Interactive Non-Malleable Codes Against Desynchronizing Attacks in the Multi-Party Setting

Nils Fleischhacker
Ruhr-Universität Bochum, Germany

Suparno Ghoshal
Ruhr-Universität Bochum, Germany

Mark Simkin
Ethereum Foundation, Aarhus, Denmark

Abstract

Interactive Non-Malleable Codes were introduced by Fleischhacker et al. (TCC 2019) in the two party setting with synchronous tampering. The idea of this type of non-malleable code is that it "encodes" an interactive protocol in such a way that, even if the messages are tampered with according to some class $\mathcal{F}$ of tampering functions, the result of the execution will either be correct, or completely unrelated to the inputs of the participating parties. In the synchronous setting the adversary is able to *modify* the messages being exchanged but cannot drop messages nor desynchronize the two parties by first running the protocol with the first party and then with the second party. In this work, we define interactive non-malleable codes in the non-synchronous multi-party setting and construct such interactive non-malleable codes for the class $\mathcal{F}^{s}_{\mathsf{bounded}}$ of bounded-state tampering functions.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols; Mathematics of computing → Coding theory

Keywords and phrases non-malleability, multi-party protocols

Digital Object Identifier 10.4230/LIPIcs.ITC.2023.5

Funding Nils Fleischhacker and Suparno Ghoshal were supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972.

1 Introduction

There is a long line of research that aims to make communication resilient to tampering, starting with error correcting codes. Error correcting codes allow a sender to encode a message m into a codeword c, such that a receiver can always recover the message m even from a tampered codeword c' as long as the tampering is done in some restricted way. Specifically, the class of tampering functions tolerated by traditional error correcting codes are those that erase or modify at most a constant fraction of the symbols in codeword c. If the tampering function, however, behaves in any other way, there is no longer any guarantee on the output of the decoding algorithm. Error *detecting* codes are a relaxation that allows the decoder to also output a special symbol $\perp$ when m is not recoverable from c'. But these codes, again, cannot tolerate, i.e. will decode incorrectly when tampered with, many simple tampering functions such as a constant function.

Dziembowski, Pietrzak, and Wichs [30] introduced a further relaxation which they called non-malleable codes (NMC). Very informally, an encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ is an NMC for a class of tampering functions, $\mathcal{F}$, if the following holds: given a tampered codeword $c' = f(\mathsf{Enc}(m))$ for some $f \in \mathcal{F}$, the decoded message $m' = \mathsf{Dec}(c')$ is either the original message m or *completely unrelated* to m. I.e., the tampering function can only "destroy" the information being transferred, but not modify it in a meaningful way. Obviously, NMCs can still not exist for the set of *all* tampering functions $\mathcal{F}_{\mathsf{all}}$. To see this, consider the tampering function that retrieves $m = \mathsf{Dec}(c)$, chooses a message m' related to m and encodes $c' = \mathsf{Enc}(m')$. This tampering trivially defeats the requirement above. In light of this observation, a rich line of works has dealt with constructing non-malleable codes for different classes of tampering attacks (see Section 1.3 for a discussion).

Non-malleable codes have the obvious advantage that we can obtain meaningful guarantees for larger classes of tampering functions (compared to error correcting codes) and they have also found a number of interesting applications in cryptography such as tamper-resilient cryptography [30, 46, 33, 34]. They have also been useful as a building block in constructing non-malleable encryption [23], round optimal non-malleable commitments [39], and non-malleable secret sharing schemes [37, 38, 7].

Interactive Coding

Traditional codes, whether error correcting, error detecting or non-malleable are only concerned with the scenario where a sender sends a single message to a receiver. Interactive Coding, introduced by Schulman [49, 50, 51], generalizes (error correcting) codes to arbitrary interactive protocols between two or more [48] parties. Consider the following scenario: n parties, each with their own input, are running an interactive protocol to perform some task involving their inputs, such as computing a joint function on them. Now, say an adversary can get access to their communication channels and tamper with the messages being sent in the protocol. An interactive code for a class of tampering functions $\mathcal{F}$ is essentially a wrapper around the protocol that would guarantee that, as long as the tampering is performed using a function $f \in \mathcal{F}$, the protocol will conclude correctly and all participants will be able to recover their correct output.

Interactive Non-Malleable Codes

In interactive coding, just as in the case of error correcting codes, there are strong limits on which classes of tampering can be dealt with. To achieve meaningful guarantees for larger classes of tampering functions Fleischhacker et al. [36] introduced the notion of interactive *non-malleable* codes (INMC). Just as interactive coding generalizes error correcting codes, INMCs generalize NMCs, by encoding "active communication" instead of "passive data". An INMC is supposed to give a similar guarantee as an NMC. Informally, that means that the participants of the protocol should either be able to recover the correct output from the protocol or the correct output would be completely "destroyed" and the participants would recover something completely unrelated to their inputs. Fleischhacker et al. in fact define two seperate notion of non-malleability, weak non-malleability only requires the outputs to be unrelated from *other* parties' inputs, while *strong* non-malleability requires the outputs to be unrelated even to the party's own input. We are only interested in the *strong* non-malleability notion, which we will simply call non-malleability. It turns out that strong non-malleability somewhat counterintuitively actually implies error detection in the interactive case.

Fleischhacker et al. [36] define three classes of tampering functions, bounded-state tampering, a variation of split-state tampering, and sliding-window tampering. For each class they give a construction of a strongly non-malleable INMC.

However, both the definitions and the constructions are limited, because they only apply to the two party setting and they only consider *synchronous* tampering. Consider a protocol between two parties, Alice and Bob. In the synchronous setting, when Alice sends a message

m to Bob, the tampering function can arbitrarily *modify* m, but it *must* then forward it to Bob without further delay. I.e., at all times, even in a tampered execution of the protocol, Alice and Bob remain in sync and the tampering function can *not* choose to, e.g., first finish the protocol with Alice, before later resuming the communication with Bob.

When considering desynchronization between parties in an interactive protocol we need to consider how protocols are modeled. As pointed out by Braverman et al. [13] there are essentially two paradigms for protocols without fully synchronized parties. In a *clock-driven* model, each party has a clock and wakes up with each clock tick, checks for incoming messages, performs some computation, and potentially send messages to the other parties. Here desynchronization can occur because the different parties' clocks may be mismatched or skewed. In a *message-driven* model, the parties sleep until they receive a new message, which triggers them to wake up and perform some action. Here desynchronization can occur because messages are dropped causing one party to be ahead in the protocol. The model for *desynchronization* considered in this paper is of the latter kind. I.e., parties in our model are purely activated by receiving messages and have no sense of time, i.e., they do not notice how long they may have been asleep. This strengthens the possible attacker, since it allows for even the most extreme forms of desynchronization.

1.1 Results and Technical Overview

In this work, we aim to remedy the shortcomings of the previous work by Fleischhacker et al. [36]. In Section 3 we introduce new definitions for arbitrary (potentially desynchronizing) tampering with interactive protocols between $n \geq 2$ parties, define interactive non-malleability and formalize the class of bounded-state tampering functions. In Section 5 we construct an INMC for bounded-state tampering functions.

The "Obvious" Solution

When faced with the task of constructing an interactive non-malleable code it may seem tempting to directly apply the huge body of work around regular non-malleable codes and try to build an INMC by simply applying an NMC on a per-message basis. While we might not get guarantees against the *same* class of tampering functions, we might still hope to get *some* useful guarantees. Sadly, this is not the case for general protocols. Consider a protocol between Alice and Bob, where Alice has input (x_1, x_2) and Bob has no input. In the protocol, Alice first sends x_1 to Bob, Bob replies with some arbitrary message and then Alice sends x_2. At the end Alice outputs nothing and Bob outputs (x_1, x_2).

If we encode the messages in this protocol individually, a tampering function can simply leave all the messages related to x_1 intact and replace the messages related to x_2 with constant messages that will decode to some x_2', causing Bob to output (x_1, x_2'), an output very much *not* unrelated (x_1, x_2). This attack works for any class of tampering functions that allow to tamper with the entire message, no matter how restrictive.

Technical Overview

On a technical level, our construction is heavily inspired by the bounded-state INMC of [36] and follows the same basic idea: At the beginning of the protocol, each pair of parties runs a key-exchange protocol that is secure against a bounded state attacker with full control of the communication channel. Once the key material has been successfully exchanged, the parties will engage in the underlying protocol while encrypting all messages with an information theoretically secure encryption scheme and authenticating each message with an information theoretically secure message authentication code.

However, since [36] was restricted to two parties and worked in a strictly synchronized setting, it is unsurprising that directly lifting their protocol to the multi-party unsynchronized setting causes it to fail in several ways. The key-exchange of [36] works as follows: The two parties P_1, P_2 each choose random strings α_1, β_1 and α_2, β_2 of sufficient length. They then send these in alternating order, use a 2-non-malleable extractor to agree on a key $k := \mathsf{nmExt}(\alpha_1, \beta_1) \oplus \mathsf{nmExt}(\alpha_2, \beta_2)$ and go on to exchange key-confirmation messages to confirm that both parties have received the same key. Once we allow the tampering function to desynchronize the two parties this key exchange becomes trivially broken. Consider the tampering function f that simply ignores P_2 and their messages. Instead, when P_1 sends α_1, the tampering function immediately sends back α_1 and likewise for β_1. Note that this attack works, even though P_1 will now receive α_2 in the "wrong round". This is because, as discussed above, we work in a purely message-driven model where P_1 does not notice how much time has passed between sending α_1 and receiving α_2. This means that P_1 will now derive the constant key $\mathsf{nmExt}(\alpha_1, \beta_1) \oplus \mathsf{nmExt}(\alpha_1, \beta_1) = 0^\kappa$. This key is of course trivially known to the tampering function in future rounds meaning the tampering function can simply engage in the underlying protocol pretending to be P_2 with some arbitrary input y' of its own choice. If the protocol is meant to evaluate a function g on the joint inputs, P_1 will now output $g(x, y')$ which is in general neither the correct result nor independent of (x, y).

To fix this problem, we split each bidirectional communication channel into two unidirectional channels and negotiate separate keys. The two parties still choose random strings α_1, β_1 and α_2, β_2 of sufficient length and send them in separate messages. However the parties agree on two separate keys $k_1 := \mathsf{nmExt}(\alpha_1, \beta_1)$ and $k_2 := \mathsf{nmExt}(\alpha_2, \beta_2)$. Each party P_i then uses k_i to *encrypt* messages *sent to* the other party and to *verify* the authentication tags on messages *received from* the other party. This way, each party always uses a key that is known to be *untampered* to perform the security critical operations.

