Tight Upper and Lower Bounds for Leakage-Resilient, Locally Decodable and Updatable Non-Malleable Codes

Dana Dachman-Soled[⋆], Mukul Kulkarni, and Aria Shahverdi

University of Maryland, College Park, USA
{danadach@ece, mukul@terpmail, ariash@terpmail}.umd.edu

Abstract. In a recent result, Dachman-Soled et al. (TCC '15) proposed a new notion called locally decodable and updatable non-malleable codes, which informally, provides the security guarantees of a non-malleable code while also allowing for efficient random access. They also considered locally decodable and updatable non-malleable codes that are *leakage-resilient*, allowing for adversaries who continually leak information in addition to tampering. Unfortunately, the locality of their construction in the continual setting was $\Omega(\log n)$, meaning that if the original message size was n blocks, then $\Omega(\log n)$ blocks of the codeword had to be accessed upon each decode and update instruction.

In this work, we ask whether super-constant locality is inherent in this setting. We answer the question affirmatively by showing tight upper and lower bounds. Specifically, in any threat model which allows for a rewind attack—wherein the attacker leaks a small amount of data, waits for the data to be overwritten and then writes the original data back—we show that a locally decodable and updatable non-malleable code with block size $\mathcal{X} \in \text{poly}(\lambda)$ number of bits requires locality $\delta(n) \in \omega(1)$, where $n = \text{poly}(\lambda)$ is message length and λ is security parameter. On the other hand, we re-visit the threat model of Dachman-Soled et al. (TCC '15)—which indeed allows the adversary to launch a rewind attack—and present a construction of a locally decodable and updatable non-malleable code with block size $\mathcal{X} \in \Omega(\lambda^{1/\mu})$ number of bits (for constant $0 < \mu < 1$) with locality $\delta(n)$, for any $\delta(n) \in \omega(1)$, and $n = \text{poly}(\lambda)$.

1 Introduction

Non-malleable codes were introduced by Dziembowski, Pietrzak and Wichs [22] as a relaxation of error-correcting codes, and are useful in settings where privacy—but not necessarily correctness–is desired. Informally, a coding scheme is **non-malleable** against a tampering function if by tampering with the codeword, the function can either keep the underlying message unchanged or change it to an unrelated message. The main application of non-malleable codes proposed in

⋆ This work is supported in part by an NSF CAREER Award #CNS-1453045 and by a Ralph E. Powe Junior Faculty Enhancement Award.

the literature is for achieving security against leakage and tampering attacks on memory (so-called *physical attacks* or *hardware attacks*), although non-malleable codes have also found applications in other areas of cryptography [17,16,29] and theoretical computer science [12].

Standard non-malleable codes are useful for protecting small amounts of secret data stored on a device (such as a cryptographic secret key) but unfortunately are not suitable in settings where, say, an entire database must be protected. This is due to the fact that non-malleable codes do not allow for random access: Once the database is encoded via a non-malleable code, in order to access just a single location, the entire database must first be decoded, requiring a linear scan over the database. Similarly, in order to update a single location, the entire database must be decoded, updated and re-encoded. In a recent result, [18] proposed a new notion called locally decodable and updatable non-malleable codes, which informally speaking, provides the security guarantees of a non-malleable code while also allowing for efficient random access. In more detail, we consider a message $m = m_1, \ldots, m_n$ consisting of n blocks, and an encoding algorithm $\textsc{enc}(m)$ that outputs a codeword $\hat{C} = \hat{c}_1, \ldots, \hat{c}_{\hat{n}}$ consisting of $\hat{n}$ blocks. As introduced by Katz and Trevisan [35], local decodability means that in order to retrieve a single block of the underlying message, one does not need to read through the whole codeword but rather, one can access just a few blocks of the codeword. Similarly, local updatability means that in order to update a single block of the underlying messages, one only needs to update a few blocks of the codeword.

As observed by [18], achieving these locality properties requires a modification of the previous definition of non-malleability: Suppose a tampering function f only modifies one block of the codeword, then it is likely that the output of the decoding algorithm, $\textsc{dec}$, remains unchanged in most locations. (Recall $\textsc{dec}$ gets as input an index $i \in [n]$ and will only access a few blocks of the codeword to recover the i-th block of the message, so it may not detect the modification.) In this case, the (overall) decoding of the tampered codeword $f(\hat{C})$ (i.e. $(\textsc{dec}^{f(\hat{C})}(1), \ldots, \textsc{dec}^{f(\hat{C})}(n))$) can be highly related to the original message, which intuitively means it is highly malleable.

To handle this issue, [18] consider a more fine-grained experiment. Informally, they require that for any tampering function f (within some class), there exists a simulator that, after every update instruction, computes a vector of decoded messages $\boldsymbol{m}^*$, and a set of indices $\mathcal{I} \subseteq [n]$. Here $\mathcal{I}$ denotes the coordinates of the underlying messages that have been tampered with. If $\mathcal{I} = [n]$, then the simulator thinks that the decoded messages are $\boldsymbol{m}^*$, which should be unrelated to the most recent messages placed in each position by the updater. On the other hand, if $\mathcal{I} \subsetneq [n]$, the simulator thinks that all the messages not in $\mathcal{I}$ remain unchanged (equivalent to the *most recent values* placed there by the simulator or the original message, if no update has occurred in that position), while those in $\mathcal{I}$ become $\perp$. This intuitively means the tampering function can do only one of the following cases:

1. It destroys a block (or blocks) of the underlying messages while keeping the other blocks unchanged, OR
2. If it modifies a block of the underlying message to a valid encoding, then it must have modified *all* blocks to encodings of unrelated messages, thus destroying the original message.

It turns out, as shown by [18], that the above is sufficient for achieving tamper-resilience for RAM computations. Specifically, the above (together with an ORAM scheme) yields a compiler for any RAM program with the guarantee that any adversary who gets input/output access to the compiled RAM program Π running on compiled database D who can additionally apply tampering functions $f \in \mathcal{F}$ to the database D adaptively throughout the computation, learns no more than what can be learned given only input/output access to Π running on database D. Dachman-Soled et al. in [18] considered locally decodable and updatable non-malleable codes that are also *leakage-resilient*, thus allowing for adversaries who continually leak information about D in addition to tampering. The locality achieved by the construction of [18] is $\Theta(\log(n))$, meaning that when encoding messages of length n number of blocks, the decode and update procedures each require access to $\Theta(\log(n))$ number of blocks of the encoding. Thus, when using the encoding scheme of [18] to compile a RAM program into its secure version, the overhead is at least $\Omega(\log(n))$ memory accesses for each read/write access in the underlying program. In practice, such an overhead is often prohibitive. [1] In this work, we ask whether it is possible to construct leakage-resilient, locally decodable and updatable non-malleable codes that achieve significantly better locality.

Rewind attacks. When considering both leakage and tampering attacks (even just a single leakage query followed in a later round by a single tampering query) so-called *rewind attacks* become possible. In a rewind attack, the attacker does the following (1) **leak** information on only a "few" blocks of memory in rounds $1, \ldots, i$; (2) **wait** during rounds $i+1, \ldots, j$ until these memory locations are (with high probability) modified by the "updater" (the entity that models the honest computation on the data); (3) **re-write** the old information into these memory locations in round $j+1$, with the goal of causing the state of the computation to be *rewound.* Rewind attacks can be thwarted by ensuring that when the old information is written back, it becomes *inconsistent* with other positions of the codeword and an error is detected. On the other hand, a bad outcome of a rewind attack occurs if when decoding certain blocks of memory, with non-negligible probability, the old values from round i are recovered and no error is detected. This is a problem since such an outcome cannot be simulated

[1] Although the ORAM scheme used in the compiler also has $\omega(\log(n))$ overhead, in many applications of interest, properties of the specific RAM program can be leveraged so that the overhead of ORAM can be reduced such that it becomes practically feasible. On the other hand, the $\Theta(\log(n))$ overhead of the encoding scheme of [18] is entirely agnostic to the RAM program being run on top and thus, the high overhead would be incurred in all applications.

by a simulator as required in the security definition: The decoding of these blocks depends on the original message and yet is no longer equal to "same" (since the values decoded are not the most recent values placed in those positions by the updater).

1.1 Our Results

Our results show that any construction of locally decodable and updatable non-malleable codes in a threat model which allows for a rewind attack as above will require "high locality." Specifically, we show tight upper and lower bounds: (1) Every such construction will require super-constant locality, moreover; (2) Super-constant locality is sufficient for achieving constructions in the same threat model as [18] (which, as discussed, allows for rewind attacks). Throughout the paper, we assume that the decode and update procedures are *non-adaptive* in the sense that once an encoding scheme $\Pi = (\textsc{enc}, \textsc{dec})$ is specified, then for each $n \in \mathbb{N}$, the sets of codeword blocks $S_i := S_i^{\textsc{dec}} \cup S_i^{\textsc{up}}$ accessed in order to decode/update the i-th message block, $i \in [n]$, are fixed (and do not depend on the codeword $\hat{C}$). This is a natural requirement, which holds true for the encoding scheme of [18].

Specifically, we show the following:

Theorem 1 (Informal). *Let λ be security parameter and let $\Pi = (\textsc{enc}, \textsc{dec})$ be a locally decodable and updatable non-malleable code with non-adaptive decode and update which takes messages over alphabet Σ and outputs codewords over alphabet $\widehat{\Sigma}$, where $|\Sigma|, |\widehat{\Sigma}| \in \mathrm{poly}(\lambda)$, in a threat model which allows for a rewind attack. Then, for $n = \mathrm{poly}(\lambda)$, Π has locality $\delta(n) \in \omega(1)$.*

Moreover, for every $\delta(n) \in \omega(1)$, there exists a $\Pi = (\textsc{enc}, \textsc{dec})$ with non-adaptive decode and update in a threat model which allows for a rewind attack, which takes messages over alphabet Σ and outputs codewords over alphabet $\widehat{\Sigma}$, where $|\Sigma| \in \mathrm{poly}(\lambda)$ and $|\widehat{\Sigma}| \in \Omega(\lambda^{1/\mu})$ for constant $0 < \mu < 1$, such that for $n = \mathrm{poly}(\lambda)$, Π has locality $\delta(n)$.

