent it to Bob. Further, this is a non-malleable bit commitment scheme, in that if Bob plays a man-in-the-middle adversary, and wants to send a commitment to Carol, he can only send the token from Alice as it is, or create a new token himself, independent of the bit committed to by Alice.

Now, we ask whether, in this model, one can make non-malleable string commitments (relying on no computational assumptions). *This is a question about non-malleable codes in disguise!* Indeed, if we required the commitment protocol to involve just a single round of sending a fixed number of tokens, then a commitment protocol is nothing but a non-malleable encoding of a string into bits, and the class of tampering functions we need to protect against is that of *bit-level permutations* and bit-wise set/reset.[4] Though we presented this

[4] For this application, bit-flipping need not be part of the admissible tampering functions. However, even if we restricted ourselves to this simpler class, our construction does not become significantly simpler. Indeed, handling permutations and set/reset present the biggest technical challenges in our construction. By handling bit-flipping as well, our tampering function family subsumes the bit-wise tampering function family.

string commitment scheme in an idealized setting involving tokens, it can be translated to *a reduction of non-malleable string commitment to* CCA-secure *bit commitment (as defined in* [6]*).*

As mentioned above, the non-malleable codes we build can withstand a slightly larger class of tampering attacks, which corresponds to the ability of the adversary to apply any set of functions from $\{0,1\}$ to $\{0,1\}$ to the bits stored in the tokens (i.e., set, reset, flip or keep), before applying the permutation attack. As such, in the above application, we do not actually require the bit commitment scheme to be CCA secure. We also present a variant of the above construction to illustrate this, which we base on a specific (non-standard) assumption on a PRG.

This application also illustrates why explicit constructions can be of importance to cryptographic constructions. While there indeed is an efficient randomized construction of non-malleable codes that can resist permutations [20], it will not be suitable in this case, because neither the sender nor the receiver in a commitment scheme can be trusted to pick the code honestly (Bob could play either role), and non-malleable codes are not guaranteed to stay non-malleable if the description of the code itself can be tampered with. While one may address this issue using a more complex protocol which securely samples a non-malleable code using (malleable) string commitments, this undermines the simplicity of our protocol (which involves no interaction beyond carrying out the bit commitments) and introduces more rounds of interaction.

1.1 Prior Work

Cramer et al. [14] introduced the notion of arithmetic manipulation detection codes, which is a special case of non-malleable codes; AMD codes with optimal parameters have been recently provided by [15]. Dziembowski et al. motivated and formalized the more general notion of non-malleable codes in [18]. They showed existence of a constant rate non-malleable code against the class of all bit-wise independent tampering functions. Existence of rate 1 non-malleable codes against various classes of tampering functions is known. For example, existence of such codes with rate $(1-\alpha)$ was shown against any tampering function family of size $2^{2^{\alpha n}}$; but this scheme has inefficient encoding and decoding [10]. For tampering functions of size $2^{\text{poly}(n)}$, rate 1 codes (with efficient encoding and decoding) exist with overwhelming probability [20].

On the other hand, explicit constructions of non-malleable codes have remained elusive, except for some well structured tampering function classes. Recently, an explicit rate 1 code for the class of bit-wise independent tampering function was proposed by [11]. Note that a tampering function in this class tampers each bit independently. For a more general compartmentalized model of tampering, in which the codeword is partitioned into separate blocks and each block can be tampered arbitrarily but independently, an encoding scheme was proposed in [12]. In the most general compartmentalized model of tampering, where there are only two compartments (known as the split-state model), an

explicit encoding scheme for bits was proposed by [17]. Recently, in a breakthrough result, an explicit scheme (of rate 0) was proposed for arbitrary length messages by [2]. Subsequently, a constant rate construction for 10 states was provided in [9] and, building on that, a constant rate construction for the split state was proposed in [1].

Codes Under Computational Assumptions. The idea of improving the rate of error-correcting codes by considering computationally limited channels has been explored in a large body of work [8,26,27,30,32,34]. In the setting of non-malleable codes as well, constructions based on computational assumptions have been explored, for example [19,31].

Non-malleable Commitments. There is extensive literature on non-malleable commitments starting from the work of Dolev, Dwork and Naor [16] leading to recent constant-round constructions based on one-way functions [24,25,29]. Our application of nonmalleable codes to non-malleable commitments is similar in spirit to the work of Meyers and Shelat [33] on the completeness of bit encryption. Concurrently, and independently of our work, Chandran et al. [7] relate non-malleable commitments to a new notion of non-malleable codes, called *blockwise non-malleable codes*.

Application of Non-malleable Codes to Cryptographic Constructions. AMD codes have found several applications in information-theoretic cryptography, for secret-sharing, randomness extraction and secure multi-party computation (e.g., [5,14,21,23]). However, the more general notion of non-malleable codes have had few other applications, outside of the direct application to protecting the contents of device memories against tampering attacks. Our application to non-malleable commitment is one of the few instances where non-malleable codes have found an application in a natural cryptographic problem that is not information-theoretic in nature. A similar application appears in the recent independent work of Coretti et al. [13].

1.2 Our Contribution

The class of tampering functions which permutes the bits of the codeword is represented by $\mathcal{S}_N$. The set of all tampering functions which allow the adversary to tamper a *bit* by passing it through a channel is denoted by $\mathcal{F}_{\{0,1\}}$; this includes forwarding a bit unchanged, toggling it, setting it to 1, or resetting it to 0. The class of tampering functions which allows the adversary to do apply both: i.e., tamper bits followed by permuting them is represented by: $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$. Our main result is a non-malleable code against this class of tampering functions.

Theorem 1 (Non-malleable Code). *There exists an* explicit *and* efficient *non-malleable code for* multi-bit *messages against the tampering class* $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$.

Our main non-malleable encoding which is robust to $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$ relies on a basic encoding scheme. The basic encoding scheme is robust to a weaker class of

tampering functions, but it provides slightly stronger security guarantees. More specifically, the basic scheme protects only against $\widetilde{\mathcal{F}}_{\{0,1\}} \circ \mathcal{S}_N$ class, where $\widetilde{\mathcal{F}}_{\{0,1\}}$ is the class of functions which either forward a bit unchanged or toggle it but do not set or reset it. The stronger security guarantee given by basic scheme is that it allows the adversary to *adaptively* choose the tampering function $\widetilde{\mathcal{F}}_{\{0,1\}} \circ \mathcal{S}_N$. The adversary first specifies n_0 and n_1, i.e. number of indices it wants to reset to 0 and number of indices it wants to set to 1. It is provided a random subset of indices of size n_0 which is all reset to 0; and a (disjoint) random subset of indices of size n_1 which is all set to 1. Given this information, the adversary can adaptively choose the tampering function in $\widetilde{\mathcal{F}}_{\{0,1\}} \circ \mathcal{S}_N$. Even given this additional power, the adversary cannot tamper the codeword to produce related messages (except with negligible probability).