It is still critical, that keys are confirmed and bound to a specific channel. If keys are not explicitly confirmed, a tampering function could replace one of the keys, say k_2 without any of the parties realizing. If P_2 would now send a message to P_1, this message would be *authenticated* using k_1 which was not tampered with, meaning P_1 would accept it. However they would then go on to *decrypt* the message with an incorrect key k_2'. This would likely result in P_1 passing a random string to the underlying protocol and there is no guarantee how the underlying protocol would behave in that case. If the key was not explicitly bound to a specific channel, a tampering function could potentially "swap" two parties. Say there's a protocol where P_3 and P_2 do not communicate with one another but *do* communicate with P_1. The tampering function could swap all messages from the channel between P_1 and P_2 to the channel between P_1 and P_3 and vice versa. If P_2 and and P_3 behave identically in the protocol and never explicitly identify themselves, this would lead P_1 to output $g(x_1, x_3, x_2)$ which again is obviously neither correct nor independent of the original input (x_1, x_2, x_3) in general.

To prevent all these and other problems introduced by the existence of multiple parties and the ability of the tampering function to desynchronize the parties each message is always authenticated together with the identifier of the channel it is being sent on and the message counter.

Different Message Topologies

The INMC for bounded-state tampering functions presented in Section 5 is still somewhat restricted in the sense that it can only directly be applied to protocols with a fixed message topology. This means the message flow of the protocol is required to be known is a priori and has to be independent of inputs and randomness. I.e., when party P_i is invoked for the rth time, we a priori know from which parties they *should be receiving* messages and to which parties they *should be sending.*

Restricting ourselves to protocols with a fixed message topology makes our live significantly easier, as it allows us to sidestep many subtle issues. The most obvious problem would be an input dependent message topology. If, whether P_i sends a message to P_j when invoked for the rth time depends on the value x_i, we can easily come up with ways to leak x_i to the tampering function, which would make non-malleability impossible. A more subtle issue are protocols that *misbehave* if messages are reordered or dropped. Consider a protocol between two parties. In an untampered execution P_1 would receive a message from P_2 in its first invocation. At the end both parties output 0. Now we can modify this protocol to *misbehave* if the messages from P_2 never arrives. In this case P_1 could simply output x_1, which is neither correct nor unrelated to (x_1, x_2). If we, however, *know* which messages should be arriving in which order, we can abort any party that did not receive messages as specified in the protocol preventing them from misbehaving.

Obviously, restricting the INMC to protocols with a fixed message topology limits its applicability at least in theory. However, we show in Section 4 that any protocol (with a fixed upper bound on the number of rounds) can be transformed into a protocol with a fixed message topology, thereby extending the applicability of the INMC to (almost) arbitrary protocols. The transformation is fairly straightforward and simply involves sending dummy messages when the original protocol decides *not* to send a message. The transformation naturally comes with a certain blowup in the communication complexity.

1.2 Instantiating the Construction

To instantiate the protocol the main question is how to instantiate the 2-source non-malleable extractor. Before going into details of the instantiation of the non-malleable extractor one needs to understand the amount of key material that will be required by each party in order to carry out the protocol execution efficiently. As per the construction of our protocol every party will use the 2-source non-malleable extractor to extract an authentication key and an encryption key per party it communicates with. The length of those keys will depend on the number of messages the party expects to exchange with each other party. For a rough ballpark estimate, let us assume that the encoded protocol is between n parties, that each party sends the same number r of messages to every other party, and that those messages are all of length $\ell \geq \lambda$.

From the two sources sent from party A to party B, both parties must thus extract a key for a statistically unforgeable $(r+1)$-time MAC and a key for stateful r-time encryption scheme with perfect indistinguishability. Following, Remark 2 and Remark 4 they have to extract at least $(2r+2) \cdot \ell$ bits from each pair of sources. For a 2-source non-malleable extractor with source length κ_{nm} we will later see, that the sources will have min-entropy at least $\kappa_{nm} - (s + 3n\lambda)$. To get the lowest possible overhead, we will need a 2-source non-malleable extractor that can tolerate sources with the lowest possible min-entropy. The best currently known extractor in this regard was described in a recent paper by Xin Li [44]. The description of the construction, as is common for the literature on extractors, unfortunately only makes *asymptotic* statements about the extractor. It is thus hard to find out what the *exact* concrete source length of the extractor needs to be. We can however make an estimate on the *best possible* overhead achievable with the extractor from [44].

The construction described in Theorem 6.3 of [44] requires sources with min-entropy $(2/3 + \gamma) \cdot \kappa_{nm}$ for the first source and k with $k \geq C \log \kappa_{nm}$ for the second source, where $0 < \gamma < 1/3$ can be chosen arbitrarily and $C > 1$ is some "large enough" constant. For large enough κ_{nm} we can choose $k = (2/3 + \gamma) \cdot \kappa_{nm}$ which is convenient as the guaranteed min-entropy of the sources will be balanced in our application. The absolute best-case scenario for Li's extractor, depending on its exact instantiation, is that the output length is $9 \cdot 10^{-6} \cdot \kappa_{nm}$ and therefore we must have that $\kappa_{nm} > (10^6/9) \cdot (2r+2)\ell$. However, this is not the only condition. Additionally, we need to consider that we must have $\kappa_{nm} - (s+3n\lambda) \geq (2/3+\gamma) \cdot \kappa_{nm}$ for the non-malleability guarantee to apply. Therefore, we must also have $\kappa_{nm} \geq (s+3n\lambda)/(1/3+\gamma)$. Which of these two bounds is larger depends on the the exact parameters of the protocol and the size of the tampering function's state. However clearly even in the best case scenario the current state of the art makes the encoded protocol incur a multiplicative overhead of roughly $440,000$. It is thus clear that, which the current state of the art, our construction is chiefly of theoretical interest.

1.3 Related Works

To the best of our knowledge, the only previous work on non-malleable codes in the interactive setting has been the already mentioned work of Fleischhacker et al. [36]. In concurrent work Lin [45] used the results of [36] to construct non-malleable *multi-party computation*. However, Lin's results are largely orthogonal to our work. In particular the tampering model is weaker. E.g., while we allow the tampering function to tamper jointly on all parties' concurrent messages, Lin requires a fixed execution order and only allows tampering based on *past* messages. At the same time Lin attempts to achieve not merely a non-malleable encoding but non-malleable MPC, where the same party who controls the tampering function *also* controls a number of corrupted parties. Overall this means that the results of [45] are incomparable even if the used techniques are similar.

In contrast, non-malleable codes in the non-interactive setting have been studied extensively for a large variety of different classes of tampering functions. The most extensively studied class in the non-interactive setting are certainly split-state tampering functions [46, 29, 3, 19, 18, 2, 20, 42, 40, 41, 4]. But other classes of tampering functions have been studied such as tampering circuits of limited size or depth [35, 10, 17, 11, 8], tampering functions computable by decision trees [12], memory-bounded tampering functions [32] where the size of the available memory is a priori bounded, bounded polynomial time tampering functions [9], bounded *parallel*-time tampering functions [26], and non-malleable codes against streaming tampering functions [11]. Non-malleable codes were also generalized in several ways, such as continuously non-malleable codes in [33, 25, 23, 47, 31, 24, 4] and locally decodable and updatable non-malleable codes [28, 15, 27].

As a general rule non-malleable codes are usually considered in the information theoretic setting. However, there has also been some work in the computational setting. [1, 5, 6, 11]

2 Preliminaries

In this section we introduce our notation and recall some definitions needed for our constructions and proofs.

2.1 Notation

We denote the security parameter by $\lambda \in \mathbb{N}$. For an integer $n \in \mathbb{N}$, denote $[n] = \{1, \ldots, n\}$.

Let $\boldsymbol{M}$ be a matrix. We denote by $\mathbf{row}_i(\boldsymbol{M})$ the i-th row vector and by $\mathbf{col}_j(\boldsymbol{M})$ the j-th column vector of $\boldsymbol{M}$. If $\boldsymbol{M}$ is square, we denote by $\mathbf{diag}(\boldsymbol{M})$ the vector representing the main diagonal of $\boldsymbol{M}$.

Let S and S' be sets, let $P : S \to \{\mathsf{true}, \mathsf{false}\}$ be a predicate, let $f : S \to S'$ be a function, and let $L = (x_1, \dots, x_\ell) \in S^\ell$ be a list. We denote by $\big(f(x) \mid x \in L \wedge P(x)\big)$ the list that contains $f(x_i)$ iff $P(x_i) = \mathtt{true}$ and preserves the relative order of the elements.

For $x' \in S$ we denote by $L \circ x'$ the list $(x_1, \dots, x_\ell, x')$, i.e. the list resulting from appending x' to L. Further, we write L_i to denote the ith entry of L and $L_{\leq i}$ to denote the length i prefix of L, i.e. $L_{\leq i} = (x_1, \dots, x_i)$.

Let D be some distribution over S. We denote by $f(D)$ the distribution over S' sampled by first sampling x according to D and then applying f to x. For a pair D_1, D_2 of distributions over a domain S, we denote their statistical distance by

$$\mathsf{SD}(D_1, D_2) = \frac{1}{2} \sum_{v \in S} \Big| \Pr[x \leftarrow D_1 : x = v] - \Pr[x \leftarrow D_2 : x = v] \Big|.$$

If $\mathsf{SD}(D_1, D_2) \leq \epsilon$, we say that D_1, D_2 are ϵ-close. For an arbitrary set S we define the functions $\mathsf{replace} : (S \cup \{\mathsf{same}\}) \times S \to S$ and $\mathsf{indicate} : S \to \{\mathsf{same}, \bot\}$ as

$$\mathsf{replace}(x, y) := \begin{cases} y & \text{if } x = \mathsf{same} \\ x & \text{otherwise} \end{cases} \quad \text{and} \quad \mathsf{indicate}(x) := \begin{cases} \mathsf{same} & \text{if } x \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

We extend $\mathsf{replace}$ and $\mathsf{indicate}$ to n-tuples in the natural way by applying them component-wise, i.e. $\mathsf{replace}(\boldsymbol{x}, \boldsymbol{y}) := (\mathsf{replace}(x_1, y_1), \dots, \mathsf{replace}(x_n, y_n))$ and $\mathsf{indicate}(\boldsymbol{x}) := (\mathsf{indicate}(x_1), \dots, \mathsf{indicate}(x_n))$.

2.2 Encryption and Message Authentication Codes

Our constructions relies exclusively on information theoretically secure primitives, specifically perfectly indistinguishable encryption and statistically secure message authentication codes. For notational convenience we formalize encryption as stateful which allows us not burden the description of the protocol with keeping track of key-usage.