Specifically, for the positive result, the construction of leakage resilient locally decodable updatable codes is secure against the same classes of tampering and leakage functions, $\mathcal{F}$, $\mathcal{G}$, as the construction of [18], but improves the locality from $O(\log n)$ to $\delta(n)$, for any $\delta(n) \in \omega(1)$.

We emphasize that, for the lower bound, our attack works even in a threat model which allows only a *single* bit of leakage in each round. We leave as an open question extending our lower bound to the setting where decode and update may be adaptive (i.e. the next position accessed by decode and/or update depends on the values read in the previous positions) or randomized.

1.2 Our Techniques

Lower Bound We assume that there exists a locally decodable and updatable non-malleable code with non-adaptive decode and update and constant locality,

c, for all message lengths $n = \mathrm{poly}(\lambda)$ (where n is the number of blocks in the message). We then arrive at contradiction by showing that for every constant c, there exists a constant $c' > c$, such that the security guarantee cannot hold when encoding messages of length $\mathcal{X}^{c'}$ number of blocks, where $\mathcal{X} \in \mathrm{poly}(\lambda)$ is the bit length of the codeword blocks. Specifically, for messages of length $n := \mathcal{X}^{c'} \in \mathrm{poly}(\lambda)$ number of blocks, we will present an explicit attacker and an explicit updater for which there cannot exist a simulator as required by the definition of locally decodable and updatable non-malleable codes.

The attack we present is a *rewind* attack, as discussed before. Intuitively, the main difficulty of designing the attack is to determine *which* positions of the codeword are to be leaked and subsequently re-wound to their original values so that with high probability in the real game, the corresponding message block will decode (with no error detected) to the original value in that position, as opposed to the most recently updated value. For purposes of our attack, we assume that the original message is either equal to 0 in all n blocks or equal to 1 in all n blocks.

Sunflower Lemma. For $i \in [n]$, let the sets $S_i \subseteq [\hat{n}]$ correspond to the blocks (where each block has size $\mathcal{X} \in \mathrm{poly}(\lambda)$ bits) of the codeword accessed in order to decode/update the i-th block of the message. Note that by the locality assumption, the size of each set S_i is $|S_i| = c$. We use the Sunflower Lemma of Erdős and Rado [24] to choose constant c' large enough such that when the message is of length $n := \mathcal{X}^{c'}$ number of blocks, we are guaranteed to have a Sunflower $\mathsf{SF} := \{S_{i_0}, S_{i_1}, \ldots, S_{i_k}\}$, where $i_0, \ldots, i_k \in [n]$, of size $k+1$, where $k \gg \mathcal{X} \cdot c$. A sunflower is a collection of sets such that the intersection of any pair is equal to the core core, i.e. $S_{i_j} \cap S_{i_\ell} = \mathsf{core}$ for all $j \neq \ell$. There exists k petals, $S_{i_j} \setminus \mathsf{core}$, and it is required that none of them are empty. See sections 3.1, 3.2 for more details.

The Compression Function. Given a *fixed* initial codeword $\hat{C}$ and sunflower SF (as defined above) we define a (randomized) compression function $F_{\hat{C}} : \{0, 1, \mathsf{same}\}^k \to \{0, 1\}^{\mathcal{X} \cdot c}$ which takes as input values $x_1, \ldots, x_k \in \{0, 1, \mathsf{same}\}$ indicating how to update (or not) the corresponding message block i_j, $j \in [k]$, where S_{i_j} is in the sunflower. Specifically, for $j = 1$ to k: If $x_j = \mathsf{same}$, message block i_j does not get updated. Otherwise $\textsc{update}^{\hat{C}}(i_j, x_j)$ is executed. The output of the function $F_{\hat{C}}$ is the contents of the sunflower core, core, after all the updates have been completed. Note that core can consist of at most c codeword blocks since $\mathsf{core} \subseteq S_{i_j}$ for all $j \in [k]$. Therefore, the output length of $F_{\hat{C}}$ is at most $\mathcal{X} \cdot c$ bits. Note that this means that $F_{\hat{C}}$ is a compression function, since we chose $k \gg \mathcal{X} \cdot c$. Now this, in turn, means that the output of $F_{\hat{C}}$ cannot contain all of the information in its input. Indeed, it can be shown (cf. [20]) that with high probability over the choice of $j^* \in [k]$, the two distributions $F_{\hat{C}}(X_1, \ldots, X_{j^*-1}, \mathsf{same}, X_{j^*+1}, \ldots, X_k)$ and $F_{\hat{C}}(X_1, \ldots, X_{j^*-1}, X_{j^*}, X_{j^*+1}, \ldots, X_k)$ are statistically close when each X_j, $j \in [k]$ is chosen uniformly at random from $\{0, 1, \mathsf{same}\}$. See sections 3.1, 3.3, 3.4 for more details.

The Attacker and the Updater. The *attacker* first finds the sunflower $\mathsf{SF} := \{S_{i_0}, S_{i_1}, \ldots, S_{i_k}\}$ in polynomial time and then chooses $j^* \in [k]$ at random. In the first round (or multiple rounds if the attacker is allowed only a single bit of leakage) the attacker leaks the contents of the positions in $\hat{C}$ corresponding to decoding of i_{j^*} ($S_{i_{j^*}}$), minus the contents of the blocks in the core of the sunflower. We denote the entire leaked information by y_{j^*}. The attacker then writes those same values, y_{j^*}, back in the $k+1$-st round. The *updater* chooses values $x_1, \ldots, x_k \in \{0, 1, \mathsf{same}\}$ and in each round from 1 to k, requests the corresponding update (i.e. update message block i_j to 0, if $x_j = 0$, update to 1 if $x_j = 1$ and do not update this block at all, if $x_j = \mathsf{same}$). See section 3.5 for more details.

Putting it All Together. Note that the input to the decoding algorithm when decoding position i_{j^*} is exactly: $(y_{j^*}, F_{\hat{C}_0}(X_1, \ldots, X_{j^*-1}, X_{j^*}, X_{j^*+1}, \ldots, X_k))$ (the contents of the positions in $\hat{C}$ corresponding to decoding of i_{j^*}, minus the contents of the blocks in the core of the sunflower, and the core itself). Additionally, note that since $\{S_{i_0}, S_{i_1}, \ldots, S_{i_k}\}$ form a sunflower, if $x_{j^*} = \mathsf{same}$, then the rewind attack has *no effect* (since the blocks in $S_{i_{j^*}} \setminus \mathsf{core}$ were not accessed during any update request) and so decode on input $(y_{j^*}, F_{\hat{C}_0}(X_1, \ldots, X_{j^*-1}, \mathsf{same}, X_{j^*+1}, \ldots, X_k))$ must correctly output 1 if the original encoding was 1 and 0 if the original encoding was 0 (without outputting $\perp$). Since $F_{\hat{C}}$ is a compression function, it means that with high probability decode on input $(y_{j*}, F_{\hat{C}}(X_1, \ldots, X_{j^*-1}, X_{j^*}, X_{j^*+1}, \ldots, X_k))$ will output 1 if the original encoding was 1 and 0 if the original encoding was 0, regardless of the value of X_{j^*}. Intuitively, since the output of decode now depends on the *original* message block in the i_{j^*}-th position, as opposed to the *most recently updated* value, the simulator must fail in at least one of the two cases (either when the original message was 0 or 1) and so the encoding scheme cannot satisfy the non-malleability definition. See section 3.6 for more details.

Upper Bound Here we take advantage of the fact that codeword blocks are large–$\mathcal{X} \in \Omega(\lambda^{1/\mu})$ number of bits, for constant $0 < \mu < 1$–to replace the Merkle Tree used in the original construction of [18] with an alternative data structure we call a t-slice Merkle Tree. Note that the $\Omega(\log \lambda)$ locality of the construction of [18] came from the fact that an entire path (and siblings) of the binary Merkle tree from root to leaf of length $\log(n)$ had to be traversed for each decode and update instruction. Our new data structure is a $t := \mathcal{X}^{1-\mu}$-ary tree for constant $0 < \lambda < 1$ and uses as a building block a collision resistant hash function $h : \{0,1\}^{\mathcal{X}} \to \{0,1\}^{\mathcal{X}^{\mu}}$ (note h has output length $\mathcal{X}^{\mu} \in \Omega(\lambda)$) and so, for messages of length $n = \mathrm{poly}(\lambda)$ blocks, an entire path of the tree from root to leaf will always have length less than $\delta(n)$, for any $\delta(n) \in \omega(1)$. Moreover, the root of the tree can be updated and verified without reading any of the siblings along the path from root to leaf, due to the use of a carefully constructed hash function with a specific structure. This allows us to achieve a locally decodable and updatable code with locality $\delta(n)$, for any $\delta(n) \in \omega(1)$. See section 4 for more details.

1.3 Related Work

Non-Malleable Codes. The concept of non-malleability was introduced by Dolev, Dwork and Naor [19] and has been applied widely in cryptography since. It has since been studied in both the computational as well as the information-theoretic setting. Error-correcting codes and early works on tamper resilience [28,32] gave rise to the study of non-malleable codes. The notion of non-malleable codes was formalized in the seminal work of Dziembowski, Pietrzak and Wichs [22]. Split state classes of tampering functions introduced by Liu and Lysyanskaya [37], have subsequently received a lot of attention with a sequence of improvements achieving reduced number of states, improved rate, or adding desirable features to the scheme [21,3,11,2,6,1]. Recently [7,5] gave efficient constructions of non-malleable codes for "non-compartmentalized" tampering function classes. Other works on non-malleable codes include [25,15,4,33,8,2,10]. We guide the interested reader to [34] and [37] for illustrative discussion of various models for tamper and leakage resilience. There are also several inefficient, existential or randomized constructions for much more general classes of functions (sometimes presented as efficient constructions in a random-oracle model) in addition to those above [22,14,27].