We present the basic encoding scheme and prove its non-malleability in Sect. 5. Theorem 1 is proved via a reduction to the basic scheme. This proof is provided in the full version [3].

Non-malleable Commitments. As noted earlier, we consider the question of constructing simple string non-malleable commitments from bit non-malleable commitments. For example, if we simply encode the given string and commit to each of its bit using the given non-malleable bit-commitment, does it result in a secure non-malleable string commitment schemes? What are the conditions we need on the underlying bit commitment scheme?

For this question, we are interested in a really simple reduction, as opposed to, e.g. "merely" black-box reductions. Indeed, if we ask for a merely black-box construction we can invoke known (but complex) reductions: a bit commitment scheme (even malleable) implies a one-way function, which in turn imply string commitments in a black box way [25]. Such reductions are not, what we call *totally* black-box. For example, if we switch to a model where we are given the bit-commitment scheme as a *functionality* which can be executed only a bounded number of times, such as a one-time program [22] or a hardware token [28], then we do not necessarily have standard one-way functions. Therefore, the reduction should avoid assuming additional complexity assumptions such as OWFs or signatures. In fact, for this reason, the reduction should also not rely on using tags and "tag-based" non-malleability [35]. It should work with standard *non-tag-based* non-malleable bit-commitments.

Our reduction actually satisfies these conditions provided that we start with a (non-tag-based) CCA-secure bit-commitment scheme [6]. We show that (perhaps the simplest construction where) if we just commit to each bit of a random codeword of the given string works! This gives us the following theorem:

Theorem 2 (CCA Bit-Commitment to Non-malleable String Commitment). *There exists a simple and efficient black-box compiler which, when provided with:*

- *A non-malleable encoding robust to $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$, and*
- *A r-round (possibly* **non-tag-based***) CCA-secure bit-commitment scheme yields a r-round non-malleable string-commitment scheme.*

We note that the theorem statement is **unconditional**: it does not assume any computational assumption beyond the given non-malleable bit-commitment. In particular, the theorem holds even if the bit-commitment is implemented in a model which does not necessarily imply OWFs. Furthermore, in our full version [3], we prove that in fact, the theorem holds even if the bit-commitment is not CCA-secure but only satisfies a much weaker notion which we call *bounded-parallel* security.

Finally, we show the power of our non-malleable codes by demonstrating that even if we start with a seemingly much weaker scheme which allows partial malleability, e.g., it may allow the MIM to toggle the committed bit, our non-malleable codes can "boost" it to full-fledged malleability. See [3] for details.

2 Preliminaries

We denote the set $\{1, \ldots, n\}$ by $[n]$. If $a \in [b - \varepsilon, b + \varepsilon]$, then we represent it as: $a = b \pm \varepsilon$.

Probability distributions are represented by bold capital alphabets, for example $\mathbf{X}$. The distribution $\mathbf{U}_S$ represents a uniform distribution over the set S. Given a distribution $\mathbf{X}$, $x \sim \mathbf{X}$ represents that x is sampled according to the distribution $\mathbf{X}$. And, for a set S, $x \xleftarrow{\$} S$ is equivalent to $x \sim \mathbf{U}_S$. For a joint variable $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_n)$ and $S = \{i_1, \ldots, i_{|S|}\} \subseteq [n]$, we define the random variable $\mathbf{X}_S = (\mathbf{X}_{i_1}, \ldots, \mathbf{X}_{i_{|S|}})$. We also define hamming distance between two samples $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2 \ldots, v_n)$ drawn from the distribution $\mathbf{X}$ as the number of indices at which they differ, i.e., $\mathrm{HD}(u, v) = |\{i \in [n] \mid u_i \neq v_i\}|$. For a function $f(\cdot)$, the random variable $\mathbf{Y} = f(\mathbf{X})$ represents the following distribution: Sample $x \sim \mathbf{X}$; and output $f(x)$. Further, $f(x_{[n]})$ represents the vector $f(x_1) \ldots f(x_n)$.

The statistical distance between two distributions $\mathbf{S}$ and $\mathbf{T}$ over a finite sample space I is defined as: $\mathrm{SD}(\mathbf{S}, \mathbf{T}) := \frac{1}{2} \sum_{i \in I} |\Pr_{x \sim \mathbf{S}}[x = i] - \Pr_{x \sim \mathbf{T}}[x = i]|$.

2.1 Classes of Tampering Functions

We shall consider the following set of tampering functions.

1. Family of Permutations. Let $\mathcal{S}_N$ denote the set of all permutations $\pi : [N] \to [N]$. Given an input codeword $x_{[N]} \in \{0,1\}^N$, tampering with function $\pi \in \mathcal{S}_N$ yields the following codeword: $x_{\pi^{-1}(1)} \ldots x_{\pi^{-1}(N)} =: x_{\pi^{-1}([N])}$.
2. Family of Fundamental Channels. The set of fundamental channels over {0,1}, represented as $\mathcal{F}_{\{0,1\}}$, contains the following binary channels f: (a) $f(x) = x$, (b) $f(x) = 1 \oplus x$, (c) $f(x) = 0$, or (d) $f(x) = 1$. These channels are, respectively, called *forward*, *toggle*, *reset* and *set* functions.
3. Family of Sensitive Channels. The set of *sensitive* functions $\widetilde{\mathcal{F}}_{\{0,1\}}$ contains only forward and toggle channels. In other words, tampering involves XORing an N-bit input string with a fixed N-bit string.

We can define more complex tampering function classes by composition of these function classes. For example, composition of $\mathcal{S}_N$ with $\mathcal{F}_{\{0,1\}}$ yields the following class of tampering functions. For any $\pi \in \mathcal{S}_N$ and $f_1, \ldots, f_N \in \mathcal{F}_{\{0,1\}}$, it transforms a codeword $x_{[N]}$ into $f_1(x_{\pi^{-1}(1)}) \ldots f_N(x_{\pi^{-1}(N)}) =: f_{1,\ldots,N}(x_{\pi^{-1}([N])})$. This class is represented by: $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$.

2.2 Non-malleable Codes

We define non-malleable codes formally in Fig. 1. Our main result provides an efficient non-malleable code against the tampering class $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$.

3 Building Blocks

In this section, we define various types of secret-sharing schemes relevant to our construction. First we present the basic notion.

Let $\mathcal{F}$ be a set of functions of the form $f : \{0,1\}^N \to \{0,1\}^N$. Consider two mappings $\mathsf{Enc} : \{0,1\}^L \to \{0,1\}^N$ (possibly randomized) and $\mathsf{Dec} : \{0,1\}^N \to \{0,1\}^L \cup \{\perp\}$.