▶ **Definition 1** (Stateful q-time Encryption with Perfect Indistinguishability)**.** *A correct stateful q-time encryption scheme $\mathcal{E}$ for message space $\{0,1\}^\ell$ and keyspace $\{0,1\}^\kappa$ consists of a pair of deterministic stateful algorithms* $(\mathsf{Enc}, \mathsf{Dec})$*, such that for all keys $k \in \{0,1\}^\kappa$ and all messages $(m_1, \dots, m_q) \in (\{0,1\}^\ell)^r$ we have that for $c_1 := \mathsf{Enc}(k, m_1), \dots, c_q := \mathsf{Enc}(k, m_q)$ and $m'_1 := \mathsf{Dec}(k, c_1), \dots, m'_q := \mathsf{Dec}(k, c_q)$ it holds that $m_i = m'_i$ for all $i \in [r]$.*

Let LoR *be the stateful "left-or-right" algorithm defined as* $\mathsf{LoR}(k, b, m_0, m_1) := \mathsf{Enc}(k, m_b)$ *for the first q invocations and as $\bot$ afterwards. A stateful q-time encryption scheme is perfectly indistinguishable if for any unbounded algorithm $\mathcal{A}$ it holds that*

$$\Pr\big[k \leftarrow \{0,1\}^\kappa : \mathcal{A}^{\mathsf{LoR}(k,0,\cdot,\cdot)} = 0\big] = \Pr\big[k \leftarrow \{0,1\}^\kappa : \mathcal{A}^{\mathsf{LoR}(k,1,\cdot,\cdot)} = 0\big]$$

For convenience we extend the notation of encryption schemes to vectors in the natural way by applying the algorithm component wise. I.e., for $\boldsymbol{m} \in (\{0,1\}^\ell)^n$ and $k \in (\{0,1\}^\kappa)^n$ we write $\boldsymbol{c} := \mathsf{Enc}(\boldsymbol{k}, \boldsymbol{m})$ to denote the vector consisting of $c_i := \mathsf{Enc}(k_i, m_i)$. Similarly we write $\boldsymbol{m'} := \mathsf{Dec}(\boldsymbol{k}, \boldsymbol{c})$ for the vector consisting of $m'_i := \mathsf{Dec}(k_i, c_i)$.

▶ Remark 2. A stateful q-time encryption with perfect indistinguishability can easily be instantiated using the one-time pad where the key k is split into keys $k_1, \dots, k_q \in \{0,1\}^\ell$ and c_i is computed as $m_i \oplus k_i$. The perfect indistinguishability follows from the regular perfect secrecy of the one-time pad.[52] In this case $\kappa = q\ell$.

▶ **Definition 3** (Statistically Unforgeable q-time MACs)**.** *A q-time message authentication code $\mathcal{M}$ for message space $\{0,1\}^\ell$ and keyspace $\{0,1\}^\kappa$ consists of a pair of deterministic algorithms* $(\mathsf{MAC}, \mathsf{Vf})$*, such that for all keys $k \in \{0,1\}^\kappa$ and all messages $m \in \{0,1\}^\ell$ it holds that* $\mathsf{Vf}(k, m, \mathsf{MAC}(k, m)) = 1$.

Let $n \in \mathbb{N}$ and let $\widetilde{\mathsf{MAC}}$ be the algorithm defined as $\widetilde{\mathsf{MAC}}(k_1, \dots, k_n, i, m) := \mathsf{MAC}(k_i, m)$. A q-time message authentication code is ϵ-unforgeable, if for all unbounded algorithms $\mathcal{A}$ it holds that

$$\Pr\left[\begin{matrix} k_1, \dots, k_n \leftarrow \{0,1\}^\kappa \\ (i, m, t) \leftarrow \mathcal{A}^{\widetilde{\mathsf{MAC}}(k_1,\dots,k_n,\cdot,\cdot)}() \end{matrix} : \begin{matrix} \mathsf{Vf}(k_i, m, t) = 1 \\ \wedge\, (m, t) \notin Q_i \wedge |Q_i| \leq q \end{matrix}\right] \leq \epsilon$$

where Q_i denotes the set of message-answer pairs, queried by $\mathcal{A}$ for index i.

Similar to encryption schemes, we extend the notation of message authentication codes to vectors in the natural way by applying the algorithm component wise. I.e., for $\boldsymbol{m} \in (\{0,1\}^\ell)^n$ and $\boldsymbol{k} \in (\{0,1\}^\kappa)^n$ we write $\boldsymbol{t} := \mathsf{MAC}(\boldsymbol{k}, \boldsymbol{m})$ to denote the vector consisting of $t_i := \mathsf{MAC}(k_i, m_i)$.

▶ Remark 4. Statistically unforgeable q-time MACs can be instantiated using any family of $q+1$-wise independent functions such as the family of degree q polynomials over $\mathbb{F}_{2^{\max\{\ell,\lambda\}}}$ [53]. In this case $\kappa = (q+1) \cdot \max\{\ell, \lambda\}$ and $\epsilon = 2^{-\max\{\ell,\lambda\}}$.

2.3 2-Non-Malleable Extractors

Our construction also makes use of 2-non-malleable extractors. These were first defined by Cheraghchi and Guruswami [19, 21] but constructing them was left as an open problem. The definition was finally instantiated by Chattopadhyay, Goyal, and Li [16]. Such an extractor allows to non-malleably extract an almost uniform random string from two sources with a given min-entropy that are being tampered by a split-state tampering function. We closely follow the definition from [16].

▶ **Definition 5** (2-Non-Malleable Extractor)**.** *A function* $\mathsf{nmExt} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ *is a 2-non-malleable extractor for sources with min-entropy $k < n$ and with error ϵ if it satisfies the following property: If X, Y are independent sources of length n with min-entropy k and $f = (f_0, f_1)$ is an arbitrary 2-split-state tampering function, then there exists a distribution D_f over* $\{0,1\}^m \cup \{\mathsf{same}\}$*, such that*

$$\mathsf{SD}\big((\mathsf{nmExt}(X,Y), \mathsf{nmExt}(f_0(X), f_1(Y))), (U_m, \mathsf{replace}(D_f, U_m))\big) \leq \epsilon$$

where both U_m refer to the same uniform m-bit string.

▶ Remark 6. The required 2-non-malleable extractor can be instantiated with the construction of Chattopadhyay Goyal and Li [16] or a number of other construction. [42, 43, 22].

3 Interactive Protocols and Tampering Model

We consider protocols Π between n parties $P_1, \dots, P_n$ for evaluating functionalities $g = (g_1, \dots, g_n)$ of the form $g_i : X_1 \times \cdots \times X_n \to Y_i$, where X_i, Y_i are finite domains. Each party P_i holds an input $x_i \in X_i$ and randomness $\omega_i \in \Omega_i$ and the goal of the protocol is to interactively evaluate the functionality, such that at the end of the protocol party P_i outputs $g_i(x_1, \dots, x_n) \in Y_i$.

Formally, an interactive protocol Π between n parties can be described either using interactive Turing machines, or using next-message functions. The two formalizations are equivalent up to a slight computational overhead. We will switch between the two formalizations whenever this is convenient for exposition.

Interactive Protocols as Interactive Turing Machines

In this formalization an interactive protocol Π between n parties is described by an n-tuple of interactive Turing machines P_i. Each interactive Turing machine P_i has an input tape containing x_i, a random tape containing ω_i, an internal work tape, as well as an incoming communication tape and an outgoing communication tape for each party P_j with $j \neq i$ and an output tape.

Interactive Protocols as Next Message Functions

In this formalization an interactive protocol Π between n parties is described by a an n-tuple of "next message" functions π_i and an n-tuple of output functions out_i. The next message function π_i takes as input the view of P_i, i.e., the input x_i, the randomness ω_i, and the sequence of message vectors received by P_i thus far and outputs the vector $\boldsymbol{s}_i \in \{0,1\}^* \cup \{\perp\}$ of messages to be sent by P_i. The output function out_i takes as input the final view of P_i, i.e., x_i, ω_i, and received message vectors and outputs P_i's protocol output.

Equivalence of Formalizations

The two formalizations are equivalent up to a slight computational overhead. To see this consider the following two simple conversions: Given an interactive Turing machine P_i, the equivalent next message function π_i can be computed on input $x_i, \omega_i, \boldsymbol{m}_i$ by simulating the Turing machine on input x_i and randomness ω_i, writing the received messages for each round on the appropriate incoming communication tapes until the current round is reached. The content of the outgoing communication tapes can then be output as $\boldsymbol{s}_i$. Similarly, given a next message function π_i, the equivalent interactive Turing machine P_i will simply store the contents of its incoming communication tapes on its internal work tape, evaluate π_i on its input x_i, randomness ω_i and all incoming messages, and write the output of π_i to its outgoing communication tapes.

3.1 Correctness and Encodings

We denote by $\Pi(\boldsymbol{x})$ the joint distribution of the outputs of an honest execution of the protocol Π using inputs $\boldsymbol{x}$ and uniformly sampled randomness $\boldsymbol{\omega}$. Further, we denote by $g(\boldsymbol{x})$ the vector $(g_1(x_1, \ldots, x_n), \ldots, g_1(x_1, \ldots, x_n))$.

▶ **Definition 7** (Correctness). *A protocol Π, is said to ϵ-correctly evaluate a functionality $g = (g_1, \ldots, g_n)$ if an untampered execution of the protocol correctly computes g with probability at least $1 - \epsilon$. I.e., for all valid input vectors $\boldsymbol{x}$ it holds that $\Pr[\boldsymbol{y} \leftarrow \Pi(\boldsymbol{x}) : \boldsymbol{y} = g(\boldsymbol{x})] \geq 1 - \epsilon$, where the probability is taken over the uniform choice of the random tape of all parties.*

▶ **Definition 8** (Encoding of an Interactive Protocol). *An encoding $\mathcal{E}$ of n-party interactive protocols is defined by n interactive oracle machines Enc_i.*

Let Π be an arbitrary interactive n-party protocol that ϵ-correctly evaluate a functionality g. The encoded *protocol is then the interactive n-party protocol between interactive Turing machines $(Q_1, \ldots, Q_n)$ defined as follows: On input x_i, Q_i samples uniform randomness ω_i,* *initiates the oracle* $\mathsf{O}_{\boldsymbol{x},\boldsymbol{\omega}'} = P_i(x_i;\omega_i)$ *and then executes* $\mathsf{Enc}_i^{\mathsf{O}_{\boldsymbol{x},\boldsymbol{\omega}'}}()$*, giving it direct access to all communication tapes. Once* $\mathsf{Enc}_i^{\mathsf{O}_{\boldsymbol{x},\boldsymbol{\omega}'}}()$ *terminates with some output* y*,* Q_i *also outputs* y*.* $\mathcal{E}$ *is a* δ*-correct protocol encoding for* Π *if for all inputs* $\boldsymbol{x}$*, the protocol* $\mathcal{E}(\Pi) = (Q_1, \dots, Q_n)$ $\epsilon + \delta$*-correctly evaluates the functionality* g*.*

3.2 Tampering Model

The transcript of a protocol executed under tampering needs to specify for each round of execution both the messages sent by each party and the messages received by each party. Remember that, due to the presence of the tampering function, the messages received are not necessarily related in any way to the messages sent.

We consider a scenario in which each party has a point-to-point channel to each other party, but not to itself. I.e., a protocol among n parties is executed over a complete directed communication graph (excluding loops) with n nodes P_i.

For each round, the transcript needs to label each edge (P_i, P_j) for $i \neq j$ in the graph with the message P_i sent to P_j and the message P_j received from P_i, the two of which need not be related. We will denote this with two $n \times n$ matrices $\boldsymbol{S}$ and $\boldsymbol{R}$ of labels per round of execution, where a label is either an arbitrary bitstring or the special symbol $\perp$ denoting that *no* message was sent or received respectively.