Locally Decodable Codes. The idea of *locally decodable* codes was introduced by Katz and Trevisan in [35], when they considered the possibility of recovering the message by looking at a limited number of bits from a (possibly) corrupted encoding obtained from an error correcting code. They also showed the impossibility of achieving the same for schemes with linear encoding length. This work was followed by [38,23,13] who achieved constant locality with super-polynomial code length, while on the other hand locally decodable codes with constant rate and sub-linear locality have been constructed by [36,30,31]. We refer the interested reader to [39], a survey on locally decodable codes by Yekhanin.

Locally Updatable and Locally Decodable Codes. The notion of *locally updatable and locally decodable codes* was introduced by Chandran et al. in [9] where the constraint of *locality*, i.e. restricting the number of bits accessed, is also applied to updating any codeword obtained from encoding of another message. They gave information theoretic construction with amortized update locality of $\mathcal{O}(\log^2 k)$ and read locality of (super-linear) polynomial in k, where k is the length of input message. Another variant called *locally updatable and locally decodable-detectable codes* was also introduced in the same work which ensures that decoding never outputs an incorrect message. Chandran et al. in [9] gave the construction of such codes in computational setting with poly-logarithmic locality.

Locally Decodable and Updatable Non-Malleable Codes. Dachman-Soled et al. in [18] introduced the notion of *locally decodable and updatable non-malleable* *codes* and presented a construction in the computational setting. The construction of [18] also achieves leakage resilience in addition to the tamper resilience. Dachman-Soled et al. in [18] then used this notion to construct compilers that transform any RAM machine into a RAM machine secure against leakage and tampering. This application was also studied by Faust et al. [26], who presented a different approach which does not use locally decodable and updatable non-malleable codes. Recently, Chandran et al. [10] gave a construction of locally decodable and updatable non-malleable codes in the information-theoretic setting. However, they addressed only the one-time leakage and tampering case, and to achieve continual leakage and tampering, require a periodic refresh of the entire memory. The locality of their construction is super-constant, thus affirming our results.

Bounds on Non-Malleable Codes. Cheragachi and Guruswami [14] studied the "capacity" of non-malleable codes in order to understand the optimal bounds on the efficiency of non-malleable codes. This work has been instrumental in asserting the claims of efficient constructions for non-malleable codes since then (cf. [1,5,6]). We note that our work is the first study establishing similar tight bounds for the locality of the *locally decodable and updatable non-malleable codes.*

2 Definitions

Definition 1 (Locally Decodable and Updatable Code). *Let $\Sigma, \hat{\Sigma}$ be sets of strings, and $n, \hat{n}, p, q$ be some parameters. An $(n, \hat{n}, p, q)$ locally decodable and updatable coding scheme consists of three algorithms* $(\textsc{enc}, \textsc{dec}, \textsc{update})$ *with the following syntax:*

- *The encoding algorithm* $\textsc{enc}$ *(*perhaps randomized*) takes input an n-block (in Σ) message and outputs an $\hat{n}$-block (in $\hat{\Sigma}$) codeword.*
- *The (local) decoding algorithm* $\textsc{dec}$ *takes input an index in $[n]$, reads at most p blocks of the codeword, and outputs a block of message in Σ. The overall decoding algorithm simply outputs* $(\textsc{dec}(1), \textsc{dec}(2), \ldots, \textsc{dec}(n))$.
- *The (local) updating algorithm* $\textsc{update}$ *(*perhaps randomized*) takes inputs an index in $[n]$ and a string in $\Sigma \cup \{\epsilon\}$, and reads/writes at most q blocks of the codeword. Here the string ϵ denotes the procedure of refreshing without changing anything.*

Let $\hat{C} \in \hat{\Sigma}^{\hat{n}}$ be a codeword. For convenience, we denote $\textsc{dec}^{\hat{C}}, \textsc{update}^{\hat{C}}$ *as the processes of reading/writing individual block of the codeword, i.e. the codeword oracle returns or modifies individual block upon a query. Here we view $\hat{C}$ as a random access memory where the algorithms can read/write to the memory $\hat{C}$ at individual different locations. In binary settings, we often set $\Sigma = \{0,1\}^{\kappa}$ and $\hat{\Sigma} = \{0,1\}^{\hat{\kappa}}$.*

Definition 2 (Correctness). *An $(n, \hat{n}, p, q)$ locally decodable and updatable coding scheme (with respect to $\Sigma, \hat{\Sigma}$) satisfies the following properties. For any message $M = (m_1, m_2, \ldots, m_n) \in \Sigma^n$, let $\hat{C} = (\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{\hat{n}}) \leftarrow \textsc{enc}(M)$ be a codeword output by the encoding algorithm. Then we have:*

- *for any index $i \in [n]$, $\Pr[\text{DEC}^{\hat{C}}(i) = m_i] = 1$, where the probability is over the randomness of the encoding algorithm.*
- *for any update procedure with input $(j, m') \in [n] \times \Sigma \cup \{\epsilon\}$, let $\hat{C}'$ be the resulting codeword by running $\text{UPDATE}^{\hat{C}}(j, m')$. Then we have $\Pr[\text{DEC}^{\hat{C}'}(j) = m'] = 1$, where the probability is over the encoding and update procedures. Moreover, the decodings of the other positions remain unchanged.*

Remark 1. The correctness definition can be directly extended to handle any sequence of updates.

Definition 3 (Continual Tampering and Leakage Experiment). *Let k be the security parameter, $\mathcal{F}, \mathcal{G}$ be some families of functions. Let $(\text{ENC}, \text{DEC}, \text{UPDATE})$ be an $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$. Let $\mathcal{U}$ be an updater that takes input a message $M \in \Sigma^n$ and outputs an index $i \in [n]$ and $m \in \Sigma$. Then for any blocks of messages $M = (m_1, m_2, \ldots, m_n) \in \Sigma^n$, and any (non-uniform) adversary $\mathcal{A}$, any updater $\mathcal{U}$, define the following continual experiment $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$:*

- *The challenger first computes an initial encoding $\hat{C}^{(1)} \leftarrow \text{ENC}(M)$.*
- *Then the following procedure repeats, at each round j, let $\hat{C}^{(j)}$ be the current codeword and $M^{(j)}$ be the underlying message:*
  - *$\mathcal{A}$ sends either a tampering function $f \in \mathcal{F}$ and/or a leakage function $g \in \mathcal{G}$ to the challenger.*
  - *The challenger replaces the codeword with $f(\hat{C}^{(j)})$, or sends back a leakage $\ell^{(j)} = g(\hat{C}^{(j)})$.*
  - *We define $\boldsymbol{m}^{(j)} \stackrel{\text{def}}{=} \left(\text{DEC}^{f(\hat{C}^{(j)})}(1), \ldots, \text{DEC}^{f(\hat{C}^{(j)})}(n)\right)$.*
  - *Then the updater computes $(i^{(j)}, m) \leftarrow \mathcal{U}(\boldsymbol{m}^{(j)})$ for the challenger.*
  - *Then the challenger runs $\text{UPDATE}^{f(\hat{C}^{(j)})}(i^{(j)}, m)$ and sends the index $i^{(j)}$ to $\mathcal{A}$.*
  - *$\mathcal{A}$ may terminate the procedure at any point.*
- *Let t be the total number of rounds above. At the end, the experiment outputs*

$$\left(\ell^{(1)}, \ell^{(2)}, \ldots, \ell^{(t)}, \boldsymbol{m}^{(1)}, \ldots, \boldsymbol{m}^{(t)}, i^{(1)}, \ldots, i^{(t)}\right).$$

Definition 4 (Non-malleability and Leakage Resilience against Continual Attacks). *An $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$ is continual non-malleable against $\mathcal{F}$ and leakage resilient against $\mathcal{G}$ if for all PPT (non-uniform) adversaries $\mathcal{A}$, and PPT updaters $\mathcal{U}$, there exists some PPT (non-uniform) simulator $\mathcal{S}$ such that for any $M = (m_1, \ldots, m_n) \in \Sigma^n$, $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$ is (computationally) indistinguishable to the following ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$:*

- *The experiment proceeds in rounds. Let $M^{(1)} = M$ be the initial message.*
- *At each round j, the experiment runs the following procedure:*
  - *At the beginning of each round, $\mathcal{S}$ outputs $(\ell^{(j)}, \mathcal{I}^{(j)}, \boldsymbol{w}^{(j)})$, where $\mathcal{I}^{(j)} \subseteq [n]$.*

– *Define*

$$\boldsymbol{m}^{(j)} = \begin{cases} \boldsymbol{w}^{(j)} & \text{if } \mathcal{I}^{(j)} = [n] \\ \boldsymbol{m}^{(j)}|_{\mathcal{I}^{(j)}} := \bot, \boldsymbol{m}^{(j)}|_{\bar{\mathcal{I}}^{(j)}} := M^{(j)}|_{\bar{\mathcal{I}}^{(j)}} & \text{otherwise,} \end{cases}$$

where $\boldsymbol{x}|_{\mathcal{I}}$ denotes the coordinates $\boldsymbol{x}[v]$ where $v \in \mathcal{I}$, and the bar denotes the complement of a set.