For $f \in \mathcal{F}$ and $s \in \{0,1\}^L$, define a random variable $\mathsf{Tamper}_f^{(s)}$ over $\{0,1\}^L \cup \{\perp\}$ as follows:

$$\mathsf{Tamper}_f^{(s)} = \mathsf{Dec}(f(\mathsf{Enc}(s))).$$

Let Sim be a map from $\mathcal{F}$ to distributions over the sample space $\{0,1\}^L \cup \{\mathsf{same}^*, \perp\}$. For $f \in \mathcal{F}$ and $s \in \{0,1\}^L$, define the random variable $\mathsf{Copy}_{\mathsf{Sim}(f)}^{(s)}$ as follows.

$$\mathsf{Copy}_{\mathsf{Sim}(f)}^{(s)} = \begin{cases} s & \text{if } \mathsf{Sim}(f) = \mathsf{same}^* \\ \mathsf{Sim}(f) & \text{otherwise.} \end{cases}$$

The simulation error (or, advantage) is defined to be:

$$\mathsf{adv}_{\mathsf{Enc},\mathsf{Dec},\mathcal{F}} := \inf_{\mathsf{Sim}} \max_{\substack{s \in \{0,1\}^L \\ f \in \mathcal{F}}} \mathrm{SD}\left(\mathsf{Tamper}_f^{(s)}, \mathsf{Copy}_{\mathsf{Sim}(f)}^{(s)}\right)$$

$(\mathsf{Enc}, \mathsf{Dec})$ is called an $[N, L, \nu]$-*non-malleable code against* $\mathcal{F}$ if the following conditions hold:

- Correctness: $\forall s \in \{0,1\}^L$, $\Pr[\mathsf{Dec}(\mathsf{Enc}(s)) = s] = 1$.
- Non-Malleability: $\mathsf{adv}_{\mathsf{Enc},\mathsf{Dec},\mathcal{F}} \leqslant \nu(N)$.

We refer to an $[N, L, \nu]$-non-malleable code simply as an $[N, L]$-non-malleable code if ν is negligible. We say that the coding scheme is *efficient* if Enc and Dec run in time bounded by a polynomial in N.

Fig. 1. Definition of non-malleable codes

Definition 1 (Secret-Sharing Scheme (SSS)). *Let* $\mathbf{S} = (\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_M)$ *be a joint distribution over* $\Lambda^L \times \Sigma^M$*, such that the support of* $\mathbf{X}_0$ *is all of* Λ^L*. (The random variable* $\mathbf{X}_0$ *represents the secret being shared and* $\mathbf{X}_i$ *for* $i \in [m]$ *represents the i-th share.)*

We say that $\mathbf{S}$ *is an* $[M, L, T, D]_{\Lambda,\Sigma}$ secret-sharing scheme *if the following conditions hold:*

1. *T-privacy:* $\forall\ s, s' \in \Lambda^L$, $\forall\ J \subseteq [M]$ *such that* $|J| \leqslant T$*, we have*

$$\mathrm{SD}\left((\mathbf{X}_J|\mathbf{X}_0 = s), (\mathbf{X}_J|\mathbf{X}_0 = s')\right) = 0.$$

2. *D-distance: For any two distinct* $c, c' \in \mathsf{Supp}(\mathbf{X}_{[M]})$*, the hamming distance between them,* $\mathrm{HD}(c, c')$*, is at least D, where* $\mathsf{Supp}(\mathbf{X}_{[M]})$ *denotes the support of distribution* $\mathbf{X}_{[M]}$*.*
3. *Reconstruction: For any* $s, s' \in \Lambda^L$ *such that* $s \neq s'$*, we have*

$$\mathrm{SD}\left((\mathbf{X}_{[M]}|\mathbf{X}_0 = s), (\mathbf{X}_{[M]}|\mathbf{X}_0 = s')\right) = 1.$$

In the remainder of the paper, by an SSS scheme, we shall implicitly refer to a family of SSS schemes indexed by M, i.e., $[M, L(M), T(M), D(M)]$-SSS schemes for each positive integer M. We define the *rate* of such a scheme to be $\lim_{M\to\infty} \frac{L(M)}{M}$. We will be interested in *efficient* SSS schemes. For this, we define two algorithms for encoding and decoding as follows:

- $\mathsf{Enc}_{\mathsf{SSS}}(s)$: This is a randomized algorithm that takes $s \in \Lambda^L$ as input and outputs a sample from the distribution $(\mathbf{X}_{[M]}|\mathbf{X}_0 = s)$.
- $\mathsf{Dec}_{\mathsf{SSS}}(\tilde{c})$: This algorithm takes a $\tilde{c} \in \Sigma^M$ as input, and outputs a secret $s \in \Lambda^L$ such that $\tilde{c} \in \mathsf{Supp}(\mathbf{X}_{[M]}|\mathbf{X}_0 = s)$. If such a secret does not exist, it outputs $\perp$.

Note that the uniqueness of the output of algorithm $\mathsf{Dec}_{\mathsf{SSS}}$ is guaranteed by the reconstruction property. An SSS scheme is said to be *efficient* if the two algorithms defined above run in time bounded by a polynomial in M.

We can instantiate a secret-sharing scheme with all the properties described above using Reed-Solomon codes. Let n, k and ℓ be any three positive integers such that $n \geqslant k \geqslant \ell$. Let $\mathbb{F}$ be a finite field of size at least $n + \ell$. Let $\{u_{-\ell}, \ldots, u_{-1}, u_1, \ldots, u_n\} \subseteq \mathbb{F}$. The secret-sharing of a message $(s_1, \ldots, s_\ell) \in \mathbb{F}^\ell$ is done by choosing a random polynomial $p(\cdot)$ of degree $< k$ conditioned on $(p(u_{-1}), \ldots, p(u_{-\ell})) = (s_1, \ldots, s_\ell)$. The shares $\{y_1, \ldots, y_n\}$ are evaluations of $p(\cdot)$ at $\{u_1, \ldots, u_n\}$ respectively. It is known that efficient encoding and decoding procedures exist using Lagrange interpolation. Further, this encoding has privacy $k - \ell$ and distance $n - k + 1$.

SSS with Independence. A secret-sharing scheme with independence, or i-SSS in short, is defined in the same way as an SSS, except that instead of privacy, it has a stronger independence property:

- T-independence: $\forall\ s \in \Lambda^L$, $\forall\ J \subseteq [M]$ such that $|J| \leqslant T$, we have

$$\mathrm{SD}\left((\mathbf{X}_J|\mathbf{X}_0 = s), \mathbf{U}_{\Sigma^{|J|}}\right) = 0.$$

In simple words, any subset of at most T shares are uniformly distributed over the corresponding codeword space (whereas privacy only guarantees that these shares have a distribution independent of the secret being shared). The encoding and decoding algorithms for i-SSS are denoted by $\mathsf{Enc}_{\text{i-SSS}}$ and $\mathsf{Dec}_{\text{i-SSS}}$ respectively. Reed-Solomon codes, as defined above, actually give independence and not just privacy.