▶ **Definition 9** (Transcripts). *Let* $\mathcal{M} = \{0,1\}^* \cup \{\perp\}$ *be the set of possible labels for the edges of the communication graph. The set of possible transcripts is then the set of lists of pairs of matrices* $\boldsymbol{S}_i, \boldsymbol{R}_i \in \mathcal{M}^{n\times n}$ *such that the diagonal of both matrices only contains* $\perp$*. I.e.,*

$$\mathcal{T} = \left(\left\{\boldsymbol{M} \in \mathcal{M}^{n\times n} \;\middle|\; \mathbf{diag}(\boldsymbol{M}) \in \{\perp\}^n\right\}^2\right)^*.$$

For any transcript $\tau = \big((\boldsymbol{S_1}, \boldsymbol{R_1}), \dots, (\boldsymbol{S_\ell}, \boldsymbol{R_\ell})\big)$*,* $\mathbf{row}_j(\boldsymbol{S_i})$ *denotes the vector of messages sent by* P_j *in round* i *of the execution, while* $\mathbf{col}_j(\boldsymbol{R_i})^\top$ *denotes the vector of messages received by* P_j *in round* i *of the execution.*
We denote by $\mathsf{Trans}_\Pi(\boldsymbol{x}, \boldsymbol{\omega})$ *the function mapping the input vector* $\boldsymbol{x}$ *along with the randomness* $\boldsymbol{\omega}$ *to the transcript of an honest execution of* Π *with inputs* $\boldsymbol{x}$ *and randomness* $\boldsymbol{\omega}$*.*

A party's view of the transcript consists exactly of the vectors of messages it receives. In particular, if a party does not receive any messages in a particular round of the execution, this round is not included in the party's view. This models that a party is not necessarily capable of detecting that desynchronization happens and allows general tampering functions to arbitrarily desynchronize different parties during protocol execution.

▶ **Definition 10** (Views). *Let* τ *be a transcript. The corresponding view of party* P_i *is then defined as* $V_i(\tau) = \big(\mathbf{col}_i(\boldsymbol{R})^\top \;\big|\; (\boldsymbol{S}, \boldsymbol{R}) \in \tau \wedge \mathbf{col}_i(\boldsymbol{R})^\top \notin \{\perp\}^n\big)$.

The interactive non-malleable code presented in Section 5 is restricted to protocols with a fixed message topology. This means that the number of messages exchanged over each channel is fixed, the expected relative ordering of all the messages received by a single party is a priori known, and whether or not a party sends a message along a communication channel does not depend on their input or their received messages. I.e., the "structure" of each vector in a party's view as well as the output vector in any particular round of execution is fixed in an untampered execution. We define this formally as follows.

▶ **Definition 11** (Fixed Message Topology). *An interactive protocol Π with n parties defined by next message functions π_i and output functions out_i is said to have a fixed message topology, if there exists a function $\mu : [n] \times \mathbb{N} \to \{0,1\}^n \times \{0,1\}^n$, such that for all vectors of inputs $\boldsymbol{x}$, all randomness vector $\boldsymbol{\omega}$ and the transcript τ of an honest untampered execution of Π on $\boldsymbol{x}$ with $\boldsymbol{\omega}$, all $i \in [n]$, and all $r \in [|V_i(\tau)|]$ it holds that $\mu(i,r) = (\boldsymbol{v}', \boldsymbol{s}')$, where*

$$v'_j := \begin{cases} 0 & \text{if } V_i(\tau)_{r,j} = \bot \\ 1 & \text{otherwise} \end{cases} \qquad s'_j := \begin{cases} 0 & \text{if } \pi_i(x_i, \omega_i, V_i(\tau)_{\leq r})_j = \bot \\ 1 & \text{otherwise} \end{cases}$$

for $j \in [n]$ and for all $r \geq |V_i(\tau)|$ it holds that $\mu(i,r) = (0^n, 0^n)$. We further define the function $\nu : [n] \times [n] \to \mathbb{N}$ as $\nu(i,j) := \sum_{r\in\mathbb{N}} \mu(i,r)_{1,j} = \sum_{r\in\mathbb{N}} \mu(j,r)_{2,i}$ as the exact number of messages received by party i from j during an execution of the protocol.

Let Π be a protocol with n parties defined by next message functions π_i and output functions out_i. For ease of notation we define the function Next_Π which describes computation of all messages sent during a particular round of execution depending on the protocol specification, the vector of inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ and the partial transcript $\tau \in \mathcal{T}$. Let $F : \mathcal{T} \times \mathcal{M}^{n\times n} \to \mathcal{M}^{n\times n}$ be an arbitrary tampering function. We describe execution of Π on inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ under tampering by F using the algorithm $\mathsf{Execute}_{\Pi,F}$.

```
$\mathsf{Next}_\Pi(\boldsymbol{x}, \boldsymbol{\omega}, \tau)$
parse $\tau = ((\boldsymbol{S_1}, \boldsymbol{R_1}), \ldots, (\boldsymbol{S_\ell}, \boldsymbol{R_\ell}))$
for $1 \leq i \leq n$ do
  if $\tau = \emptyset \lor \mathsf{col}_i(\boldsymbol{R_\ell})^\top \neq \bot^n$
    $\boldsymbol{s_i} := \pi_i(x_i, \omega_i, V_i(\tau))$
  else
    $\boldsymbol{s_i} := \bot^n$
return $\begin{bmatrix} \boldsymbol{s_1} \\ \vdots \\ \boldsymbol{s_n} \end{bmatrix}$

$\mathsf{Execute}_{\Pi,F}(\boldsymbol{x}; \boldsymbol{\omega})$
$\tau := \emptyset$
$\boldsymbol{S} := \mathsf{Next}(\Pi, \boldsymbol{x}, \boldsymbol{\omega}, \tau)$
$\boldsymbol{R} := F(\tau, \boldsymbol{S})$
while $\boldsymbol{R} \neq \bot^{n\times n}$
  $\tau := \tau \circ (\boldsymbol{S}, \boldsymbol{R})$
  $\boldsymbol{S} := \mathsf{Next}(\Pi, \boldsymbol{x}, \boldsymbol{\omega}, \tau)$
  $\boldsymbol{R} := F(\tau, \boldsymbol{S})$
return $\big(\mathsf{out}_1(x_1, V_1(\tau)), \ldots, \mathsf{out}_n(x_n, V_n(\tau))\big)$
```

Let $I : \mathcal{T} \times \mathcal{M}^{n\times n} \to \mathcal{M}^{n\times n}$ be the function defined as $I(\tau, \boldsymbol{S}) := \boldsymbol{S}$. We call I the identity tampering function. Note that the distribution $\Pi(\boldsymbol{x})$ is identical to the distribution $\mathsf{Execute}_{\Pi,I}(\boldsymbol{x})$.

▶ **Definition 12** (Protocol Non-malleability). *An n-party protocol Π for functionality g is ϵ-protocol non-malleable for a family $\mathcal{F}$ of tampering functions if for every tampering function $F \in \mathcal{F}$ there exists a distribution D_F over $\{\bot, \mathsf{same}\}^n$ such that for all $\boldsymbol{x}$, it holds that*

$$\mathsf{SD}(\mathsf{Execute}_{\Pi,F}(\boldsymbol{x}), \mathsf{replace}(D_F, \Pi(\boldsymbol{x}))) \leq \epsilon.$$

▶ **Definition 13** (Interactive Non-Malleable Code). *A protocol encoding $\mathcal{E}$ is called a $(\delta, \epsilon)-$*interactive non-malleable code *for a family $\mathcal{F}$ of tampering functions and a class of protocols, if for any protocol Π of this class, $\mathcal{E}(\Pi)$ δ-correctly encodes Π and $\mathcal{E}(\Pi)$ is ϵ-protocol non-malleable for $\mathcal{F}$.*

3.3 Bounded State Tampering

We now define bounded state tampering functions for multi-party protocols. This is a natural model in which the adversary can *arbitrarily* and *jointly* tamper with *all* channels, however there exists an a priori upper bound on the size of the state they can hold. Similar classes of adversaries have already been considered starting with the work of Cachin and Maurer [14] which proposed encryption and key exchange protocols secure against computationally unbounded adversaries. With respect to non-malleable codes Faust et al. [32] introduced the notion of non-malleable codes against space-bounded tampering. Our formalization closely follows the one of Fleischhacker et al. [36] but adapted to the multi-party case. This means that we do not limit the size of the memory available for computing the tampering function in each round of tampering. Instead, we only limit the size of the state that can be carried over to the next round of tampering. I.e., an adversary in this model can jointly tamper with all of the messages exchanged in one round of execution depending on some function of *all* previously exchanged messages. *But* the function can only depend on up to some fixed number of s bits of information about previous messages. This is formalized as follows.

▶ **Definition 14** (Bounded State Tampering Functions)**.** *Functions of the class of s-bounded state tampering functions $F \in \mathcal{F}^s_{\mathsf{bounded}}$ for an interactive protocol are defined by a function*

$$f : \{0,1\}^s \cup \{\perp\} \times \mathcal{M}^{n\times n} \to \{0,1\}^s \times \mathcal{M}^{n\times n}$$

The function f takes as input a previous state of the tampering function and a matrix of sent messages and outputs an updated state and a matrix of received messages.

The full tampering function $F : \mathcal{T} \times \mathcal{M}^{n\times n} \to \mathcal{M}^{n\times n}$ is then defined in terms of f as seen below.

```
$F(\tau, \boldsymbol{S})$
$\sigma := \perp$
for $(\boldsymbol{S}', \boldsymbol{R}')$ in $\tau$
    $(\sigma, \boldsymbol{R}) := f(\sigma, \boldsymbol{S}')$
$(\sigma, \boldsymbol{R}) := f(\sigma, \boldsymbol{S})$
return $\boldsymbol{R}$
```

4 Arbitrary Message Topologies

The INMC for Bounded-state tampering functions that is introduced in Section 5 requires the underlying protocol to have a fixed message topology. In this section we show that is is not in general a restriction, as any protocol can be transformed protocol with a fixed message topology. Therefore, the INMC can be applied to *any* protocol by first transforming it to a protocol with a fixed message topology and then applying the INMC itself.

For this purpose we first introduce a general definition of a message topology, which for any party and round defines the *probability* that messages are received or sent over each channel, maximized over all possible inputs.

▶ **Definition 15** (Message Topology). *Let Π be interactive protocol with n parties defined by next message functions π_i and output functions out_i. The message topology of Π is defined by a function $\mu : [n] \times \mathbb{N} \to [0,1]^n \times [0,1]^n$, such that for all $i \in [n]$, and all $r \in \mathbb{N}$ it holds that $\mu(i, r) = (\boldsymbol{v}', \boldsymbol{s}')$, where*

$$v'_j := \max_{\boldsymbol{x}} \left\{ \Pr\left[\begin{matrix} \boldsymbol{\omega} \leftarrow \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n \\ \tau \leftarrow \mathsf{Trans}_\Pi(\boldsymbol{x}, \boldsymbol{\omega}) \end{matrix} : |V_i(\tau)| \geq r \wedge V_i(\tau)_{r,j} \neq \bot \right] \right\}$$

$$s'_j := \max_{\boldsymbol{x}} \left\{ \Pr\left[\begin{matrix} \boldsymbol{\omega} \leftarrow \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n \\ \tau \leftarrow \mathsf{Trans}_\Pi(\boldsymbol{x}, \boldsymbol{\omega}) \end{matrix} : |V_i(\tau)| \geq r \wedge \pi_i(x_i, \omega_i, V_i(\tau)_{\leq r})_j \neq \bot \right] \right\}$$

for $j \in [n]$.