– *The updater runs $(i^{(j)}, m) \leftarrow \mathcal{U}(\boldsymbol{m}^{(j)})$ and sends the index $i^{(j)}$ to the simulator. Then the experiment updates $M^{(j+1)}$ as follows: set $M^{(j+1)} := M^{(j)}$ for all coordinates except $i^{(j)}$, and set $M^{(j+1)}[i^{(j)}] := m$.*

– *Let t be the total number of rounds above. At the end, the experiment outputs*

$$\left(\ell^{(1)}, \ell^{(2)}, \ldots, \ell^{(t)}, \boldsymbol{m}^{(1)}, \ldots, \boldsymbol{m}^{(t)}, i^{(1)}, \ldots, i^{(t)}\right).$$

3 Lower Bound

In this section we prove the following theorem:

Theorem 2. *Let λ be security parameter and let $\Pi = (\text{ENC}, \text{DEC})$ be a locally decodable and updatable non-malleable code with non-adaptive decode and update which takes messages over alphabet Σ and outputs codewords over alphabet $\widehat{\Sigma}$, where $\log|\Sigma|, \log|\widehat{\Sigma}| \in \text{poly}(\lambda)$, in a threat model which allows for a rewind attack. Then, for $n := n(\lambda) \in \text{poly}(\lambda)$, Π has locality $\delta(n) \in \omega(1)$.*

We denote by $\mathcal{X} := \log|\widehat{\Sigma}| \in \text{poly}(\lambda)$ the number of bits in each block of the codeword. For purposes of the lower bound, we can take $\mathcal{X}$ to be any polynomial in λ (or smaller).

In the following, we assume that $\Pi = (\text{ENC}, \text{DEC})$ is a locally decodable and updatable non-malleable code with non-adaptive decode and update and with *constant* locality. We then present an efficient rewind attacker along with an updater that break the security of Π, thus proving the theorem.

3.1 Attack Preliminaries

Definition 5 (Sunflower). *A sunflower (or Δ-system) is a collection of sets S_i for $1 \leq i \leq k$ such that the intersection of any two set is core Y, i.e. $S_i \cap S_j = \mathsf{core}$ for all $i \neq j$. There exists k petals $S_i \setminus \mathsf{core}$ and it's required that none of them are empty. A family of pairwise disjoint sets form a sunflower with an empty core.*

The following famous lemma is due to Erdős and Rado.

Lemma 1 (Sunflower Lemma [24]). *Let $\mathcal{F}$ be family of sets each of cardinality s. If $|\mathcal{F}| > s!(k-1)^s$ then $\mathcal{F}$ contains a sunflower with k petals.*

Definition 6 (Statistical Distance). *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two distribution over a shared universe of outcomes. let $supp(\mathcal{D})$ be the set of values assumed by $\mathcal{D}$ with nonzero probability, and let $\mathcal{D}(u) := \Pr[\mathcal{D} = u]$. The statistical distance of $\mathcal{D}_1$ and $\mathcal{D}_2$ is defined as*

$$||\mathcal{D}_1 - \mathcal{D}_2||_{stat} := \frac{1}{2} \sum_{u \in supp(\mathcal{D}_1) \cup supp(\mathcal{D}_2)} |\mathcal{D}_1(u) - \mathcal{D}_2(u)|.$$

Definition 7 (Distributional Stability [20]). *Let $\mathcal{U}$ be a finite universe and $t, n \geq 1$ be integers. Let $\mathcal{D}_i$ for $1 \leq i \leq t$ be a collection of t mutually independent distributions over $\{0,1\}^n$ and F be a possibly-randomized mapping $F(x^1, \ldots, x^t) : \{0,1\}^{n \times t} \to \mathcal{U}$, for $j \in [t]$ let*

$$\gamma_j := \mathop{\mathbb{E}}_{y \sim \mathcal{D}_j}[||F(\mathcal{D}_1, \ldots, \mathcal{D}_{j-1}, y, \mathcal{D}_{j+1}, \ldots, \mathcal{D}_t) - F(\mathcal{D}_1, \ldots, \mathcal{D}_t)||_{stat}].$$

F is δ-distributionally stable for $\delta \in [0,1]$ with respect to $\mathcal{D}_1, \ldots, \mathcal{D}_t$ if

$$\frac{1}{t} \sum_{j=1}^{t} \gamma_j \leq \delta.$$

Lemma 2 (Compression Functions are Distributionally Stable [20]). *Let $R(x^1, \ldots, x^t) : \{0,1\}^{n \times t} \to \{0,1\}^{\leq t'}$ be any possibly-randomized mapping, for any $n, t, t' \in \mathbb{N}^+$. R is δ-distributionally stable with respect to any independent input distributions $\mathcal{D}_1, \ldots, \mathcal{D}_t$, where it may take either of the following two bounds:*

1. $\delta := \sqrt{\frac{\ln 2}{2} \cdot \frac{t'+1}{t}}$
2. $\delta := 1 - 2^{-\frac{t'}{t} - 3}$.

3.2 Applying the Sunflower Lemma

For $i \in [n]$, the sets $S_i \subseteq [\hat{n}]$ correspond to the blocks (each of size $\mathcal{X}$) of the codeword accessed in order to update/decode m_i (i.e. the set $S_i := S_i^{\text{DEC}} \cup S_i^{\text{UP}}$, where S_i^{DEC}, S_i^{UP} are the sets of blocks accessed by the decode and update procedures, respectively). By hypothesis, we have that for $i \in [n]$, $|S_i| = c$, for constant c. Choose $n = \mathcal{X}^{c'} \in \text{poly}(\lambda)$, where c' is a constant such that

$$\mathcal{X}^{c'} > c! \cdot (22{,}500 \cdot c \cdot \mathcal{X})^c$$

Then by the Sunflower Lemma, $\{S_1, \ldots, S_n\}$ contains a sunflower with $k + 1 := 22{,}500 \cdot c \cdot \mathcal{X} + 1$ petals. Let $\mathsf{SF} := \{S_{i_0}, S_{i_1}, \ldots, S_{i_k}\}$, where $i_0, \ldots, i_k \in [n]$. For codeword $\hat{C}$, Let $\mathsf{core}(\hat{C})$ denote the content of the set of blocks that make up the core of the sunflower. For set S_ℓ, $\ell \in [n]$, let $\mathsf{set}_\ell(\hat{C})$ denote the content of the blocks in set S_ℓ.

3.3 The Compression Functions

Given a *fixed* initial codeword $\hat{C}$, sunflower $\mathsf{SF} := \{S_{i_0}, \ldots, S_{i_k}\}$, where $i_0, \ldots, i_k \in [n]$ (as defined above) with $k+1 := 22,500 \cdot c \cdot \mathcal{X} + 1$ petals, define the following (randomized) function $F_{\hat{C}} : \{0, 1, \mathsf{same}\}^k \to \{0,1\}^{\mathcal{X} \cdot c}$ as follows:

- On input $x_1, \ldots, x_k \in \{0, 1, \mathsf{same}\}$
- For $j = 1$ to k:
  - If $x_j = \mathsf{same}$, run $\textsc{update}^{\hat{C}}(i_0, 0)$.
  - Otherwise run $\textsc{update}^{\hat{C}}(i_j, x_j)$.

  where $\hat{C}$ denotes the current codeword at any point in time.
- Run $\textsc{update}^{\hat{C}}(i_0, 0)$.
- Output the contents of $\mathsf{core}(\hat{C})$.

3.4 Closeness of Distributions

For $\ell \in [k]$, let X_ℓ be a random variable distributed as X, where X is distributed as $U_{\{0,1,\mathsf{same}\}}$, i.e. its value is chosen uniformly from the set $\{0, 1, \mathsf{same}\}$. Let $\hat{C}_0 \leftarrow \textsc{enc}(0 \ldots 0)$ and $\hat{C}_1 \leftarrow \textsc{enc}(1 \ldots 1)$. Let $y_j^0 := \mathsf{set}_{i_j}(\hat{C}_0) \setminus \mathsf{core}(\hat{C}_0)$ denote the contents of the positions in $\hat{C}_0$ corresponding to decoding of i_j, minus the contents of the blocks in the core of the sunflower. Similarly, let $y_j^1 := \mathsf{set}_{i_j}(\hat{C}_1) \setminus \mathsf{core}(\hat{C}_1)$ denote the contents of the positions in $\hat{C}_1$ corresponding to decoding of i_j, minus the contents of the blocks in the core of the sunflower. We prove the following claim, which will be useful in the subsequent analysis.

Claim 3.1. For every $\hat{C}_0 \leftarrow \textsc{enc}(0 \ldots 0)$ and $\hat{C}_1 \leftarrow \textsc{enc}(1 \ldots 1)$, we have that:

- With probability at least 0.8 over $j \sim [k]$, the statistical distance between $(y_j^0, F_{\hat{C}_0}(X_1, \ldots, X_{j-1}, \mathsf{same}, X_{j+1}, \ldots, X_k))$ and $(y_j^0, F_{\hat{C}_0}(X_1, \ldots, X_k))$ is at most 0.1.
- With probability at least 0.8 over $j \sim [k]$, the statistical distance between $(y_j^1, F_{\hat{C}_1}(X_1, \ldots, X_{j-1}, \mathsf{same}, X_{j+1}, \ldots, X_k))$ and $(y_j^1, F_{\hat{C}_1}(X_1, \ldots, X_k))$ is at most 0.1.