Augmented SSS. A secret-sharing scheme (with or without independence) can be augmented to have each share also specify *which* share it is – the first, the second, etc. More formally, if we have an $[M, L, T, D]_{\Lambda,\Sigma}$-SSS (or i-SSS) with $(\mathsf{Enc}_{\mathsf{SSS}}, \mathsf{Dec}_{\mathsf{SSS}})$ algorithms, we define algorithms $\mathsf{Enc}_{\text{a-SSS}}$ and $\mathsf{Dec}_{\text{a-SSS}}$ for the augmented secret-sharing scheme over $\Lambda^L \times ([M] \times \Sigma)^M$ as follows:

- $\mathsf{Enc}_{\text{a-SSS}}(s)$: Run $\mathsf{Enc}_{\mathsf{SSS}}(s)$ to obtain $c_1, \ldots, c_M$. Output $(1, c_1), \ldots, (M, c_M)$.
- $\mathsf{Dec}_{\text{a-SSS}}(\tilde{c})$: Let $\tilde{c} = ((i_1, \tilde{c}_1), \ldots, (i_M, \tilde{c}_M))$. Sort the shares according to the first element in each tuple, check that each index occurs exactly once, and then output $\mathsf{Dec}_{\mathsf{SSS}}((\tilde{c}_1, \ldots, \tilde{c}_M))$.

It is easy to observe that a-SSS defined in this way has T-privacy and D-distance.

Additive Secret Sharing. An $[M, L]_{\Lambda,\Sigma}$ additive secret sharing scheme, referred to simply as add, is an $[M, L, T, D]_{\Lambda,\Sigma}$-i-SSS with $T = M-1$ and $D = 1$. One can instantiate such sharing schemes over any Abelian group $(G, +)$. The joint distribution $(\mathbf{X}_0, \ldots, \mathbf{X}_M)$ is defined via the following sampling procedure: pick $x_1, \ldots, x_M \xleftarrow{\$} G$ and set $x_0 = \sum_{i \in [M]} x_i$. It is easy to see that there exist efficient encoding and decoding algorithms, which are denoted by $\mathsf{Enc}_{\mathsf{add}}$ and $\mathsf{Dec}_{\mathsf{add}}$ respectively.

Balanced Unary Encoding. This scheme is parameterized by a message space F and a positive integer p. Let $\pi \colon F \to \mathbb{Z}_{|F|}$ be a bijection and $m = 3p|F| + 1$. Then, given a message $s \in F$, the encoding $\mathsf{Enc}_{\mathsf{unary}}(s)$ is performed as follows: Sample a random set S of $[m]$ of weight $\lceil m/3 \rceil + p\pi(s)$. The codeword is defined to be the characteristic vector of set S. Note that this scheme has efficient encoding and decoding algorithms, $\mathsf{Enc}_{\mathsf{unary}}$ and $\mathsf{Dec}_{\mathsf{unary}}$, respectively. For any $s \in F$ and any set S used for encoding s, the total weight of the final shares lie in $[m/3, 2m/3]$. Hence, the name *balanced unary* secret sharing scheme.

We now define how to combine two or more schemes.

Definition 2 (Concatenating Sharing Schemes). *Consider two secret sharing schemes, an outer scheme* $\mathbf{S}^{(\mathsf{out})} = (\mathbf{X}_0^{(\mathsf{out})}, \mathbf{X}_1^{(\mathsf{out})}, \ldots, \mathbf{X}_n^{(\mathsf{out})})$ *over* $\Lambda^L \times \Sigma^N$ *and an inner scheme* $\mathbf{S}^{(\mathsf{in})} = (\mathbf{X}_0^{(\mathsf{in})}, \mathbf{X}_1^{(\mathsf{in})}, \ldots, \mathbf{X}_m^{(\mathsf{in})})$ *over* $\Sigma \times \Gamma^M$*. The concatenation of the outer scheme with the inner scheme is defined as the joint distribution* $\mathbf{S}^{(\mathsf{concat})} = (\mathbf{X}_0^{(\mathsf{concat})}, \mathbf{X}_1^{(\mathsf{concat})}, \ldots, \mathbf{X}_{NM}^{(\mathsf{concat})})$ *over* $\Lambda^L \times \Gamma^{MN}$*. Given a secret* $s \in \Lambda^L$*, sample* $\mathbf{x}_{[NM]}^{(\mathsf{concat})} \sim (\mathbf{X}_{[NM]}^{(\mathsf{concat})} | \mathbf{X}_0^{(\mathsf{concat})} = s)$ *as follows: first sample* $\mathbf{x}_{[N]}^{(\mathsf{out})} \sim \left(\mathbf{X}_{[N]}^{(\mathsf{out})} \;\middle|\; \mathbf{X}_0^{(\mathsf{out})} = s\right)$*, and then for each* $i \in [N]$*, sample* $\mathbf{x}_{(i-1)m+[M]}^{(\mathsf{concat})} \sim \left(\mathbf{X}_{[M]}^{(\mathsf{in})} \;\middle|\; \mathbf{X}_0^{(\mathsf{in})} = \mathbf{x}_i^{(\mathsf{out})}\right)$*. We use* $\mathbf{S}^{(\mathsf{concat})} = \mathbf{S}^{(\mathsf{out})} \circ \mathbf{S}^{(\mathsf{in})}$ *to represent the concatenation of* $\mathbf{S}^{(\mathsf{out})}$ *with* $\mathbf{S}^{(\mathsf{in})}$*.*

If the encoding and decoding procedures for outer and inner schemes are $(\mathsf{Enc}^{(\mathsf{out})}, \mathsf{Dec}^{(\mathsf{out})})$ and $(\mathsf{Enc}^{(\mathsf{in})}, \mathsf{Dec}^{(\mathsf{in})})$ respectively, then the corresponding procedures for the concatenated scheme are denoted by $(\mathsf{Enc}^{(\mathsf{out})} \circ \mathsf{Enc}^{(\mathsf{in})}, \mathsf{Dec}^{(\mathsf{out})} \circ \mathsf{Dec}^{(\mathsf{in})})$. Note that the final encoding and decoding procedures are efficient if the corresponding procedures are efficient for inner and outer schemes.

Moreover, we emphasize that we do not focus on error correcting codes. In particular, if any of inner or outer decoding procedures fails, we output $\perp$ as the decoding of the overall code.

4 Our Non-malleable Encoding Scheme

In this section, we describe our non-malleable encoding scheme against the class of tampering functions $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$. It proceeds in following two steps.