A fixed message topology can then be seen as a special case, where μ is defined over $\{0, 1\}$ and the probabilities used in the definition are *independent of the input vectors.*

4.1 Transformations from Arbitrary to Fixed Message Topology

We propose three different transformations from an arbitrary message topology (AMT) to a fixed message topology (FMT). The first transformation is very naive, resulting in a very large blowup of the communication complexity but can be applied without any detailed consideration to the original message topology, i.e. it does not even reference the above definition. The second transformation *considers* the original message topology and will result in a lower blowup in the communication complexity for most reasonable protocols. However, in the worst case, for pathological examples, it can still result in the same blowup as the naive transformation. The third transformation finally allows us to limit the blowup, even in the worst case, but at the cost of potentially degrading the correctness of the protocol. Throughout this section, we assume that there exists a fixed upper bound on the number rounds the execution of a protocol may take.

4.1.1 Trivial Transformation

The simplest transformation floods the entire network in every round by sending dummy messages whenever there's no actual message to be sent. To reliably distinguish between real and dummy messages, real messages are marked by a prefix identifying them as real. Specifically, if in any round of the original protocol an actual message is sent by party P_i to party P_j then party P_i just prepends 1 to the message and sends it to the concerned party P_j. If on the other hand *no* message is sent in the original protocol a dummy message consisting of 0 is sent to P_j. To formally describe the next-message functions of the transformed protocol, we first define two functions $\mathsf{addDummies}$ and $\mathsf{remDummies}$ used to add and remove the dummy messages. We define the function

$$\mathsf{addDummies} : [n] \times [r_{\max}] \times (\{0,1\}^* \cup \{\bot\})^n \to (\{0,1\}^* \cup \{\bot\})^n$$

as

$$\mathsf{addDummies}(i, r, \boldsymbol{m}) := \boldsymbol{m}', \quad \text{where} \quad m'_j := \begin{cases} \bot & \text{if } i = j \\ 0 & \text{if } m_j = \bot \text{ and } i \neq j \\ 1 \| m_j & \text{otherwise.}^1 \end{cases}$$

and the function

[1] Note that $\mathsf{addDummies}$ takes an input r which is then ignored. This will make our lives easier when we modify the transformation going forward.

$$\mathsf{remDummies} : (\{0,1\}^* \cup \{\bot\})^n \to (\{0,1\}^* \cup \{\bot\})^n$$

as

$$\mathsf{remDummies}(\boldsymbol{m}) := \boldsymbol{m}', \quad \text{where} \quad m'_j := \begin{cases} \bot & \text{if } m_j \in \{0, \bot\} \\ m'' & \text{if } m_j = 1 \| m'' \end{cases}$$

For ease of notation we will apply remDummies to lists of vectors, which is to be interpreted as component-wise application. Let Π be an arbitrary ϵ-correct protocol described by next message functions π_i with an upper bound of $r_{\max}$ on the number of rounds. The next message function π'_i of the naively transformed protocol can then simply be defined as

$$\pi'_i(x_i, \omega_i, V_i) := \begin{cases} \bot^n & \text{if } |V_i| \geq r_{\max} \\ \mathsf{addDummies}(i, |V_i|, \pi_i(x_i, \omega_i, \mathsf{remDummies}(V_i))) & \text{otherwise.} \end{cases}$$

This trivially transformed protocol works exactly in the same way as the original protocol with the only exception being that in every round, all channels on which the original protocol would not have sent messages, the transformed protocol sends dummy messages and then promptly ignores them. The protocol terminates after exactly $r_{\max}$ rounds. This means that since the transformed protocol doesn't drop any messages and the original views of the parties can easily be reconstructed by ignoring the dummy messages. Hence it will still be ϵ-correct. This transformation clearly serves the purpose of transforming any protocol into a protocol with a fixed message topology. A clear downside, however, is the blowup in communication complexity, especially if the original protocol used a rather sparse communication graph. In every round each party starts sends messages to every other party whether they were expecting messages or not. In the worst case, this means that the expected communication complexity of the protocol blows up infinitely.[2] But even in more reasonable protocols that happen to use a sparse communication graph, the blowup is quite severe. Luckily we can do a bit better at least for reasonable protocols.

4.1.2 Maintaining the Communication Graph

The overhead of the transformation can be reduced if we only flood those channels where messages *could possibly be sent.* In order for that to happen we let party P_i send a dummy message only on those channels where there's a *non-zero* probability of a real message being sent. We can achieve that if we redefine the function addDummies as follows

$$\mathsf{addDummies}(i, r, \boldsymbol{m}) := \boldsymbol{m}', \text{ where } m'_j := \begin{cases} 0 & \text{if } \boldsymbol{m}'_j = \bot \text{ and } \mu(i,r)_{2,j} > 0 \\ 1 \| \boldsymbol{m}'_j & \text{if } \boldsymbol{m}'_j \neq \bot \\ \bot & \text{otherwise.} \end{cases}$$

The next message function is still defined as before. Clearly, this again results in a fixed message topology. This transformed protocol will also be ϵ-correct if the actual protocol is ϵ-correct as no messages are dropped and the original view can be reconstructed. Even though this transformation eliminates quite a lot of redundant messages and will result in a much smaller blowup for many protocols run over a sparse communication graph, the worst case blowup still remains infinite by the same argument as before.

[2] An example of a pathological protocol that exhibits infinite blowup is a protocol with at most one round, where one party sends a message with probability ζ, where ζ tends towards 0.

4.1.3 Dropping Low Probability Messages

To fix this issue of an infinite blowup in the expected communication complexity we can modify the transformation, by introducing a threshold value t and dropping all messages that are sent with probability less than t. In order to achieve this we again redefine the function addDummies as as

$$\mathsf{addDummies}(i, r, \boldsymbol{m}) := \boldsymbol{m}', \text{ where } m'_j := \begin{cases} 0 & \text{if } \boldsymbol{m}'_j = \bot \text{ and } \mu(i,r)_{2,j} > t \\ 1\|\boldsymbol{m}'_j & \text{if } \boldsymbol{m}'_j \neq \bot \text{ and } \mu(i,r)_{2,j} > t \\ \bot & \text{otherwise.} \end{cases}$$

This transformation only allows messages to be sent if their probability is above the threshold probability of t. The implementation of the next message function π_i remains the same for this transformation as was for the last transformation. For this transformation potentially degrades the correctness of the protocol. In a protocol with $r_{\max}$ rounds there are $r_{\max}(n^2 - n)$ potential messages, each of which may have a probability of being sent infinitesimally less than t. Each message that *is actually sent* by the underlying protocol but them blocked by the transformation can result in the protocol computing an incorrect output. But we can apply a union bound and get that the transformed protocol remains $\epsilon + tr_{\max}(n^2 - n)$-correct. However, this degraded correctness buys us a finite bound on the blowup of the expected communication complexity. In the worst case, each message in the original protocol is sent with probability exactly t, whereas it is sent with probability 1 in the transformed protocol. Therefore, the blowup can be at most $1/t$ therefore allowing us to bound the blowup.

5 An INMC for Bounded-State Tampering Functions

We devise an interactive non-malleable code for bounded state tampering functions that can be applied to any multi-party protocol Π' with *fixed message topology*, i.e., to any protocol where for every party P_i and every invocation r of the next message function π_i, whether or not a message is sent to party P_j is a priori known and does *not* depend on any of $\pi'_i s$ inputs. The basic idea is that each pair of parties will first run a key-exchange in which they will exchange enough key-material to the execute the original protocol encrypted under an information theoretically secure encryption scheme and authenticated with a statistically unforgeable MAC. Besides making sure that the tampering function cannot replay, redirect or omit messages by binding the authentication to a specific channel and including message counters in the authentication, the main challenge is to construct a key exchange that is secure against a computationally unbounded but bounded state adversary. We achieve this using a 2-non-malleable extractor. Essentially each party chooses a key by choosing two random sources α, β which will be much longer than the bounded state of the tampering function and extracting a key $k := \mathsf{nmExt}(\alpha, \beta)$. They will be using this key which they *know* is untampered to *encrypt* messages and to *verify* authentication tags. The two sources α, β are then sent in seperate rounds, ensuring that they cannot be tampered jointly, except for some amount of leakage through the state of the tampering function and potentially conditional aborts. This leakage can be handled by reinterpreting the sources as coming from a different distribution with slightly less min-entropy. Once the keys are exchanged, the parties verify that the keys were not modified in transit by sending a MAC computed over the ID of the channel with the key they *received* from the other party.

```
$\mathsf{CheckMsgOrder}(V)$
$\boldsymbol{r} = 0^n$
for $j \in [n] \setminus \{i\}$ do
  if $V_{|V|,j} \neq \bot$
    $r_j := 1$
if $\mu(i, |V| - 3)_1 \neq \boldsymbol{r}$
  return 0
return 1
```

Figure 1 The function checking the ordering of messages against the fixed message topology.

```
$\pi_i^{\mathsf{O}'_{\boldsymbol{x},\boldsymbol{\omega}'}}(\boldsymbol{\omega}, V_i(\tau))$
$\boldsymbol{s} := \bot^n$
if $|V| = 0$
  $\mathsf{rec} := 0^n, \mathsf{sent} := 0^n$
  $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) \leftarrow ((\{0,1\}^{\kappa_{nm}})^2)^n$
  $(\boldsymbol{k}_i^{enc}, \boldsymbol{k}_i^{auth}) := \mathsf{nmExt}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)$
  $\boldsymbol{s} := \boldsymbol{\alpha}_i$
elseif $|V| = 1$
  if $\bot \in V_1$
    abort
  $\boldsymbol{s} := \boldsymbol{\beta}_i$
elseif $|V| = 2$
  if $\bot \in V_2$
    abort
  $(\tilde{\boldsymbol{k}}_j^{enc}, \tilde{\boldsymbol{k}}_j^{auth}) := \mathsf{nmExt}(V_1, V_2)$
  for $j \in [n] \setminus \{i\}$ do
    $s_j := \mathsf{MAC}(\tilde{k}_{i,j}^{auth}, (i, j))$
else
  if $|V| = 3$
    if $\bot \in V_3$
      abort
    for $j \in [n] \setminus \{i\}$ do
      if $\mathsf{Vf}(k_i^{auth}, (j, i), V_{3,j}) = 0$
        abort
  else
    $\boldsymbol{m} := \bot^n$
    if $\mathsf{CheckMsgOrder}(V) = 0$
      abort
    else
      for $j \in [n] \setminus \{i\}$ do
        if $V_{|V|,j} \neq \bot$
          $(c, t) := V_{|V|,j}, \mathsf{rec}_j := \mathsf{rec}_j + 1$
          if $\mathsf{Vf}(k_i^{auth}, (c, j, i, \mathsf{rec}_j), t) = 0$
            abort
          $m_j := \mathsf{Dec}(\tilde{k}_j^{enc}, c)$
  $\boldsymbol{s}' \leftarrow \mathsf{O}'_{\boldsymbol{x},\boldsymbol{\omega}'}(\boldsymbol{m})$
  for $j \in [n] \setminus \{i\}$ do
    if $s'_j \neq \bot$
      $c := \mathsf{Enc}(\tilde{k}_i^{enc}, s'_j), \mathsf{sent}_j := \mathsf{sent}_j + 1$
      $s_j := (c, \mathsf{MAC}(\tilde{k}_j^{auth}, (c, i, j, \mathsf{sent}_j)))$
return $\boldsymbol{s}$
```

Figure 2 The next message function describing the INMC for bounded state tampering functions. For the sake of readability, we write the function as if it were stateful. I.e., in particular the variables rec and sent retain their value accross different invocations of π_i and do not need to be recomputed.