Proof. First, by Lemma 2 and the fact that $F_{\hat{C}}$ is a compression function, we have that for every codeword $\hat{C}$:

$$\frac{1}{k}\sum_{j=1}^{k} \mathop{\mathbb{E}}_{x \sim X}[||F_{\hat{C}}(X_1, \ldots, X_{j-1}, x, X_{j+1}, \ldots, X_k) - F_{\hat{C}}(X_1, \ldots, X_k)||_{stat}] < \sqrt{\frac{c \cdot \mathcal{X}}{k}}.$$

By linearity of expectation, we have

$$\mathop{\mathbb{E}}_{x \sim X}\left[\frac{1}{k}\sum_{j=1}^{k} \left(||F_{\hat{C}}(X_1, \ldots, X_{j-1}, x, X_{j+1}, \ldots, X_k) - F_{\hat{C}}(X_1, \ldots, X_k)||_{stat}\right)\right] < \sqrt{\frac{c \cdot \mathcal{X}}{k}}.$$

Now, by Markov's inequality, we have that

$$\frac{1}{k}\sum_{j=1}^{k}\left(||F_{\hat{C}}(X_1,\ldots,X_{j-1},\mathsf{same},X_{j+1},\ldots,X_k) - F_{\hat{C}}(X_1,\ldots,X_k)||_{stat}\right) < 3\sqrt{\frac{c\cdot\mathcal{X}}{k}}.$$

Applying Markov's inequality again, we have that with probability at least 0.8 over choice of $j \sim [k]$,

$$||F_{\hat{C}}(X_1,\ldots,X_{j-1},\mathsf{same},X_{j+1},\ldots,X_k) - F_{\hat{C}}(X_1,\ldots,X_k)||_{stat} < 15\cdot\sqrt{\frac{c\cdot\mathcal{X}}{k}} = 0.1,$$

where the final equality holds since we take $k+1 := 22,500\cdot c\cdot\mathcal{X}+1$. Finally, since the above holds for every $\hat{C}$, we have that for every $\hat{C}_0 \leftarrow \textsc{enc}(0\ldots0)$, and $\hat{C}_1 \leftarrow \textsc{enc}(1\ldots1)$:

- With probability at least 0.8 over $j \sim [k]$, the statistical distance between $F_{\hat{C}_0}(X_1,\ldots,X_{j-1},\mathsf{same},X_{j+1},\ldots,X_k)$ and $F_{\hat{C}_0}(X_1,\ldots,X_k)$ is at most 0.1.
- With probability at least 0.8 over $j \sim [k]$, the statistical distance between $F_{\hat{C}_1}(X_1,\ldots,X_{j-1},\mathsf{same},X_{j+1},\ldots,X_k)$ and $F_{\hat{C}_1}(X_1,\ldots,X_k)$ is at most 0.1.

The above implies that for every $\hat{C}_0 \leftarrow \textsc{enc}(0\ldots0)$ and $\hat{C}_1 \leftarrow \textsc{enc}(1\ldots1)$, we have that with probability at least 0.8 over $j \sim [k]$, the statistical distance between $(y_j^0, F_{\hat{C}_0}(X_1,\ldots,X_{j-1},\mathsf{same},X_{j+1},\ldots,X_k))$ and $(y_j^0, F_{\hat{C}_0}(X_1,\ldots,X_k))$ is at most 0.1, and with probability at least 0.8 over $j \sim [k]$, the statistical distance between $(y_j^1, F_{\hat{C}_1}(X_1,\ldots,X_{j-1},\mathsf{same},X_{j+1},\ldots,X_k))$ and $(y_j^1, F_{\hat{C}_1}(X_1,\ldots,X_k))$ is at most 0.1, since y_j^0, y_j^1 can be deduced from $\hat{C}_0$, $\hat{C}_1$, respectively, and $\hat{C}_0$, $\hat{C}_1$ are part of the description of the functions. This concludes the proof of the claim.

3.5 The Attack

In this section we describe the polynomial-time attacker and updater:

Description of attacker:

- Find the Sunflower $\mathsf{SF} := \{S_{i_0},\ldots,S_{i_k}\}$, where $i_0,\ldots,i_k \in [n]$ and $k+1 := 22,500\cdot c\cdot\mathcal{X}+1$, contained in $\{S_1,\ldots,S_n\}$ in $O(n^2)$ time.[2]
- Choose $j^* \sim [k]$
- In the first round, submit leakage function $\ell(\hat{C})$ defined as $\ell(\hat{C}) := \mathsf{set}_{i_{j^*}}(\hat{C}) \setminus \mathsf{core}(\hat{C})$ which returns Leaked, i.e. the contents of the positions in $\hat{C}$ corresponding to decoding of i_{j^*}, minus the contents of the blocks in the core of the sunflower.[3]

[2] This can be done by finding the pairwise intersection $S_i \cap S_j$ for all $i,j \in [n]$, yielding sets $\mathsf{core}_1,\ldots,\mathsf{core}_{n^2}$ and the sorting these sets lexicographically. The core of the sunflower $\mathsf{core} := \mathsf{core}_i$, where core_i is the most frequently appearing core. The petals are the corresponding sets that share that pairwise intersection.

[3] If the attacker may leak only a single bit per round, we instead add here $r < \mathcal{X}\cdot c$ number of rounds where in each round the attacker leaks a single bit from

– Wait until the $k+1$-st round. In the $k+1$-st round, choose tampering function f which replaces the contents of $\mathsf{set}_{i_{j^*}}(\hat{C}^{(k)}) \setminus \mathsf{core}(\hat{C}^{(k)})$, i.e. the positions in $\hat{C}^{(k)}$ corresponding to decoding of i_{j^*}, minus the contents of the blocks in the core of the sunflower, with the values, Leaked, that were leaked via ℓ.

Description of Updater:

– Choose $x_1, \ldots, x_k \sim \{0, 1, \mathsf{same}\}^k$.
– For $j = 1$ to k:
  - If $x_j = \mathsf{same}$, request $\text{UPDATE}^{\hat{C}^{(j)}}(i_0, 0)$
  - Otherwise request $\text{UPDATE}^{\hat{C}^{(j)}}(i_j, x_j)$

  where $\hat{C}^{(j)}$ denotes the current codeword in round j.
– In round $j > k$, request $\text{UPDATE}^{\hat{C}^{(j)}}(i_0, 0)$.

3.6 Attack Analysis

Let J^* be the random variable corresponding to choice of j^* in the attack described above. For $j \in [k]$, let UP_{i_j} be the event that location i_j gets updated and let $\overline{\text{UP}_{i_j}}$ be the event that location i_j does not get updated. Recall that for $j \in [k]$, m_{i_j} denotes the original message in block i_j. We have the following properties, which can be verified by inspection:

Fact 1.

(a) For $j \in [k]$, $\Pr[\text{UP}_{i_j} \mid m_{i_j} = 0] = \Pr[\text{UP}_{i_j} \mid m_{i_j} = 1] = 0.67$; $\Pr[\overline{\text{UP}_{i_j}} \mid m_{i_j} = 0] = \Pr[\overline{\text{UP}_{i_j}} \mid m_{i_j} = 1] = 0.33$.
(b) For $j \in [k]$, if the i_j-th block of original message was a $m_{i_j} = 0$, then conditioned on an update occurring on block i_j, $m_{i_j}^{(k)} = 0$ with probability 0.5 and $m_{i_j}^{(k)} = 1$ with probability 0.5. Conditioned on no update occurring on block i_j, $m_{i_j}^{(k)} = 0$ with probability 1.
(c) For $j \in [k]$, if the i_j-th block of original message was a $m_{i_j} = 1$, then conditioned on an update occurring on block i_j, $m_{i_j}^{(k)} = 1$ with probability 0.5 and $m_{i_j}^{(k)} = 0$ with probability 0.5. Conditioned on no update occurring on block i_j, $m_{i}^{(k)} = 1$ with probability 1.

We next present the main technical claim of this section:

Claim 3.2. For the attack and updater specified in Section 3.5:

Case 1: If the original message was $\boldsymbol{m} = \mathbf{0}$, then with probability at least 0.7, $m_{i_{J^*}}^{(k+1)} = 0$.
Case 2: If the original message was $\boldsymbol{m} = \mathbf{1}$, then with probability at least 0.7, $m_{i_{J^*}}^{(k+1)} = 1$.

$\mathsf{set}_{i_{j^*}}(\hat{C}) \setminus \mathsf{core}(\hat{C})$. During each of these rounds, the updater requests a "dummy" update, $\text{UPDATE}^{\hat{C}^{(j)}}(i_0, 0)$.

We first show how to use Claim 3.2 to complete the proof of Theorem 2 and then present the proof of Claim 3.2.

Proof (of Theorem 2.). We show that the above claim implies that the candidate scheme is not secure under Definition 3 and Definition 4. Definition 4 requires the existence of a simulator $\mathcal{S}$ which (for the above attack and updater) outputs one of $\{\mathsf{same}, \perp\} \cup \{0,1\}^\kappa$ for the decoding of position i in round $k+1$. Recall that if $\mathcal{S}$ outputs same, then the output of the experiment in the corresponding position, denoted $m^{(k+1)}_{i_{j^*},\mathcal{S}}$, is set to $m^{(k+1)}_{i_{j^*},\mathcal{S}} := m^{(k)}_{i_{j^*}}$. We begin by defining the following notation for each $j \in [k]$:

$$\begin{aligned}
p^0_{up,j} &:= \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \mid m_{i_j} = 0 \wedge \textsc{up}_{i_j}] \\
p^1_{up,j} &:= \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \mid m_{i_j} = 1 \wedge \textsc{up}_{i_j}] \\
p^0_{\overline{up},j} &:= \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \mid m_{i_j} = 0 \wedge \overline{\textsc{up}_{i_j}}] \\
p^0_{0,j} &:= \Pr[\mathcal{S} \text{ outputs } 0 \mid m_{i_j} = 0] \\
p^1_{0,j} &:= \Pr[\mathcal{S} \text{ outputs } 0 \mid m_{i_j} = 1]
\end{aligned}$$

Note that since $\mathcal{S}$ does not see the original message, we have that for each $j \in [k]$:

$$(a)\, p^0_{up,j} = p^1_{up,j} \qquad (b)\, p^0_{0,j} = p^1_{0,j}. \tag{1}$$

Additionally we have, for each $j \in [k]$::

$$\begin{aligned}
&\Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m^{(k)}_{i_j} = 0 \wedge \textsc{up}_{i_j} \mid m_{i_j} = 0] \\
&\quad = \Pr[\textsc{up}_{i_j} \mid m_{i_j} = 0] \cdot \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \mid m_{i_j} = 0 \wedge \textsc{up}_{i_j}] \\
&\qquad \cdot \Pr[m^{(k)}_{i_j} = 0 \mid m_{i_j} = 0 \wedge \textsc{up}_{i_j}] \\
&\quad = 0.67 \cdot p^0_{up,j} \cdot 0.5,
\end{aligned} \tag{2}$$