1. **Basic Encoding Scheme.** Though this scheme will offer non-malleability against a weaker class of tampering functions, it will offer stronger guarantees beyond standard non-malleability. We refer to this as "2-Phase Non-malleability" property. The security proof of our main construction described below reduces to the 2-phase non-malleability of our basic scheme.

   The basic encoding scheme is described formally in Fig. 2. As a high level, our encoding scheme is a concatenation code (see Definition 2) which does the following: Given a message s, it samples an outer code according to augmented Reed-Solomon code based secret sharing. Then for each outer code element, it samples an inner codeword which itself is a concatenation code using balanced unary secret sharing scheme and additive sharing scheme.
2. **Main Construction.** Our main non-malleable coding scheme resistant against the class of attacks $\mathcal{F}_{\{0,1\}} \circ \mathcal{S}_N$ is built on top of the basic coding scheme. In order to encode a message s, we choose a random permutation σ. The codeword consists of two parts: the first part is the basic encoding of s with σ applied on it, and the second part is a secret sharing of σ with high distance and independence encoding. Intuitively, applying a random permutation ensures that setting/resetting bits in the main codeword results in random positions being modified in the basic codeword, exactly the kind of attack basic code can handle. The scheme is described formally in Fig. 3.

In the following section, we first describe the 2-phase security of basic encoding scheme and then our main construction.

5 Basic Encoding Scheme and 2-Phase Non-malleability

2-Phase Non-Malleability is a two-phase attack experiment where the adversary gets additional information about the codeword in the first phase before it gets to choose the tampering function in second phase.

Let κ be the statistical security parameter. Let $\{0,1\}^L$ be the message space, such that $L = \kappa^{\Theta(1)}$.
Let $\ell = \frac{L}{2 \lg L}$. Let $\mathbb{F}$ be a characteristic 2 field. Let $(\mathsf{Enc}_{\mathsf{a\text{-}SSS}}, \mathsf{Dec}_{\mathsf{a\text{-}SSS}})$ be a $[n, \ell, k, d]_{\mathbb{F},\mathbb{F}}$ augmented secret sharing scheme such that $k = 2n/3$ and $d = \Theta(n)$. Define $(\mathsf{Enc}^{(\mathsf{out})}, \mathsf{Dec}^{(\mathsf{out})}) = (\mathsf{Enc}_{\mathsf{a\text{-}SSS}}, \mathsf{Dec}_{\mathsf{a\text{-}SSS}})$.

Let $F = [n] \times \mathbb{F}$ and $u = \omega(\log \kappa)$. Let $(\mathsf{Enc}_{\mathsf{unary}}, \mathsf{Dec}_{\mathsf{unary}})$ be the unary encoding scheme mapping F to $\{0,1\}^m$ and $(\mathsf{Enc}_{\mathsf{add}}, \mathsf{Dec}_{\mathsf{add}})$ be the additive encoding scheme mapping $\{0,1\}$ to $\{0,1\}^u$ (see Section 3). Define $\mathsf{Enc}^{(\mathsf{in})} = \mathsf{Enc}_{\mathsf{unary}} \circ \mathsf{Enc}_{\mathsf{add}}$ as the concatenation encoding scheme (see Definition 2). Let $\mathsf{Dec}^{(\mathsf{in})}$ be the corresponding decoding algorithm.

Define $\mathsf{Enc}^{(\mathsf{basic})} = \mathsf{Enc}^{(\mathsf{out})} \circ \mathsf{Enc}^{(\mathsf{in})}$ and $\mathsf{Dec}^{(\mathsf{basic})} = \mathsf{Dec}^{(\mathsf{out})} \circ \mathsf{Dec}^{(\mathsf{in})}$.

Encoding: Given $s \in \mathbb{F}^\ell$, output $c_{[umn]} \sim \mathsf{Enc}^{(\mathsf{basic})}(s)$.

Decoding: Given $c_{[umn]}$, output $\mathsf{Dec}^{(\mathsf{basic})}(c)$.

Fig. 2. Basic non-malleable Code achieving 2-phase non-malleability.

1. In the first phase the adversary sends message s and $n_0, n_1, n_p \in [N]$ such that $n_0 + n_1 + n_p \leqslant N$ and $n_p \leqslant \log^2 \kappa$. Here n_0 and n_1 refer to the number of bits in the tampered codeword that will be set to 0 and 1, respectively. n_p refers to the number of bits of the original codeword which will be revealed to the adversary before he chooses the final tampering function.
2. The challenger picks an index set $I \xleftarrow{\$} \binom{[N]}{n_0 + n_1 + n_p}$ and randomly partitions I into I_0, I_1 and I_p of size n_0, n_1 and n_p, respectively. It picks $c = \mathsf{Enc}(s)$. Then it sends (I_0, I_1, I_p, c_{I_p}) to the adversary. Here I_0 and I_1 refer to the indices which will be set to 0 and 1, respectively. I_p refers to the indices of c which are revealed in c_{I_p}.
3. In the second phase, the adversary sends a tampering function $f \in \widetilde{\mathcal{F}}_{\{0,1\}} \circ \mathcal{S}_N$, using the information obtained from first phase.
4. The tampered codeword is created by setting the bits at positions I_0 and I_1 to 0 and 1, respectively, to obtain c'. Then, f is applied to c'.

Observe that in the above experiment the adversary can specify n_0, n_1, n_p and a function map in advance, and then the challenger can carry out the entire experiment on its own. The function map takes three disjoint subsets of indices $I_0, I_1, I_p \subseteq [N]$ of size n_0, n_1, and n_p respectively, and a bit string of length $|I_p| \leqslant \log^2 \kappa$, and outputs a function $f \in \widetilde{\mathcal{F}}_{\{0,1\}} \circ \mathcal{S}_N$. Let $\mathcal{F}^*$ be a tampering function family where an $f^* \in \mathcal{F}^*$ is specified by n_0, n_1, n_p and map, and tampers the codeword as the challenger does. Our security requirement can now be simply stated as non-malleability against the tampering class $\mathcal{F}^*$. The only issue is that the functions in this family are randomized and we have defined non-malleability w.r.t. deterministic functions. However, we could just define the randomness of

Let $\{0,1\}^L$ be the message space and $\{0,1\}^N$ be the codeword space. Let $N^{(1)}$ be the size of codeword output by $\mathsf{Enc}_{\mathsf{basic}}$. Let $\mathbb{F}$ be a finite field of characteristic 2 and $\ell = \left\lceil \frac{N^{(1)} \log N^{(1)}}{\log_2 |\mathbb{F}|} \right\rceil$. Let $(\mathsf{Enc}_{\mathsf{i\text{-}sss}}, \mathsf{Dec}_{\mathsf{i\text{-}sss}})$ be the encoding and decoding algorithms of a $[2N^{(1)} + \ell, \ell, N^{(1)}, N^{(1)} + 1]_{\mathbb{F},\mathbb{F}}$ secret sharing scheme with independence. Note that a permutation in $\mathcal{S}_{N^{(1)}}$ can be represented using $N^{(1)} \log N^{(1)}$ bits.