5.1 Defining the Next Message Function

The INMC is restricted to protocols with a fixed message topology as defined in Definition 11. Refer to Section 4 for a discussion on how arbitrary protocols can be transformed into protocols with a fixed message topology. To formally describe the next message function and output function of the INMC, we need an algorithm that checks whether the sequence of messages received from the other parties involved in the protocol confirm to the fixed message topology. The function CheckMsgOrder defined in Figure 1 allows to perform this check. Now that we have defined the CheckMsgOrder function we are ready to define the next message function in Figure 2. Remember, that according to Definition 8 an encoding is specified by an oracle machine or equivalently a next message function that is defined relative to an stateful oracle representing the next message function of the underlying protocol. The next message function $\pi_i^{\mathsf{O}_{i,\boldsymbol{x},\boldsymbol{\omega}'}}$ has three phases. In the initial phase every party shares their keys with the rest of the parties taking part in the protocol. In the next phase all of the parties confirms their respective keys with the other parties by sending a key confirmation value. The last phase of the execution of the next message function just deals with the actual message exchanges that happens between all the parties taking part in the protocol Π'. The output function out_i of the INMC simply takes the view V' of the underlying protocol that it can extract from it's own view exactly as in the next message function and outputs $\mathsf{out}'_i(\boldsymbol{x}, \boldsymbol{\omega'}, V')$ if the view conforms to the fixed message topology or $\perp$ otherwise.

▶ **Theorem 16.** *Let Π' be a protocol between n parties with fixed message topology, with $r = \max_{i,j\in[n]}\{\nu(i,j)\}$ and message length ℓ. If $(\mathsf{MAC}, \mathsf{Vf})$ is a statistically ϵ_{mac}-unforgeable $r+1$-time message authentication code with with message length $\ell + 2\lceil \log n \rceil + \lceil \log r \rceil$ and key length κ_{mac}, $(\mathsf{Enc}, \mathsf{Dec})$ is a perfectly indistinguishable stateful t-time encryption scheme with message length ℓ and key length κ_{enc}, and $\mathsf{nmExt} : \{0,1\}^{\kappa_{nm}} \times \{0,1\}^{\kappa_{nm}} \to \{0,1\}^{\kappa_{mac}+\kappa_{\mathsf{Enc}}}$ is an ϵ_{nm}-non-malleable 2-source extractor for sources with min-entropy at least $\kappa_{nm} - s - 3n\lambda$, then Π as described by π_i and out_i specified above is a $(0, (2n^2+n)\cdot 2^{-\lambda} + (n^2-n)\cdot\epsilon_{nm} + \epsilon_{\mathsf{MAC}})$-interactive non-malleable code for Π' for the class $\mathcal{F}^s_{\mathsf{bounded}}$ of bounded state tampering functions.*

Due to space constraints the proof is deferred to Appendix A.

References

1 Divesh Aggarwal, Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Optimal computational split-state non-malleable codes. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 393–417, Tel Aviv, Israel, January 10–13 2016. Springer, Heidelberg, Germany. `doi:10.1007/978-3-662-49099-0_15`.

2 Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 459–468, Portland, OR, USA, June 14–17 2015. ACM Press. `doi:10.1145/2746539.2746544`.

3 Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 774–783, New York, NY, USA, May 31 – June 3 2014. ACM Press. `doi:10.1145/2591796.2591804`.

4 Divesh Aggarwal, Nico Döttling, Jesper Buus Nielsen, Maciej Obremski, and Erick Purwanto. Continuous non-malleable codes in the 8-split-state model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 531–561, Darmstadt, Germany, May 19–23 2019. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-17653-2_18`.

5 Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Explicit non-malleable codes against bit-wise tampering and permutations. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 538–557, Santa Barbara, CA, USA, August 16–20 2015. Springer, Heidelberg, Germany. `doi:10.1007/978-3-662-47989-6_26`.

6 Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 375–397, Warsaw, Poland, March 23–25 2015. Springer, Heidelberg, Germany. `doi:10.1007/978-3-662-46494-6_16`.

7 Saikrishna Badrinarayanan and Akshayaram Srinivasan. Revisiting non-malleable secret sharing. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 593–622, Darmstadt, Germany, May 19–23 2019. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-17653-2_20`.

8 Marshall Ball, Dana Dachman-Soled, Siyao Guo, Tal Malkin, and Li-Yang Tan. Non-malleable codes for small-depth circuits. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 826–837, Paris, France, October 7–9 2018. IEEE Computer Society Press. doi:10.1109/FOCS.2018.00083.

9 Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, Huijia Lin, and Tal Malkin. Non-malleable codes against bounded polynomial time tampering. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 501–530, Darmstadt, Germany, May 19–23 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17653-2_17.

10 Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 881–908, Vienna, Austria, May 8–12 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_31.

11 Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 618–650, Tel Aviv, Israel, April 29 – May 3 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-78372-7_20.

12 Marshall Ball, Siyao Guo, and Daniel Wichs. Non-malleable codes for decision trees. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 413–434, Santa Barbara, CA, USA, August 18–22 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26948-7_15.

13 Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for interactive communication correcting insertions and deletions. *IEEE Transactions on Information Theory*, 63(10):6256–6270, 2017. doi:10.1109/TIT.2017.2734881.

14 Christian Cachin and Ueli M. Maurer. Unconditional security against memory-bounded adversaries. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 292–306, Santa Barbara, CA, USA, August 17–21 1997. Springer, Heidelberg, Germany. doi:10.1007/BFb0052243.

15 Nishanth Chandran, Bhavana Kanukurthi, and Srinivasan Raghuraman. Information-theoretic local non-malleable codes and their applications. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 367–392, Tel Aviv, Israel, January 10–13 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49099-0_14.

16 Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 285–298, Cambridge, MA, USA, June 18–21 2016. ACM Press. doi:10.1145/2897518.2897547.

17 Eshan Chattopadhyay and Xin Li. Non-malleable codes and extractors for small-depth circuits, and affine functions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 1171–1184, Montreal, QC, Canada, June 19–23 2017. ACM Press. doi:10.1145/3055399.3055483.

18 Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *55th Annual Symposium on Foundations of Computer Science*, pages 306–315, Philadelphia, PA, USA, October 18–21 2014. IEEE Computer Society Press. doi:10.1109/FOCS.2014.40.

19 Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*,

volume 8349 of *Lecture Notes in Computer Science*, pages 440–464, San Diego, CA, USA, February 24–26 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54242-8_19.

20 Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. *IEEE Transactions on Information Theory*, 62(3):1097–1118, March 2016.

21 Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. *Journal of Cryptology*, 30(1):191–241, January 2017. doi:10.1007/s00145-015-9219-z.

22 Eldon Chung, Maciej Obremski, and Divesh Aggarwal. Extractors: Low entropy requirements colliding with non-malleability. arXiv, 2021. doi:10.48550/arXiv.2111.04157.

23 Sandro Coretti, Yevgeniy Dodis, Björn Tackmann, and Daniele Venturi. Non-malleable encryption: Simpler, shorter, stronger. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 306–335, Tel Aviv, Israel, January 10–13 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49096-9_13.

24 Sandro Coretti, Antonio Faonio, and Daniele Venturi. Rate-optimizing compilers for continuously non-malleable codes. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 3–23, Bogota, Colombia, June 5–7 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-21568-2_1.

25 Sandro Coretti, Ueli Maurer, Björn Tackmann, and Daniele Venturi. From single-bit to multi-bit public-key encryption via non-malleable codes. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 532–560, Warsaw, Poland, March 23–25 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_22.

26 Dana Dachman-Soled, Ilan Komargodski, and Rafael Pass. Non-malleable codes for bounded parallel-time tampering. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 535–565, Virtual Event, August 16–20 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-84252-9_18.

27 Dana Dachman-Soled, Mukul Kulkarni, and Aria Shahverdi. Tight upper and lower bounds for leakage-resilient, locally decodable and updatable non-malleable codes. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 310–332, Amsterdam, The Netherlands, March 28–31 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-54365-8_13.

28 Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 427–450, Warsaw, Poland, March 23–25 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_18.

29 Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 239–257, Santa Barbara, CA, USA, August 18–22 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40084-1_14.

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

31 Antonio Faonio, Jesper Buus Nielsen, Mark Simkin, and Daniele Venturi. Continuously non-malleable codes with split-state refresh. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, volume 10892 of *Lecture Notes in Computer Science*, pages 121–139, Leuven, Belgium, July 2–4 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-93387-0_7.

32 Sebastian Faust, Kristina Hostáková, Pratyay Mukherjee, and Daniele Venturi. Non-malleable codes for space-bounded tampering. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 95–126, Santa Barbara, CA, USA, August 20–24 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-63715-0_4.

33 Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 465–488, San Diego, CA, USA, February 24–26 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54242-8_20.

34 Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. A tamper and leakage resilient von neumann architecture. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 579–603, Gaithersburg, MD, USA, March 30 – April 1 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46447-2_26.

35 Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 111–128, Copenhagen, Denmark, May 11–15 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-55220-5_7.

36 Nils Fleischhacker, Vipul Goyal, Abhishek Jain, Anat Paskin-Cherniavsky, and Slava Radune. Interactive non-malleable codes. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 233–263, Nuremberg, Germany, December 1–5 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-36033-7_9.

37 Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM Symposium on Theory of Computing*, pages 685–698, Los Angeles, CA, USA, June 25–29 2018. ACM Press. doi:10.1145/3188745.3188872.

38 Vipul Goyal and Ashutosh Kumar. Non-malleable secret sharing for general access structures. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 501–530, Santa Barbara, CA, USA, August 19–23 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96884-1_17.

39 Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 1128–1141, Cambridge, MA, USA, June 18–21 2016. ACM Press. doi:10.1145/2897518.2897657.

40 Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Four-state non-malleable codes with explicit constant rate. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 344–375, Baltimore, MD, USA, November 12–15 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-70503-3_11.

41 Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Non-malleable randomness encoders and their applications. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 589–617, Tel Aviv, Israel, April 29 – May 3 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-78372-7_19.

42 Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM*

Symposium on Theory of Computing, pages 1144–1156, Montreal, QC, Canada, June 19–23 2017. ACM Press. doi:10.1145/3055399.3055486.