Where the first equality follows since ($\mathcal{S}$ outputs $\mathsf{same} \mid m_{i_j} = 0 \wedge \textsc{up}_{i_j}$) and ($m^{(k)}_{i_j} = 0 \mid m_{i_j} = 0 \wedge \textsc{up}_{i_j}$) are independent events and the last line follows from Fact 1, items (a) and (b). Similarly, for each $j \in [k]$:

$$\begin{aligned}
&\Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m^{(k)}_{i_j} = 0 \wedge \textsc{up}_{i_j} \mid m_{i_j} = 1] \\
&\quad = \Pr[\textsc{up}_{i_j} \mid m_{i_j} = 1] \cdot \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \mid m_{i_j} = 1 \wedge \textsc{up}_{i_j}] \\
&\qquad \cdot \Pr[m^{(k)}_{i_j} = 0 \mid m_{i_j} = 1 \wedge \textsc{up}_{i_j}] \\
&\quad = 0.67 \cdot p^1_{up,j} \cdot 0.5 \\
&\quad = 0.67 \cdot p^0_{up,j} \cdot 0.5,
\end{aligned} \tag{3}$$

where the second to last line follows from Fact 1, items (a) and (c), and the last line follows due to (1)(a). Moreover, we have for each $j \in [k]$:

$$\begin{aligned}
&\Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m^{(k)}_{i_j} = 0 \wedge \overline{\textsc{up}_{i_j}} \mid m_{i_j} = 0] \\
&\quad = \Pr[\overline{\textsc{up}_{i_j}} \mid m_{i_j} = 0] \cdot \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \mid m_{i_j} = 0 \wedge \overline{\textsc{up}_{i_j}}] \\
&\quad = 0.33 \cdot p^0_{\overline{up},j},
\end{aligned} \tag{4}$$

where the last line follows from Fact 1, item (a). Finally, for each $j \in [k]$:

$$\Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m_{i_j}^{(k)} = 0 \wedge \overline{\mathrm{UP}_{i_j}} \mid m_{i_j} = 1] = 0. \tag{5}$$

Given Claim 3.2, in order for $\mathcal{S}$ to succeed, if the original message was $\boldsymbol{m} = \mathbf{0}$, then $m_{i_{J^*},\mathcal{S}}^{(k+1)}$ must be equal to 0 with probability (nearly) 0.7, whereas if the original message was $\boldsymbol{m} = \mathbf{1}$, then $m_{i_{J^*},\mathcal{S}}^{(k+1)}$ must be equal to 1 with probability (nearly) 0.7. Thus we have that:

$$\begin{aligned} 0.7 &= \sum_{j\in[k]} \Pr[J^* = j] \cdot \Pr[m_{i_j,\mathcal{S}}^{(k+1)} = 0 \mid m_{i_j} = 0] \\ &= \sum_{j\in[k]} \frac{1}{k} \cdot (\Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m_{i_j}^{(k)} = 0 \wedge \mathrm{UP}_{i_j} \mid m_{i_j} = 0] \\ &\qquad + \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m_{i_j}^{(k)} = 0 \wedge \overline{\mathrm{UP}_{i_j}} \mid m_{i_j} = 0] \\ &\qquad + \Pr[\mathcal{S} \text{ outputs } 0 \mid m_{i_j} = 0]) \\ &= \sum_{j\in[k]} \frac{1}{k} \cdot (0.67 \cdot p_{up,j}^0 \cdot 0.5 + 0.33 \cdot p_{\overline{up},j}^0 + p_{0,j}^0), \end{aligned} \tag{6}$$

where the last line follows due to (2) and (4). On the other hand we have:

$$\begin{aligned} 0.3 &\geq \sum_{j\in[k]} \Pr[J^* = j] \cdot \Pr[m_{i_j,\mathcal{S}}^{(k+1)} = 0 \mid m_{i_j} = 1] \\ &= \sum_{j\in[k]} \frac{1}{k} \cdot (\Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m_{i_j}^{(k)} = 0 \wedge \mathrm{UP}_{i_j} \mid m_{i_j} = 1] \\ &\qquad + \Pr[\mathcal{S} \text{ outputs } \mathsf{same} \wedge m_{i_j}^{(k)} = 0 \wedge \overline{\mathrm{UP}_{i_j}} \mid m_{i_j} = 1] \\ &\qquad + \Pr[\mathcal{S} \text{ outputs } 0 \mid m_{i_j} = 1]) \\ &= \sum_{j\in[k]} \frac{1}{k} \cdot (0.67 \cdot p_{up,j}^0 \cdot 0.5 + p_{0,j}^1) \\ &= \sum_{j\in[k]} \frac{1}{k} \cdot (0.67 \cdot p_{up,j}^0 \cdot 0.5 + p_{0,j}^0). \end{aligned} \tag{7}$$

where the second to last line follows due to (3) and (5) and the last line follows due to (1)(b). But subtracting (7) from (6), this implies that $0.33 \cdot \sum_{j\in[k]} \frac{1}{k} \cdot p_{\overline{up},j}^0 \geq 0.4$, which is impossible since for each $j \in [k]$, $p_{\overline{up},j} \leq 1$. Thus we have reached contradiction and so the theorem is proved.

We conclude by proving the Claim.

Proof (of Claim 3.2). The proof of the claim relies on the fact that decode takes as input $\mathtt{DEC}(y_{j^*}^0, F_{\hat{C}_0}(X_1, \ldots, X_k))$ in Case 1 and $\mathtt{DEC}(y_{j^*}^1, F_{\hat{C}_1}(X_1, \ldots, X_k))$ in Case 2, where $y_j^0 := \mathsf{set}_{i_j}(\hat{C}_0) \setminus \mathsf{core}(\hat{C}_0)$ denotes the contents of the positions in

$\hat{C}_0$ corresponding to decoding of i_j, minus the contents of the blocks in the core of the sunflower, and similarly, $y_j^1 := \mathsf{set}_{i_j}(\hat{C}_1) \setminus \mathsf{core}(\hat{C}_1)$ denotes the contents of the positions in $\hat{C}_1$ corresponding to decoding of i_j, minus the contents of the blocks in the core of the sunflower.

But note that, due to the structure of the Sunflower, updates to positions $i_0, \ldots, i_{j^*-1}, i_{j^*+1}, \ldots, i_k$ do not modify the contents of $\mathsf{set}_{i_{j^*}}(\hat{C}_0) \setminus \mathsf{core}(\hat{C}_0)$ (and $\mathsf{set}_{i_{j^*}}(\hat{C}_1) \setminus \mathsf{core}(\hat{C}_1)$) and so $\mathtt{DEC}(y_{j^*}^0, F_{\hat{c}_0}(X_1, \ldots, X_{j^*-1}, \mathsf{same}, X_{j^*+1}, \ldots, X_k)) = 0$ with overwhelming probability and $\mathtt{DEC}(y_{j^*}^1, F_{\hat{c}_1}(X_1, \ldots, X_{j^*-1}, \mathsf{same}, X_{j^*+1}, \ldots, X_k)) = 1$ with overwhelming probability, since when $X_j = \mathsf{same}$, the rewind attack has no effect and decode outputs the original message.

Moreover, we have shown in Claim 3.1 that for every $\hat{C}_0 \leftarrow \textsc{enc}(0 \ldots 0)$ and $\hat{C}_1 \leftarrow \textsc{enc}(1 \ldots 1)$, we have that:

1. With probability at least 0.8 over $j^* \sim [k]$, the statistical distance between $(y_j^0, F_{\hat{C}_0}(X_1, \ldots, X_{j-1}, \mathsf{same}, X_{j+1}, \ldots, X_k))$ and $(y_j^0, F_{\hat{C}_0}(X_1, \ldots, X_k))$ is at most 0.1.
2. With probability at least 0.8 over $j^* \sim [k]$, the statistical distance between $(y_j^1, F_{\hat{C}_1}(X_1, \ldots, X_{j-1}, \mathsf{same}, X_{j+1}, \ldots, X_k))$ and $(y_j^1, F_{\hat{C}_1}(X_1, \ldots, X_k))$ is at most 0.1.

Hence with each will not be satisfied with probability at most 0.2. Now, conditioned on each being satisfied, it can be concluded from (1) that the probability of $\mathtt{DEC}(y_j^0, F_{\hat{C}_0}(X_1, \ldots, X_k)) = 1$ is at most 0.1. Similarly from (2), $\mathtt{DEC}(y_j^1, F_{\hat{C}_1}(X_1, \ldots, X_k)) = 0$ with probability at most 0.1. Taking a union bound, we have that in each case, DEC procedure will fail to output the original message with probability at most 0.3. This means that with probability at least 0.7 over all coins, $\mathtt{DEC}(y_{j^*}^0, F_{\hat{C}_0}(X_1, \ldots, X_k)) = 0$, whereas with probability at least 0.7 over all coins $\mathtt{DEC}(y_{j^*}^1, F_{\hat{C}_1}(X_1, \ldots, X_k)) = 1$, completing the proof of the claim.

4 Matching Upper Bound

In this section we show how to construct a locally updatable and decodable non-malleable code with super-constant locality. This is achieved by replacing the Merkle Tree in the construction presented in [18] by a new data structure, *t-slice Merkle Tree* which we defined below (see Definition 8). Intuitively, the locality of updating/decoding in the construction given by Dachman-Soled et al. [18] is lower-bounded by the depth of the Merkle Tree, since, in order to detect tampering, each update/decode instruction must check the consistency of a leaf by traversing the path from leaf to root. Our initial idea is to replace the binary Merkle Tree of depth $\log(n)$ with a t-ary Merkle tree (where t is a super-constant function of n defined below) of constant depth. Unfortunately, this simple solution does not quite work. Recall that in order to verify consistency of a leaf in a standard Merkle tree, one needs to access not only the path from leaf to root, but also the *siblings* of each node on the path. This would mean that in the

t-ary tree, we would need to access at least $\Omega(t)$ sibling nodes, where t is super-constant, thus still requiring super-constant locality. Our solution, therefore, is to construct t-ary Merkle trees of a particular form, where verifying consistency of a leaf can be done by traversing only the path from leaf to root, *without* accessing any sibling nodes. We call such trees t-slice Merkle trees. Details of the construction follow in Definitions 8, 9, 10 and 11. Finally, in Theorem 3 we show that the t-slice Merkle Tree is collision resistant, which allows us to retain security while replacing the Merkle tree in the construction of [18] with our t-slice Merkle Tree. This then leads to our matching upper bound in Theorem 4.