$\mathsf{Enc}(s \in \{0,1\}^L)$:

(a) Choose a permutation $\sigma \in \mathcal{S}_{N^{(1)}}$ at random.
(b) Let $c^{(1)}_{[N^{(1)}]} \sim \mathsf{Enc}_{\mathsf{basic}}(s)$.
(c) Let $c^{(2)}_{[N^{(2)}]} \sim \mathsf{Enc}_{\mathsf{i\text{-}sss}}(\sigma)$. Here we interpret σ as an element in $\mathbb{F}^{\ell}$; and the shares $\in \mathbb{F}^{2N^{(1)}+\ell}$ as an element in $\{0,1\}^{N^{(2)}}$.
(d) Output $c_{[N]} := (c^{(1)}_{[N^{(1)}]}, c^{(2)}_{[N^{(2)}]})$.

$\mathsf{Dec}(\tilde{c} \in \{0,1\}^N)$:

(a) Let $\left(\tilde{c}^{(1)}_{[N^{(1)}]}, \tilde{c}^{(2)}_{[N^{(2)}]}\right) \equiv \tilde{c}_{[N]}$.
(b) Decode $\tilde{c}^{(2)}_{[N^{(2)}]}$ by the decoding algorithm of $\mathsf{Dec}_{\mathsf{i\text{-}sss}}$ to obtain a permutation $\tilde{\sigma}$.
(c) Output $\tilde{s} = \mathsf{Dec}_{\mathsf{basic}}(\tilde{\sigma}^{-1}(\tilde{c}^{(1)}_{[N^{(1)}]}))$. (If either of the decoding algorithms fail, output $\perp$.)

Fig. 3. Main non-malleable code

$\mathsf{Tamper}^{(s)}_f$ in Fig. 1 to be over the coin tosses of not only the encoding function but also over the tampering function, and take care of this problem.

Theorem 3 (2-Phase Non-malleability). *There exists an* explicit *and* efficient *non-malleable code for* multi-bit *messages against the tampering class* $\mathcal{F}^*$.

5.1 Proof of Theorem 3

In this section we show that the basic encoding scheme described in Fig. 2 satisfies Theorem 3.

Useful Terminology. We will call the field elements of augmented secret sharing scheme $\mathsf{Enc}^{(\mathsf{out})}$ as "elements". The encoding of each element via inner encoding $\mathsf{Enc}^{(\mathsf{in})}$ will be called a "block" or "inner codeword". We shall visualize our inner code blocks as a two-dimensional objects, where each "column" represents the additive secret shares of a bit in the unary encoding scheme.

Below we crucially rely on the notion of "equivalence of codes" and "dirty inner codewords" or "dirty blocks" as defined below.

Equivalence of Codewords for our Scheme. Here we describe equivalence of codes for $\mathsf{Enc}^{(\mathsf{in})}$ and $\mathsf{Enc}^{(\mathsf{basic})}$.

1. Inner Codes. Two inner codewords, $g^{(\mathsf{in})}_{[um]}$ and $h^{(\mathsf{in})}_{[um]}$ are equivalent codes if they encode the same message according to the inner encoding scheme $\mathsf{Enc}^{(\mathsf{in})}$.

2. Non-Malleable Codes. Two codewords $g^{(\mathsf{basic})}_{[umn]}$ and $h^{(\mathsf{basic})}_{[umn]}$ are equivalent codes if their outer codes under $\mathsf{Enc}^{(\mathsf{out})}$ are identical[5]. That is, following holds. For all $i \in [n]$, define $g^{(\mathsf{in})}_i = g^{(\mathsf{basic})}_{(i-1)um+[um]}$ and $h^{(\mathsf{in})}_i = h^{(\mathsf{basic})}_{(i-1)um+[um]}$. Then, there exists a $\pi : [n] \to [n]$ such that for all $i \in [n]$, $g^{(\mathsf{in})}_i \cong h^{(\mathsf{in})}_{\pi(i)}$.

Criteria for Valid Inner Codeword or Block. For a block to be valid, the corresponding unary code should have parity 0 mod 3.

Classification of Blocks. We classify the inner codewords or blocks in following three categories.

Fixed Blocks. We say that a block is *completely fixed* if all its bits are obtained from bits in $I_0 \cup I_1$. That is, the tampering function explicitly writes down the whole block using bits from $I_0 \cup I_1$. Note that some of these bits might have been toggled.

Copied Blocks. We say that a block is *completely copied* if it is generated by copying one whole block in c and (possibly) performing column preserving permutations to the block. Also, even number of toggles might have been applied to any of these columns. Note that copied blocks are valid with probability 1.

Dirty Blocks. We say that an inner codeword or a block in $\tilde{c}$ is dirty if it is neither fixed nor copied. In other words, one of the following holds:

1. The block receives its bits partially from one block of c. To clarify, it can be the case that it receives bits from more than one blocks, or it receives bits from one block but some of its bits are obtained from $I_0 \cup I_1$.
2. (The block receives all its bits from one block of c but) The permutation within the block is not column preserving. That is, there exists a column which receives bits from more than one columns of the same inner codeword.
3. (The block receives all its bits from one block and the permutation is column preserving but) There exists a column which has odd number of toggle gates.

We show that a dirty block fails to be a valid block (according to inner encoding scheme) with at least a constant probability (see [3] for details).

We denote the number of dirty blocks by n_{dirty}, fixed blocks by n_{fixed}, and copied blocks by n_{copy}. Note that $n = n_{\mathsf{dirty}} + n_{\mathsf{fixed}} + n_{\mathsf{copy}}$. Finally, we define peeked blocks as follows:

Peeked Blocks. We say that a block is a peeked if one of its bits has been copied from I_p. Let n_{peek} be the number of such blocks. Note that $n_{\mathsf{peek}} \leqslant n_p$.

[5] Note that we only insist that $g^{(\mathsf{basic})}_{[umn]}$ and $h^{(\mathsf{basic})}_{[umn]}$ not only encode the same message s but also every outer codeword element is identical. Note that we allow for permutation of outer code elements.

5.2 Key Steps of the Proof

In this section, we give a high level proof idea by doing a case analysis on $n_0 + n_1$ and explaining how Sim is determined on each case (depending on map). The threshold value $\log^{10}\kappa$ chosen below for analysis is arbitrary; any suitable $\mathrm{poly}\log\kappa$ will suffice. For a formal proof of non-malleability, please refer to the full version [3].