43 Xin Li. Non-malleable extractors and non-malleable codes: Partially optimal constructions. In *Proceedings of the 34th Computational Complexity Conference*, CCC '19, Dagstuhl, DEU, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CCC.2019.28.

44 Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. arXiv, 2023. doi:10.48550/arXiv.2303.06802.

45 Fuchun Lin. Non-malleable multi-party computation. Cryptology ePrint Archive, Report 2022/978, 2022. URL: https://eprint.iacr.org/2022/978.

46 Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 517–532, Santa Barbara, CA, USA, August 19–23 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_30.

47 Rafail Ostrovsky, Giuseppe Persiano, Daniele Venturi, and Ivan Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 608–639, Santa Barbara, CA, USA, August 19–23 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96878-0_21.

48 Sridhar Rajagopalan and Leonard J. Schulman. A coding theorem for distributed computation. In *26th Annual ACM Symposium on Theory of Computing*, pages 790–799, Montréal, Québec, Canada, May 23–25 1994. ACM Press. doi:10.1145/195058.195462.

49 Leonard J. Schulman. Communication on noisy channels: A coding theorem for computation. In *33rd Annual Symposium on Foundations of Computer Science*, pages 724–733, Pittsburgh, PA, USA, October 24–27 1992. IEEE Computer Society Press. doi:10.1109/SFCS.1992.267778.

50 Leonard J. Schulman. Deterministic coding for interactive communication. In *25th Annual ACM Symposium on Theory of Computing*, pages 747–756, San Diego, CA, USA, May 16–18 1993. ACM Press. doi:10.1145/167088.167279.

51 Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, November 1996.

52 Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

53 Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. doi:10.1016/0022-0000(81)90033-7.

A Proof of Main Theorem

Proof. In order to prove that the protocol Π is a $(0, \epsilon)$-interactive non-malleable code we need to prove the *correctness* as well as *non-malleability* of the protocol as stated in Lemma 17 and Lemma 18.

▶ **Lemma 17.** *For any protocol* Π'*,* Π 0*-correctly encodes* Π'*.*

Proof. The extractor is deterministic and hence all the parties involved in the protocol will extract identical keys in an untampered execution. Since the MAC is correct, and tags are computed and verified with the correct keys, all messages will always verify and no party will abort during the protocol. As the correctness of the stateful encryption scheme Enc allows each party decrypt all received messages correctly all the parties will be able to faithfully execute a perfectly honest untampered execution of the underlying protocol Π. Therefore Π evaluates correctly whith the same probability as Π'. ◀

▶ **Lemma 18.** *The interactive protocol* Π *is* ϵ*-protocol non-malleable, where* $\epsilon = (2n^2 + n) \cdot 2^{-\lambda} + (n^2 - n) \cdot \epsilon_{nm} + \epsilon_{\mathsf{MAC}}$.

Proof. In order to show that the coding scheme is non-malleable we need to provide a *distribution* D_F as defined in Definition 12. In order to achieve a sampler for the distribution D_F, we start with the output distribution of an honest execution of the actual protocol and modify it through a serie of hyrids, until we reach a distribution that can be sampled independently of $\boldsymbol{x}$. To define the different hybrid distributions, first define a function $\bar{V}_i$ which essentially gives us the equivalent of a party's current view in the protocol, but replaces all received messages, with the messages that were originally sent.

```
$\bar{V}(\tau)$
$V := V_i(\tau)$
for $j \in [n]$ do
  $s := (S_{i,j} \mid (S, R) \in \tau \wedge S_{i,j} \neq \bot)$
  $c := 1$
  for $q \in [|V|]$ do
    if $V_{q,j} \neq \bot$
      $V_{q,j} := t_{j,c}$
      $c = c + 1$
return $V$
```

Now, let $F \in \mathcal{F}^s_{\mathsf{bounded}}$ be an arbitrary tampering function. For $i, j \in [n]$ and $o \in \{\alpha, \beta\}$, let $\zeta_{\alpha,i,j}$ be the probability that the tampering function modifies or drops $o_{i,j}$ during an execution of the protocol. We define the modified tampering function F' which behaves exactly like F, but for any (i, j, o) such that $\zeta_{\alpha,i,j} < 2^{-\lambda}$ it always keeps $o_{i,j}$ unmodified. We then further define for $i \in [n]$ and $r \in \{1, 2, 3\}$, $\gamma_{i,r}$ to be the probability that in an execution tampered by F', π_i aborts in execution round r, i.e., after receiving the αs, after receiving the βs, or after receiving the key confirmation values. Finally, let $\boldsymbol{x}' \in X_1 \times \cdots \times X_n$ be arbitrary but fixed. We then define several variants of $\mathsf{Execute}$, Next, and π_i in Figure 3 and Figure 4 respectively and are then finally ready to specify a series of hybrid distribution we construct to reach the distribution that corresponds to $\mathsf{replace}(D_F, \Pi(\boldsymbol{x}))$.

$\boldsymbol{H_0}$: Hybrid 0 is the original output distribution of a tampered execution. I.e, $H_0 = \mathsf{Execute}_{\Pi,F}(\boldsymbol{x})$.

$\boldsymbol{H_1}$: Hybrid 1 is still the distribution of a tampered execution, however we replace the tampering function with the modified tampering function F'. I.e., $H_1 = \mathsf{Execute}_{\Pi,F'}(\boldsymbol{x})$.

$\boldsymbol{H_2}$: In hybrid 2, we switch to using the modified execution algorithm $\mathsf{Execute}^1$ and Π^1. This change gives the next message function access to the message it *should* have received, i.e., those that were originally sent. I.e., $H_2\mathsf{Execute}^1_{\Pi^1,F'}(\boldsymbol{x})$.

$\boldsymbol{H_3}$: In hybrid 3 we switch to using Π^2, which means that parties that abort with overwhelming probability during the key exchange or key confirmation phase, now abort with probability 1. I.e., $H_3 = \mathsf{Execute}^1_{\Pi^2,F'}(\boldsymbol{x})$.

$\boldsymbol{H_4}$: In hybrid 4, we switch to using Π^3 which means that the keys are now no longer extracted but instead sampled uniformly at random on the sender's side and according to D_f on the receiver's side, where f is a split state tampering function induced by F'. I.e., $H_4 = \mathsf{Execute}^1_{\Pi^3,F'}(\boldsymbol{x})$.

$\boldsymbol{H_5}$: Hybrid 5 switches to using Π^4, which means that instead of verifying MACs the next message functions now directly check if messages were modified or not. I.e., $H_5 = \mathsf{Execute}^1_{\Pi^4,F'}(\boldsymbol{x})$.

```
$\mathsf{Execute}^{\chi}_{\Pi,F}(\boldsymbol{x};\boldsymbol{\omega})$
$\tau := \emptyset$
$\boldsymbol{S} := \mathsf{Next}^{1}(\Pi,\boldsymbol{x},\boldsymbol{\omega},\tau)$                         // $\chi = 1$
$\boldsymbol{R} := F(\tau,\boldsymbol{S})$
while $\boldsymbol{R} \neq \bot^{n\times n}$
    $\tau := \tau \circ (\boldsymbol{S},\boldsymbol{R})$
    $\boldsymbol{S} := \mathsf{Next}^{1}(\Pi,\boldsymbol{x},\boldsymbol{\omega},\tau)$                     // $\chi = 1$
    $\boldsymbol{R} := F(\tau,\boldsymbol{S})$
return $\big(\mathsf{out}_1(x_1,V_1(\tau)),\ldots,\mathsf{out}_n(x_n,V_n(\tau))\big)$   // $\chi = 1$
return $\begin{pmatrix}\mathsf{indicate}(\mathsf{out}_1(x_1,V_1(\tau))),\\ \ldots,\\ \mathsf{indicate}(\mathsf{out}_n(x_n,V_n(\tau)))\end{pmatrix}$   // $\chi = 2$
```

```
$\mathsf{Next}^{1}_{\Pi}(\boldsymbol{x},\boldsymbol{\omega},\tau)$
parse $\tau = ((\boldsymbol{S_1},\boldsymbol{R_1}),\ldots,(\boldsymbol{S_\ell},\boldsymbol{R_\ell}))$
for $1 \leq i \leq n$ do
    if $\tau = \emptyset \ \vee\ \mathbf{col}_i(\boldsymbol{R_\ell})^{\top} \neq \bot^{n}$
        $\boldsymbol{s_i} := \pi^{l}_{i}(x_i,\omega_i,V_i(\tau),\bar{V}_i(\tau))$
    else
        $\boldsymbol{s_i} := \bot^{n}$
return $\begin{bmatrix}\boldsymbol{s_1}\\ \vdots\\ \boldsymbol{s_n}\end{bmatrix}^{\top}$
```

Figure 3 Variants of Execute and Next as used in the hybrid distributions. Differences from the original are highlighted in gray. The different versions of Execute used in the hybrids are differentiated by the index χ. Each line where differences exist is marked with a comment indicating for which values of χ this line will be executed.

H_6 : In hybrid 6 we switch to $\mathsf{Execute}^2$. This means that the execution no longer outputs the actual outputs of the parties. Instead it only indicates which parties produced an output and which aborted. The outputs of all non-aborting parties are then replaced by the outputs of an honest untampered execution of $\Pi(\boldsymbol{x})$. I.e., $H_6 = \mathsf{replace}(\mathsf{Execute}^2_{\Pi^4,F'}(\boldsymbol{x}),\Pi(\boldsymbol{x}))$.

H_7 : Finally in hybrid 7, we replace the input $\boldsymbol{x}$ of the tampered execution with the arbitrary fixed input $\boldsymbol{x'}$. I.e., $H_7 = \mathsf{replace}(\mathsf{Execute}^2_{\Pi^4,F'}(\boldsymbol{x'}),\Pi(\boldsymbol{x}))$.

We note, that in H_7, the distribution of $\mathsf{Execute}^2_{\Pi^4,F'}(\boldsymbol{x'})$ no longer depends on $\boldsymbol{x}$. I.e., we define D_F as $\mathsf{Execute}^2_{\Pi^4,F'}(\boldsymbol{x'})$ and it is then sufficient to bound that $\mathsf{SD}(H_0,H_7)$ to prove the Lemma. We do so by bounding the statistical distance of each pair of neighboring hybrids.

▷ Claim 19. $\mathsf{SD}(H_0,H_1) \leq 2(n^2-n)\cdot 2^{-\lambda}$.