Definition 8 (t-slice Merkle Tree). *Let $\mathcal{X}$ and $h : \{0,1\}^{\mathcal{X}} \rightarrow \{0,1\}^{\mathcal{X}/t}$ be a hash function that maps a block of size $\mathcal{X}$ to block of size $\mathcal{X}/t$. Let a block of data at level j with index i denoted by α_i^j and $M = (m_1, m_2, \ldots, m_n)$ being the input data and set $\alpha_i^0 := m_{i+1}$ for $0 \leq i \leq n-1$. A t-slice Merkle Tree $\mathsf{Tree}_h^t(M)$ is defined recursively in the following way:*

- *Bottom layer of the tree contains n blocks of data each of size $\mathcal{X}$, i.e., $(\alpha_0^0, \alpha_1^0, \ldots, \alpha_{n-1}^0)$.*
- *To compute the content of non-leaf node at level j with index i set $\alpha_i^j := h(\alpha_{i\cdot t}^{j-1})||\ldots||h(\alpha_{((i+1)\cdot t)-1}^{j-1})$.*
- *Once a single block α_i^j remains, set the root of Merkle Tree $\mathsf{Root}_h^t(M) := h(\alpha_i^j)$ and the height of tree $\mathcal{H} := j+1$ and terminate.*

For $k \in [0, \ldots, t-1]$, we denote the k-th slice of α_i^j by $\alpha_i^j[k]$ The internal blocks of Merkle Tree (including the root) are denoted as $\mathsf{Tree}_h^t(M)$.

Definition 9 (Path). *Given a Merkle Tree $\mathsf{Tree}_h^t(M)$ with n leaves of height $\mathcal{H}$ and its root $\mathsf{Root}_h^t(M)$, a path $p_i := p_i^0, \ldots, p_i^{\mathcal{H}-1}$, for $i \in [0, \ldots n-1]$ is a sequence of $\mathcal{H}$ blocks from leaf to root defined as follows: For $j \in [0, \ldots, \mathcal{H}-1]$, $p_i^j := \alpha_\ell^j$, where $\ell := \sum_{k=j}^{\mathcal{H}-1} \beta_k \cdot t^{k-j}$ and $\beta_{\mathcal{H}-1}, \ldots, \beta_0$ is the base t representation of i, where $\beta_{\mathcal{H}-1}$ is the most significant digit and β_0 is the least significant digit.*

Definition 10 (Consistency). *Let $\beta_{\mathcal{H}-1}, \ldots, \beta_0$ be the base t representation of i, where $\beta_{\mathcal{H}-1}$ is the most significant digit and β_0 is the least significant digit. Path $p_i := p_i^0, \ldots, p_i^{\mathcal{H}-1}$ is consistent with $\mathsf{Root}_h^t(M)$ if the following hold:*

- *$p_i^{\mathcal{H}-1} = \mathsf{Root}_h^t(M)$.*
- *For $j \in [\mathcal{H}-2]$, $h(p_i^j) = p_i^{j+1}[\ell \mod t]$, where $\ell := \sum_{k=j}^{\mathcal{H}-1} \beta_k \cdot t^{k-j}$ (i.e. the hash of the j-th element on the path is equal to the $(\ell \mod t)$-th slice of the $j+1$-st element on the path).*

Definition 11 (Update). *Given a path $p_i := p_i^0, \ldots, p_i^{\mathcal{H}-1}$ in Merkle Tree $\mathsf{Tree}_h^t(M)$ and new message block ${\alpha'}_i^0$, Let $\beta_{\mathcal{H}-1}, \ldots, \beta_0$ be the base t representation of i, where $\beta_{\mathcal{H}-1}$ is the most significant digit and β_0 is the least significant digit. The update procedure computes a modified path $p'_i := {p'}_i^0, \ldots, {p'}_i^{\mathcal{H}-1}$ as follows (the rest of the tree remains the same):*

- $p'^{0}_{i} := \alpha'^{0}_{i}$.
- *For* $j \in [1, \dots, \mathcal{H}-1]$, $p'^{j+1}_{i}[\ell \mod t] := h(p'^{j}_{i})$, *where* $\ell := \sum_{k=j}^{\mathcal{H}-1} \beta_k \cdot t^{k-j}$ *(i.e. the* $(\ell \mod t)$*-th slice of the* $j+1$*-st element on the path is equal to the hash of the* j*-th element on the path).*
- *For* $j \in [\mathcal{H}-1]$, $\gamma \in [0, \dots, t] \setminus \{\ell \mod t\}$, *where* $\ell := \sum_{k=j}^{\mathcal{H}-1} \beta_k \cdot t^{k-j}$, $p'^{j+1}_{i}[\gamma] := p^{j+1}_{i}[\gamma]$ *(i.e. all other slices of the* $j+1$*-st element on the path stay the same as in the original path* p_i*).*

Lemma 3. *Let* $\mathcal{X} \in \Omega(\lambda^{1/\mu})$, $h : \{0,1\}^{\mathcal{X}} \to \{0,1\}^{\mathcal{X}^{\mu}}$, *and* $t := \mathcal{X}^{1-\mu}$, *for constant* $0 < \mu < 1$. *Assuming* $n = \mathrm{poly}(\lambda) := \mathcal{X}^{c}$ *for constant* c, *the height of the* t*-slice Merkle Tree will be constant* $\mathcal{H} = \frac{c-1}{1-\mu}$.

Proof. In the beginning the message blocks $M = (m_1, m_2, \dots, m_n)$ are at the leaves of the tree and size of each block is $\mathcal{X}$, i.e. $|m_i| = \mathcal{X}$. After applying a hash function to each of the blocks separately, their size becomes $\mathcal{X}^{\mu}$ and by concatenating $\mathcal{X}^{1-\mu}$ number of hashes a single block of size $\mathcal{X}$ will be formed. In this level there will therefore be $\frac{\mathcal{X}^{c}}{\mathcal{X}^{1-\mu}} = \mathcal{X}^{c+\mu-1}$ block of size $\mathcal{X}$. Applying hash function to each of them will form new blocks of size $\mathcal{X}^{\mu}$ and there will be $\mathcal{X}^{c+2\mu-2}$ blocks of size $\mathcal{X}$. In general in level i-th there will be $\mathcal{X}^{c+i\mu-i}$ blocks of size $\mathcal{X}$. The root of the t-slice Merkle Tree is of size $\mathcal{X}$, so the height of the tree is for the case where $\mathcal{X}^{c+i\mu-i} = \mathcal{X}$ resulting the i and hence the height of tree is $\frac{c-1}{1-\mu}$.

Theorem 3. *Let* $\mathcal{X} \in \Omega(\lambda^{1/\mu})$, $h : \{0,1\}^{\mathcal{X}} \to \{0,1\}^{\mathcal{X}^{\mu}}$, *and* $t := \mathcal{X}^{1-\mu}$, *for constant* $0 < \mu < 1$. *Assuming* h *is a collision resistant hash function, consider the resulting* t*-slice Merkle Tree. Then for any message* $M = (m_1, m_2, \dots, m_n)$ *with* $m_i \in \{0,1\}^{\mathcal{X}}$, *any polynomial time adversary* $\mathcal{A}$,

$$\Pr\left[(m'_i, p_i) \leftarrow \mathcal{A}(M, h) : m'_i \neq m_i, p_i \textit{ is a consistent path with } \mathsf{Root}^{t}_{h}(M)\right] \leq \mathsf{negl}(k).$$

Moreover, given a path p_i *passing the leaf* m_i, *and a new value* m'_i, *the update algorithm computes* $\mathsf{Root}^{t}_{h}(M')$ *in constant time* $\mathcal{H} := \frac{c-1}{1-\mu}$, *where* $M' = (m_1, \dots, m_{i-1}, m'_i, m_{i+1}, \dots, m_n)$.

Proof. The second part of Theorem 3 is immediate by inspection of Definition 11.

For the first part of the theorem, we assume towards contradiction that for some message $M = (m_1, m_2, \dots, m_n)$ with $m_i \in \{0,1\}^{\mathcal{X}}$, there is an efficient adversary $\mathcal{A}$ such that

$$\Pr\left[(m'_i, p'_i) \leftarrow \mathcal{A}(M, h) : m'_i \neq m_i, p'_i \text{ is a consistent path with } \mathsf{Root}^{t}_{h}(M)\right] = 1/\mathrm{poly}(\lambda).$$

We construct adversary $\mathcal{A}'$ which finds a collision in hash function h. The procedure is as follows:

- On input h, adversary $\mathcal{A}'$ instantiates $\mathcal{A}$ on input (M, h).
- Adversary $\mathcal{A}$ returns (m'_i, p'_i), where $p'_i := p'^{0}_{i}, \dots, p'^{\mathcal{H}-1}_{i}$.

– $\mathcal{A}'$ checks that ${p'}_i^{\mathcal{H}-1} = \mathsf{Root}_h^t(M)$.
– For $j \in [\mathcal{H}-2]$, if ${p'}_i^{j+1} = p_i^{j+1}$, ${p'}_i^j \neq p_i^j$ and $h({p'}_i^j) = {p'}_i^{j+1}[\ell \mod t]$, where $\ell := \sum_{k=j}^{\mathcal{H}-1} \beta_k \cdot t^{k-j}$, then $\mathcal{A}'$ returns collision $({p'}_i^j, p_i^j)$.