Case 1. $\log^6\kappa \leqslant n_0 + n_1 \leqslant N - um\log^3\kappa$.
In this case, Sim outputs $\perp$ with probability 1. This incurs $\mathsf{negl}(\kappa)$ simulation error. This is because, as we can show, $n_{\mathsf{dirty}} \geqslant \log^3\kappa$ with probability $1-\mathsf{negl}(\kappa)$. Further, as described below, if n_{dirty} is large, then the probability that $\mathsf{Tamper}^{(s)}$ outputs something other than $\perp$ is negligible.

Case 2. $n_0 + n_1 \leqslant \log^6\kappa$.
Note that Sim has n_0, n_1, n_p and map. It begins by sampling I_0, I_1, I_p, c_{I_p} and $f \in \mathcal{F}$ according to map as in real execution. Based on these we have the following cases on n_{dirty}, the number of dirty blocks.

Case 2.1. $n_{\mathsf{dirty}} \geqslant \log^3\kappa$.
In this case, Sim outputs $\perp$ with probability 1. Again, as explained below, this incurs a negligible simulation error.

Case 2.2. $n_{\mathsf{dirty}} < \log^3\kappa$.
In this case, $n_{\mathsf{copy}} = n - n_{\mathsf{dirty}} - n_{\mathsf{fixed}} \geqslant n - \log^3\kappa - \log^6\kappa/um$. That is, the tampering function copies most of the blocks identically into the tampered codeword. Now, let n'_{copy} be the number of copied blocks which do not contain any bit from I_p. Then $n'_{\mathsf{copy}} \geqslant n_{\mathsf{copy}} - n_{\mathsf{peek}} \geqslant n - 2\log^6\kappa$.

So, the tampered codeword can either be invalid or (if valid) equivalent to the original codeword (because distance of the outer codeword $\gg \Theta(\log^6\kappa)$). Now the probability that the completely fixed, dirty and peeked blocks are each identical to their counterparts in the input codeword does not depend the message s because the privacy of the outer codeword is $\gg n_{\mathsf{dirty}} + n_{\mathsf{fixed}} + n_{\mathsf{peek}}$. This probability σ can be computed exhaustively by Sim and does not depend on the message s. So, given these $I_0, I_1, I_p, c_{I_p}, f$, Sim outputs same^* with probability σ; otherwise outputs $\perp$. This is clearly identical to the real execution random variable $\mathsf{Tamper}^{(s)}$ on $I_0, I_1, I_p, c_{I_p}, f$.

Case 3. $n_0 + n_1 \geqslant N - um\log^3\kappa$.
In this case, $n_{\mathsf{fixed}} = n - n_{\mathsf{dirty}} - n_{\mathsf{copy}} > n - \log^3\kappa$. In this case, the tampered code word is either invalid or (if valid) equivalent to the codeword consistent with the fixed inner codewords (because the distance of the outer encoding scheme is much greater than $\Theta(\log^3\kappa)$).

Now Sim samples I_0, I_1, I_p, c_{I_p} and $f \in \mathcal{F}$ according to map as in real execution. We say that (I_0, I_1) is *good* if it contains at least one bit from each column of c.

Since we have $n_0 + n_1 > N - um\log^3\kappa$, we can show that $\Pr[(I_0, I_1) \text{ is good}] = 1-\mathsf{negl}(\kappa)$. If (I_0, I_1) is not good, then we define Sim to output $\perp$ with probability

1 on this (I_0, I_1). This incurs additional $\mathsf{negl}(\kappa)$ expected simulation error over the choices of (I_0, I_1).

Now we are in the case that (I_0, I_1) is good. Given this, Sim first checks whether the fixed inner blocks can define a valid outer codeword. If not, then Sim outputs $\perp$. Simulation error in this case is 0.

Finally, we are in the case when (I_0, I_1) are good and the fixed blocks define a valid outer codeword, say g^* and a message s^*. Now Sim needs to check that remaining blocks are consistent with g^* or not. Note that since (I_0, I_1) is good, the bits restricted to $[N] \setminus (I_0 \cup I_1 \cup I_p)$ are uniform bits. This is because all proper subsets of any column are uniform random bits. Now the probability of forming a codeword for s^* can be exhaustively computed starting from uniform random bits for $[N] \setminus (I_0 \cup I_1 \cup I_p)$ and taking fixed value for bits in I_p as c_{I_p}. Let this probability be σ. Note that peeked blocks cannot intersect with set of fixed blocks. Now, Sim outputs s^* with probability σ and $\perp$ otherwise. The simulation error in this case is again 0.

Analysis for $n_{\mathsf{dirty}} \geqslant \log^3 \kappa$. In Case 1 and Case 2.1 above we relied on the claim that if n_{dirty} is large, then the probability of the tampered codeword being valid is negligible. To prove this, we need to consider different ways in which a dirty block can be formed. In all cases, it is easy to argue that each dirty block has a constant positive probability of being an invalid block (the unary codeword encoded by the block will not be 0 (mod 3)). However, these events need not be independent of each other. The main reason for dependence is that bits from a single block can be moved into two different blocks. Nevertheless, we can show that there is an ordering of the dirty blocks that at least $n_{\mathsf{dirty}}/2$ dirty blocks are "risky," i.e., the probability of it being an invalid block is a positive constant, *conditioned on all previous blocks being valid*. Technically, the i^{th} block (according to the given ordering) is risky if there is some block u in the original codeword, such that the i^{th} block as well as the j^{th} block, for some $j > i$, each has at least one bit copied to it from u; further a sufficiently large number of bits from u should be copied to blocks numbered i or larger. We can consider any arbitrary ordering, and its reverse ordering, and argue that every dirty block is risky in one of the two orderings. This guarantees that in one of the two orderings, there are at least $n_{\mathsf{dirty}}/2$ risky blocks. The above analysis assumes that the adversary does not see the bits from the original codeword that it copies, which is not true since it is given the bits in I_p. To account for this we restrict to orderings in which the n_{peek} blocks containing the bits from I_p appear at the beginning, and also define a block to be risky only if it is not one of these blocks. Since n_{peek} is much smaller than n_{dirty}, we can argue that the number of risky blocks remains large.