Proof. In H_1 we replaced F with the modified tampering function F'. This function is modified such that a series of low probability events (that $o_{i,j}$ for $o \in \{\alpha,\beta\}$ and $i,j \in [n]$ is modified by F) does not happen. Each event happens with probability less than $2^{-\lambda}$. The number of events is bounded by two times the number of edges in the communication graph. This is a directed complete graph, i.e., the number of edges is n^2-n. Hence, by a union bound over all events, the statistical distance between hybrids H_0 and H_1 can be bounded by $2(n^2-n)\cdot 2^{-\lambda}$. ◁

▷ Claim 20. $\mathsf{SD}(H_1,H_2) = 0$

Proof. In hybrids H_1, H_2, it is easy to see that the only differences between the hybrids are syntactic. I.e., the next message function receives the additional input $\bar{V}_i(\tau)$ in H_2, but does not actually use it yet. Therefore the output distributions remain identical. ◁

▷ Claim 21. $\mathsf{SD}(H_2,H_3) \leq 3n\cdot 2^{-\lambda}$

Proof. In hybrid H_3 we eliminate another series of low probability events. If F' causes any of the parties to abort with overwhelming probability $> (1-2^{-\lambda})$ in the first three rounds of

```
$\pi_i^{d,\mathsf{O}'_{\boldsymbol{x},\boldsymbol{\omega}'}}(\boldsymbol{\omega}, V_i(\tau))$

$\boldsymbol{s} := \perp^n$
if $|V| = 0$
  $\mathsf{rec} := 0^n, \mathsf{sent} := 0^n$
  $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) \leftarrow ((\{0,1\}^{\kappa_{nm}})^2)^n$
  $(\boldsymbol{k}_i^{enc}, \boldsymbol{k}_i^{auth}) := \mathsf{nmExt}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)$    // $d < 3$
  $(\boldsymbol{k}_i^{enc}, \boldsymbol{k}_i^{auth}) \leftarrow \{0,1\}^{\kappa_{enc}+\kappa_{mac}}$    // $d \geq 3$
  $\boldsymbol{s} := \boldsymbol{\alpha}_i$
elseif $|V| = 1$
  if $\perp \in V_1$    // $d < 2$
  if $\perp \in V_1$ or $\gamma_{i,1} > 1 - 2^{-\lambda}$    // $d \geq 2$
    abort
  $\boldsymbol{s} := \boldsymbol{\beta}_i$
elseif $|V| = 2$
  if $\perp \in V_2$    // $d < 2$
  if $\perp \in V_2$ or $\gamma_{i,2} > 1 - 2^{-\lambda}$    // $d \geq 2$
    abort
  $(\tilde{\boldsymbol{k}}_j^{enc}, \tilde{\boldsymbol{k}}_j^{auth}) := \mathsf{nmExt}(V_1, V_2)$    // $d < 3$
  $(\tilde{\boldsymbol{k}}_j^{enc}, \tilde{\boldsymbol{k}}_j^{auth}) := \mathsf{replace}(D_f, (\boldsymbol{k}_j^{enc}, \boldsymbol{k}_j^{auth}))$    // $d \geq 3$
  for $j \in [n] \setminus \{i\}$ do
    $s_j := \mathsf{MAC}(\tilde{k}_{i,j}^{auth}, (i,j))$
else
  if $|V| = 3$
    if $\perp \in V_3$    // $d < 2$
    if $\perp \in V_3$ or $\gamma_{i,3} > 1 - 2^{-\lambda}$    // $d \geq 2$
      abort
    for $j \in [n] \setminus \{i\}$ do
      if $\mathsf{Vf}(k_i^{auth}, (j,i), V_{3,j}) = 0$    // $d < 4$
      if $(\tilde{k}_i \neq k_i)$ or $\bar{V}_{3,j} \neq V_{3,j}$    // $d \geq 4$
        abort
  else
    $\boldsymbol{m} := \perp^n$
    if $(\mathsf{CheckMsgOrder} = 0)$
      abort
    else
      for $j \in [n] \setminus \{i\}$ do
        if $V_{|V|,j} \neq \perp$
          $(c,t) := V_{|V|,j}, \mathsf{rec}_j := \mathsf{rec}_j + 1$
          if $\mathsf{Vf}(k_i^{auth}, (c,j,i,\mathsf{rec}_j), t) = 0$    // $d < 4$
          if $\bar{V}_{|V|,j} \neq V_{|V|,j}$    // $d \geq 4$
            abort
          $m_j := \mathsf{Dec}(\tilde{k}_j^{enc}, c)$
  $\boldsymbol{s}' \leftarrow \mathsf{O}'_{\boldsymbol{x},\boldsymbol{\omega}'}(\boldsymbol{m})$
  for $j \in [n] \setminus \{i\}$ do
    if $s'_j \neq \perp$
      $c := \mathsf{Enc}(\tilde{k}_i^{enc}, s'_j), \mathsf{sent}_j := \mathsf{sent}_j + 1$
      $s_j := (c, \mathsf{MAC}(\tilde{k}_j^{auth}, (c,i,j,\mathsf{sent}_j)))$
return $\boldsymbol{s}$
```

Figure 4 The modified next message functions used in the hybrid distributions. Differences from the original are highlighted in gray. The different next message functions used in the hybrids are differentiated by the index d. Each line where differences between next message functions exist is marked with a comment indicating for which values of d this line will be executed.

the protocol, i.e., during key-exchange or key-confirmation, the party now aborts at the same point in time with probability 1. I.e., each eliminated event, i.e. the "non-abort", happens with probability less than $2^{-\lambda}$. The number of eliminated events is bounded by three times the number of parties in the protocol. Therefore a union bound over all eliminated events gives us that the statistical distance between H_2 and H_3 can be bounded by $3n \cdot 2^{-\lambda}$. ◁

▷ Claim 22. $\mathsf{SD}(H_3, H_4) \leq (n^2 - n) \cdot \epsilon_{nm}$.

Proof. For any i, j, let $f_{i,j}$ be the tampering function for $\alpha_{i,j}, \beta_{i,j}$ induced by F. We observe that the changes that were made in H_4 are that rather than using the extracted keys the sender uses uniformly chosen keys while the receiver either receives keys that are distributed according to $D_{f_{i,j}}$ that is *independent* of the actual key, or it receives the same uniformly distributed key used by the sender.

Now, *if* $f_{i,j}$ were split state, then the non-malleability of the extractor would imply that the statistical distance caused by each replaced key can be at most ϵ_{nm}. The main issue is that $f_{i,j}$ is in fact *not* split state. The tampering function can use both, its bounded state as well as conditional aborts (and non-aborts) of the individual parties to leak information from the first part of the tampering function to the second part and from both parts to the rest of the protocol. However, if we can *bound* the amount of information that can be leaked, then we can change our perspective and look at $f_{i,j}$ as a split state tampering functions, that tampers with sources sampled from a distribution defined by sampling almost uniformly, but conditioned on the leakage.

It remains to actually bound the leakage. Clearly a tampering function in $\mathcal{F}^s_{\mathsf{bounded}}$ can leak s bits simply through its persistent state. Additional leakage is obtained by causing any of the parties to abort or not to abort with low probability. However, due to the elination of low probability events in previous hybrids, we know that each of these events happens with probability at *least* $2^{-\lambda}$. Per party there exist three abort/non-abort events, i.e. the tampering function can leak at most $3n \log \frac{1}{2^{-\lambda}} = 3n\lambda$ additional bits of information.

We can thus reinterpret $f_{i,j}$ as a split-state tampering function on sources with min-entropy $\kappa_{nm} - s - 3n\lambda$. Since, nmExt is specified as working with sources of this type, we have that each replaced key increases the statistical distance by at most ϵ_{nm}. As there are, as mentioned before, $(n^2 - n)$ keys to deal with, we can bound the total statistical distance between the hybrids H_1 and H_2 with $(n^2 - n) \cdot \epsilon_{nm}$. ◁

▷ Claim 23. $\mathsf{SD}(H_4, H_5) \leq \epsilon_{\mathsf{MAC}}$

Proof. Here we bound the statistical distance between the hybrids using a reduction from the statistical unforgeability of the MAC. The output distribution of the two hybrids only differs, if at any point one of the parties receives a ciphertext and tag pair (c, t) such that for some (i, j, r), $\mathsf{Vf}(k^{auth}_{i,j}, (c, i, j, r), t) = 1$ but where none of the parties ever computed $\mathsf{MAC}(k^{auth}_{i,j}, (c, i, j, r))$. That means that the statistical distance between the hybrids is equal to the probability that the above event occurs. We can then construct an attacker $\mathcal{A}$ against the MAC scheme as follows: $\mathcal{A}$ executes H_4 as specified, except that it ignores the actual authentication keys and instead uses the MAC oracle to compute all tags. When the event specified above occurs, $\mathcal{A}$ outputs $(c, i, j, r), t, i$. If the event never occurs, $\mathcal{A}$ aborts. Clearly $\mathcal{A}$ forges a MAC with probability $\mathsf{SD}(H_4, H_5)$. Since the MAC is ϵ_{mac}-statistically unforgeable, we therefore have $\mathsf{SD}(H_4, H_5) \leq \epsilon_{\mathsf{MAC}}$ as claimed. ◁

▷ Claim 24. $\mathsf{SD}(H_5, H_6) = 0$.

Proof. Due to the changes in the previous hybrids, we know that all messages received by any party that does not abort are exactly those messages that were originally sent. Further, whenever a party aborts it does not send any more messages, ensuring that all messages that *are* sent are computed solely based on untampered messages. Additionally, since the protocol has a fixed message topology and both the next message function as well as the output function check that the view conforms to this topology, we know that any party that does not abort computed their output based on a complete view consisting of honestly computed messages that were received in the correct order. I.e., in H_5 the outputs of the *non-aborting* parties are distributed according to the same distribution as in a completely untampered execution of Π on $\boldsymbol{x}$. In H_6, $\mathsf{Execute}^2_{\Pi^4, F}(\boldsymbol{x})$ returns $\perp$ for all aborting parties and same for all non-aborting parties. The function $\mathsf{indicate}$ then replaces the same entries with consistent outputs of an honest execution of $\Pi(\boldsymbol{x})$. Therefore the two distributions are identical. ◁

▷ Claim 25. $\mathsf{SD}(H_6, H_7) = 0$.

Proof. Since the message topology is fixed in both the hybrids, the "shape" of the transcripts of the underyling protocol during the execution in both the hybrids are identical, only the *content* of the messages might differ based on the inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$. However, due to the perfect indistingishability of the stateful encryption scheme, the distribution of the *ciphertexts* is identical. Therefore the distributions of the overall transcripts observed by the tampering function are identical and therefore, so are the output distributions. ◁

Using the triangle inequality over the bounds from Claim 19 through Claim 25 we can thus conclude that

$$\begin{aligned}
&\mathsf{SD}(\mathsf{Execute}_{\Pi,F}(\boldsymbol{x}), \mathsf{replace}(D_F, \Pi(\boldsymbol{x})))\\
&= \mathsf{SD}(\mathsf{Execute}_{\Pi,F}(\boldsymbol{x}), \mathsf{replace}(\mathsf{Execute}^2_{\Pi^4,F}(\boldsymbol{x}'), \Pi(\boldsymbol{x})))\\
&= \mathsf{SD}(H_0, H_7) \leq \sum_{i=1}^{7} \mathsf{SD}(H_{i-1}, H_i) = (2n^2 + n) \cdot 2^{-\lambda} + (n^2 - n) \cdot \epsilon_{nm} + \epsilon_{\mathsf{MAC}}
\end{aligned}$$ ◀

The theorem finally follows immediately from Lemma 17 and Lemma 18. ◀