Note that if $m'_i \neq m_i$, then ${p'}_i \neq p_i$ and so at some point the "if statement" above must hold. Moreover, if p'_i is a consistent path, then it must be the case that ${p'}_i^{\mathcal{H}-1} = \mathsf{Root}_h^t(M)$ and for $j \in [\mathcal{H}-2]$, $h({p'}_i^j) = {p'}_i^{j+1}[\ell \mod t]$, where $\ell := \sum_{k=j}^{\mathcal{H}-1} \beta_k \cdot t^{k-j}$, by definition of consistency. Thus, the above adversary $\mathcal{A}'$ will succeeds with same probability as the adversary $\mathcal{A}$ and breaks collision resistance of h with probability $1/\text{poly}(\lambda)$. Thus, we arrive at contradiction and so the theorem is proved.

Theorem 4. *Assume there exists a semantically secure symmetric encryption scheme, and a non-malleable code against the tampering function class $\mathcal{F}$, and leakage resilient against the function class $\mathcal{G}$. Then there exists a leakage resilient, locally decodable and updatable coding scheme that is non-malleable against continual attacks of the tampering class*

$$\bar{\mathcal{F}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} f : \hat{\Sigma}^{2n+1} \to \hat{\Sigma}^{2n+1} \textit{ and } |f| \leq \text{poly}(k), \textit{ such that :} \\ f = (f_1, f_2),\ f_1 : \hat{\Sigma}^{2n+1} \to \hat{\Sigma},\ f_2 : \hat{\Sigma}^{2n} \to \hat{\Sigma}^{2n}, \\ \forall (x_2, \ldots, x_{2n+1}) \in \hat{\Sigma}^{2n}, f_1(\ \cdot\ , x_2, \ldots, x_{2n+1}) \in \mathcal{F}, \\ f(x_1, x_2, \ldots, x_{2n+1}) = (f_1(x_1, x_2, \ldots, x_{2n+1}), f_2(x_2, \ldots, x_{2n+1}))\,. \end{array} \right\},$$

and is leakage resilient against the class

$$\bar{\mathcal{G}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} g : \hat{\Sigma}^{2n+1} \to \mathcal{Y} \textit{ and } |g| \leq \text{poly}(k), \textit{ such that :} \\ g = (g_1, g_2),\ g_1 : \hat{\Sigma}^{2n+1} \to \mathcal{Y}',\ g_2 : \hat{\Sigma}^{2n} \to \hat{\Sigma}^{2n}, \\ \forall\ (x_2, \ldots, x_{2n+1}) \in \hat{\Sigma}^{2n}, g_1(\ \cdot\ , x_2, \ldots, x_{2n+1}) \in \mathcal{G}. \end{array} \right\}.$$

Moreover, for $n := \mathcal{X}^c \in \text{poly}(\lambda)$, the coding scheme has locality $\delta(n)$, for any $\delta(n) \in \omega(1)$.

Our construction is exactly the same as that of Dachman-Soled et al. [18], except we replace their (standard) Merkle tree with our t-slice Merkle tree with the parameters described above. We note that the only property of the Merkle hash used in the security proof of [18] is the "collision resistance" property, analogous to our Theorem 3 above for the t-slice Merkle tree. Thus, our security proof follows exactly as theirs does and we therefore omit the full proof. On the other hand, as described in Definitions 11 and 10, updates and consistency checks require time and number of accesses to memory proportional to the height of the tree, $\mathcal{H}$, which is $\frac{c-1}{1-\mu}$ for our choice of parameters, as shown in Lemma 3 above. Since $n = \mathcal{X}^c \in \text{poly}(\lambda)$, it means that the height of the tree will always be less than $\delta(n)$, for any $\delta(n) \in \omega(1)$. On the other hand, [18] used a standard (binary) Merkle tree with height $\Theta(\log n)$. Therefore, while [18] requires locality $\Theta(\log n)$, we achieve locality $\delta(n)$, for any $\delta(n) \in \omega(1)$.

Finally, we give a concrete example of the resulting leakage and tampering classes we can tolerate via Theorem 4 when instantiating the underlying non-malleable code with a concrete construction. Specifically, we consider instantiating the underlying non-malleable code with the construction of of Liu and Lysyanskaya [37], which achieves both leakage and tamper resilience for split-state functions. Combining the constructions of [37] and [18] yields codewords consisting of $2n + 1$ blocks. We next describe the leakage and tampering classes $\bar{\mathcal{G}}, \bar{\mathcal{F}}$ that can be tolerated on the $2n + 1$-block codeword. $\bar{\mathcal{G}}$ consists of leakage functions g such that g restricted to the first block (i.e. g_1) is any (poly-sized) length-bounded split-state function; g_2 on the other hand, can leak all other parts. $\bar{\mathcal{F}}$ consists of tampering functions f such that f restricted to the first block (i.e. f_1) is any (poly-sized) split-state function. On the other hand f restricted to the rest (i.e. f_2) is any poly-sized function. We also remark that the function f_2 itself can depend on the split-state leakage on the first part.

References

1. Aggarwal, D., Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Optimal computational split-state non-malleable codes. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A, Part II. LNCS, vol. 9563, pp. 393–417. Springer, Heidelberg (Jan 2016)
2. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th ACM STOC. pp. 459–468. ACM Press (Jun 2015)
3. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 774–783. ACM Press (May / Jun 2014)
4. Aggarwal, D., Dziembowski, S., Kazana, T., Obremski, M.: Leakage-resilient non-malleable codes. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 398–426. Springer, Heidelberg (Mar 2015)
5. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Explicit non-malleable codes against bit-wise tampering and permutations. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 538–557. Springer, Heidelberg (Aug 2015)
6. 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 (Mar 2015)
7. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes for bounded depth, bounded fan-in circuits. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 881–908. Springer, Heidelberg (May 2016)
8. 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/2015/129`
9. 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 (Feb 2014)
10. Chandran, N., Kanukurthi, B., Raghuraman, S.: Information-theoretic local non-malleable codes and their applications. In: Kushilevitz, E., Malkin, T. (eds.)

TCC 2016-A, Part II. LNCS, vol. 9563, pp. 367–392. Springer, Heidelberg (Jan 2016)

11. Chattopadhyay, E., Zuckerman, D.: Non-malleable codes against constant split-state tampering. In: 55th FOCS. pp. 306–315. IEEE Computer Society Press (Oct 2014)
12. Chattopadhyay, E., Zuckerman, D.: Explicit two-source extractors and resilient functions. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 670–683. ACM Press (Jun 2016)
13. Chee, Y.M., Feng, T., Ling, S., Wang, H., Zhang, L.F.: Query-efficient locally decodable codes of subexponential length. computational complexity 22(1), 159–189 (2013), `http://dx.doi.org/10.1007/s00037-011-0017-1`
14. Cheraghchi, M., Guruswami, V.: Capacity of non-malleable codes. In: Naor, M. (ed.) ITCS 2014. pp. 155–168. ACM (Jan 2014)
15. 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 (Feb 2014)
16. Coretti, S., Dodis, Y., Tackmann, B., Venturi, D.: Non-malleable encryption: Simpler, shorter, stronger. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A, Part I. LNCS, vol. 9562, pp. 306–335. Springer, Heidelberg (Jan 2016)
17. Coretti, S., Maurer, U., Tackmann, B., Venturi, D.: From single-bit to multi-bit public-key encryption via non-malleable codes. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 532–560. Springer, Heidelberg (Mar 2015)
18. Dachman-Soled, D., Liu, F.H., Shi, E., Zhou, H.S.: Locally decodable and updatable non-malleable codes and their applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 427–450. Springer, Heidelberg (Mar 2015)
19. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM Journal on Computing 30(2), 391–437 (2000)
20. Drucker, A.: New limits to classical and quantum instance compression. SIAM Journal on Computing 44(5), 1443–1479 (2015)
21. 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 (Aug 2013)
22. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: Yao, A.C.C. (ed.) ICS 2010. pp. 434–452. Tsinghua University Press (Jan 2010)
23. Efremenko, K.: 3-query locally decodable codes of subexponential length. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 39–44. ACM Press (May / Jun 2009)
24. Erdős, P., Rado, R.: Intersection theorems for systems of sets. Journal of the London Mathematical Society 35(1), 85–90 (1960)
25. 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 (Feb 2014)
26. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: A tamper and leakage resilient von neumann architecture. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 579–603. Springer, Heidelberg (Mar / Apr 2015)
27. 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 (May 2014)

28. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (Feb 2004)
29. Goyal, V., Pandey, O., Richelson, S.: Textbook non-malleable commitments. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 1128–1141. ACM Press (Jun 2016)
30. Guo, A., Kopparty, S., Sudan, M.: New affine-invariant codes from lifting. In: Kleinberg, R.D. (ed.) ITCS 2013. pp. 529–540. ACM (Jan 2013)
31. Hemenway, B., Ostrovsky, R., Wootters, M.: Local correctability of expander codes. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 540–551. Springer, Heidelberg (Jul 2013)
32. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits II: Keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (May / Jun 2006)
33. Jafargholi, Z., Wichs, D.: Tamper detection and continuous non-malleable codes. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 451–480. Springer, Heidelberg (Mar 2015)
34. Kalai, Y.T., Kanukurthi, B., Sahai, A.: Cryptography with tamperable and leaky memory. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 373–390. Springer, Heidelberg (Aug 2011)
35. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: 32nd ACM STOC. pp. 80–86. ACM Press (May 2000)
36. Kopparty, S., Saraf, S., Yekhanin, S.: High-rate codes with sublinear-time decoding. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC. pp. 167–176. ACM Press (Jun 2011)
37. 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 (Aug 2012)
38. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. J. ACM 55(1), 1:1–1:16 (Feb 2008), `http://doi.acm.org/10.1145/1326554.1326555`
39. Yekhanin, S.: Locally Decodable Codes: A Brief Survey, pp. 273–282. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), `http://dx.doi.org/10.1007/978-3-642-20901-7_18`