6 Application to Non-malleable String Commitment

For a bit b and auxiliary input z, let $\mathsf{STR}_b(\langle C, R\rangle, A, n, z)$ denote the output of the following experiment: on input $(1^n, z)$, A adaptively chooses two *strings* (v_0, v_1) of length n (where n is the security parameter), and receives a commitment to v_b while simultaneously A also interacts with a receiver, attempting to

commit to some value. Define $\widetilde{v}$ to be the value contained in the right commitment[6]. If A's commitment transcript on right is either not accepting, or identical to the transcript on left, then output a special symbol $\perp$; if $\widetilde{v} \in \{v_0, v_1\}$, output a special symbol same^*; otherwise, output $\widetilde{v}$.[7]

Definition 3 (Non-malleable String Commitments). *We say that $\langle C, R\rangle$ is a non-malleable string commitment scheme (for all strings in $\{0,1\}^n$) if for every PPT A and every $z \in \{0,1\}^*$ it holds that*

$$\mathsf{STR}_0(\langle C, R\rangle, A, n, z) \quad \approx_c \quad \mathsf{STR}_1(\langle C, R\rangle, A, n, z).$$

We remark that the definition above requires the message space to be large (i.e., at least super-polynomial in n). Otherwise, the definition cannot be achieved. This definition, however, is equivalent to a standard simulation-based formulation of non-malleability (see Theorem A.1 in [18]).

Below we give our construction for non-malleable string commitments from non-malleable bit commitments. We prove the security of our scheme in the full version [3].

Construction. Given a bit commitment scheme $\langle C, R\rangle$, we construct a string commitment scheme $\langle C', R'\rangle$ for $\{0,1\}^n$ as follows. Let $\mathsf{nm\text{-}code}$ be a non-malleable coding scheme for messages of length n that is robust to $\mathcal{F} := \mathcal{F}_{0,1} \circ \mathcal{S}_N$, and let $t(n)$ denote the length of the codewords for some fixed polynomial t. Let Enc and Dec be encoding and decoding algorithms. To commit to a string $v \in \{0,1\}^n$, C' generates a random codeword $w \leftarrow \mathsf{Enc}(v)$, and commits to each bit of w independently, and in parallel using the bit-commitment protocol $\langle C, R\rangle$. The receiver checks that no two bit-commitment transcripts, out of t such transcripts, are identical. If the check fails, or if any of the bit-commitment transcripts are not accepting, the receiver rejects; otherwise it accepts the commitment. To decommit to v, the receiver sends v along with decommitment information for each bit of w denoted by (w_i, d_i) for every $i \in [t]$; the receiver accepts v if and only if all recommitments verify and the resulting codeword decodes to v.

References

1. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. STOC (2015, to appear)
2. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: STOC, pp. 774–783 (2014)

[6] Note that $\widetilde{v}$ is unique w.h.p. and there exists $\widetilde{d}$ s.t. $\mathsf{open}(\widetilde{c}, \widetilde{v}, \widetilde{d}) = 1$ where $\widetilde{c}$ is the right commitment.

[7] Following [16], this definition allows MIM to commit to the same value. It is easy to prevent MIM from committing the same value generically in case of string commitments: convert the scheme to tag based by appending the tag with v, and then sign the whole transcript using the tag.

3. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Explicit non-malleable codes resistant to permutations and perturbations. Cryptology ePrint Archive, Report 2014/841 (2014). http://eprint.iacr.org/
4. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 375–397. Springer, Heidelberg (2015)
5. Broadbent, A., Tapp, A.: Information-theoretic security without an honest majority. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 410–426. Springer, Heidelberg (2007)
6. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: FOCS, pp. 541–550 (2010)
7. Chandran, N., Goyal, V., Mukherjee, P., Pandey, O., Upadhyay, J.: Block-wise non-malleable codes. Cryptology ePrint Archive, Report 2015/129 (2015). http://eprint.iacr.org/
8. Chandran, N., Kanukurthi, B., Ostrovsky, R.: Locally updatable and locally decodable codes. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 489–514. Springer, Heidelberg (2014)
9. Chattopadhyay, E., Zuckerman, D.: Non-malleable codes against constant split-state tampering. In: STOC, pp. 306–315 (2014)
10. Cheraghchi, M., Guruswami, V.: Capacity of non-malleable codes. In: ITCS, pp. 155–168 (2014)
11. Cheraghchi, M., Guruswami, V.: Non-malleable coding against bit-wise and split-state tampering. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 440–464. Springer, Heidelberg (2014)
12. Choi, S.G., Kiayias, A., Malkin, T.: BiTR: built-in tamper resilience. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 740–758. Springer, Heidelberg (2011)
13. Coretti, S., Maurer, U., Tackmann, B., Venturi, D.: From single-bit to multi-bit public-key encryption via non-malleable codes. Cryptology ePrint Archive, Report 2014/324 (2014). http://eprint.iacr.org/
14. Cramer, R., Dodis, Y., Fehr, S., Padró, C., Wichs, D.: Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 471–488. Springer, Heidelberg (2008)
15. Cramer, R., Padró, C., Xing, C.: Optimal algebraic manipulation detection codes in the constant-error model. Cryptology ePrint Archive, Report 2014/116 (2014). http://eprint.iacr.org/
16. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography (extended abstract). In: STOC, pp. 542–552 (1991)
17. Dziembowski, S., Kazana, T., Obremski, M.: Non-malleable codes from two-source extractors. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 239–257. Springer, Heidelberg (2013)
18. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: ICS, pp. 434–452 (2010)
19. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: Continuous non-malleable codes. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 465–488. Springer, Heidelberg (2014)
20. Faust, S., Mukherjee, P., Venturi, D., Wichs, D.: Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 111–128. Springer, Heidelberg (2014)

21. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: STOC, pp. 495–504 (2014)
22. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
23. Gordon, D., Ishai, Y., Moran, T., Ostrovsky, R., Sahai, A.: On complete primitives for fairness. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 91–108. Springer, Heidelberg (2010)
24. Goyal, V.: Constant round non-malleable protocols using one way functions. In: STOC, pp. 695–704 (2011)
25. Goyal, V., Lee, C., Ostrovsky, R., Visconti, I.: Constructing non-malleable commitments: a black-box approach. In: FOCS, pp. 51–60 (2012)
26. Guruswami, V., Smith, A.: Codes for computationally simple channels: explicit constructions with optimal rate. In: FOCS, pp. 723–732 (2010)
27. Hemenway, B., Ostrovsky, R.: Public-key locally-decodable codes. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 126–143. Springer, Heidelberg (2008)
28. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
29. Lin, H., Pass, R.: Constant-round non-malleable commitments from any one-way function. In: STOC, pp. 705–714 (2011)
30. Lipton, R.J.: A new approach to information theory. In: STACS, pp. 699–708 (1994)
31. Liu, F.-H., Lysyanskaya, A.: Tamper and leakage resilience in the split-state model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 517–532. Springer, Heidelberg (2012)
32. Micali, S., Peikert, C., Sudan, M., Wilson, D.A.: Optimal error correction against computationally bounded noise. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 1–16. Springer, Heidelberg (2005)
33. Myers, S., Shelat, A.: Bit encryption is complete. In: FOCS, pp. 607–616 (2009)
34. Ostrovsky, R., Pandey, O., Sahai, A.: Private locally decodable codes. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 387–398. Springer, Heidelberg (2007)
35. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: STOC, pp. 533–542 (2005)