Interactive Non-Malleable Codes
(Full Version)

Nils Fleischhacker[1]*, Vipul Goyal[2]**, Abhishek Jain[3]***,
Anat Paskin-Cherniavsky[4], and Slava Radune[4,5]

[1] Ruhr University Bochum, Bochum, Germany
[2] Carnegie Mellon University, Pittsburgh, USA
[3] Johns Hopkins University, Baltimore, USA
[4] Ariel University, Ariel, Israel
[5] The Open University of Israel, Ra'anana, Israel

Abstract. Non-malleable codes (NMC) introduced by Dziembowski et al. [ICS'10] allow one to encode "passive" data in such a manner that when a codeword is tampered, the original data either remains completely intact or is essentially destroyed.

In this work, we initiate the study of *interactive non-malleable codes* (INMCs) that allow for encoding "active communication" rather than passive data. An INMC allows two parties to engage in an interactive protocol such that an adversary who is able to tamper with the protocol messages either leaves the original transcript intact (i.e., the parties are able to reconstruct the original transcript) or the transcript is completely destroyed and replaced with an unrelated one.

We formalize a tampering model for interactive protocols and put forward the notion of INMCs. Since constructing INMCs for general adversaries is impossible (as in the case of non-malleable codes), we construct INMCs for several specific classes of tampering functions. These include bounded state, split state, and fragmented sliding window tampering functions. We also obtain lower bounds for threshold tampering functions via a connection to interactive coding. All of our results are unconditional.

1 Introduction

Error correcting codes allow a message m to be encoded into a codeword c, such that m can always be recovered even from a tampered codeword c' if the tampering is done in a specific way. More formally, the class of tampering functions, $\mathcal{F}$, tolerated by traditional error correction codes are ones that erase or modify only a constant fraction of the codeword c. However, no guarantees are provided on the output of the decoding algorithm when the tampering function $f \notin \mathcal{F}$. A more relaxed notion, error detecting codes, allows the decoder to also output a special symbol $\perp$ when m is unrecoverable from c'. But here too, the codes can not tolerate many simple tampering functions such as a constant function.

Non-malleable Codes. The seminal work of Dziembowski, Pietrzak, and Wichs [36] introduced the notion of non-malleable codes (NMC). Informally, an encoding scheme $\mathsf{code} := (\mathsf{Enc}, \mathsf{Dec})$ is an NMC against 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 equal to the original message m or the original message is essentially "destroyed" and m' is completely unrelated to m. In general, NMCs cannot exist for the set of all tampering functions $\mathcal{F}_{\mathsf{all}}$. To see this, observe that a tampering function that simply runs the decode algorithm to retrieve m and then encodes a message related to m trivially defeats the requirement above. 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 1.2 for a discussion).

* Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

** Vipul Goyal is supported in part by NSF grant 1916939, a gift from Ripple, a gift from DoS Networks, a JP Morgan Faculty Fellowship, and a Cylab seed funding award.

*** Abhishek Jain is supported in part by NSF SaTC grant 1814919 and Darpa Safeware grant W911NF-15-C-0213.

While non-malleable codes have the obvious advantage that one can obtain meaningful guarantees for a larger class of tampering functions (compared to error correcting codes), they have also found a number of interesting applications in cryptography. In particular, NMCs have found a number of applications in tamper-resilient cryptography [36, 60, 40, 41] and they have also been useful in constructing non-malleable encryption [29]. Recently, non-malleable codes were also used to obtain a round optimal protocol for non-malleable commitments [53], as well to build non-malleable secret sharing schemes [51, 52].

Interactive Non-Malleable Codes. In this work, we seek to generalize the notion of non-malleable codes. Regular non-malleable codes can be seen as dealing with "passive" data in that data is encoded and, upon being tampered, the data either remains completely intact or is essentially destroyed. Now consider the following scenario. Two parties, each holding their own inputs are interested in running a protocol to perform some task involving their inputs, such as computing a joint function on them. Now, say an adversary is able to somehow get access to their communication channel and modify messages being sent in the protocol. We would like to have a similar guarantee: either the original transcript of the underlying protocol remains fully recoverable from the encoded communication, or, very informally, the original transcript is essentially "destroyed" and any transcript possibly recovered is "unrelated" to the interaction that was originally supposed to take place. Hence, we are concerned with encoding "active communication" rather than passive data.

An interesting special case of the above scenario could also occur in terms of computation being performed on a piece of hardware. Suppose several different chips on an integrated circuit board are communicating via interconnecting wires to perform some computation on the secrets stored within them. An adversary could tamper in some way with the communication going through those wires. We would like to require that either the computation remains intact, or that the original computation is "destroyed" and whatever computation takes place is completely unrelated.

Of course, this basic idea raises a number of questions: What does it actually mean for a computation to be "unrelated" to another computation. How much power can the tampering adversary reasonably be allowed to have? Are we concerned with the secrecy of inputs in this setting?

In the setting of non-interactive non-malleable codes (INMCs), "unrelated" is easily defined as independent of the original message. However, in the interactive setting, things are a bit more complicated since there exists more than one input. Indeed, there are multiple notions of non-malleability that we can envision in the interactive setting. Below, we discuss possible notions of non-malleability.

Suppose, Alice and Bob are holding inputs x and y respectively and they jointly execute a protocol that results in a transcript τ when not tampered with. Now suppose an adversary tampers with the messages sent over the communication channel and Alice and Bob recover transcripts τ_1 and τ_2, respectively. A possible notion of non-malleability could then require that either $\tau_1 = \tau$ (i.e., the original transcript remains intact) or the distribution of τ_1 should be completely independent of Bob's input y.

Such a notion would still allow an adversary to simply "cut off" Bob from the communication and essentially execute the protocol honestly, but with a different input y'. Clearly, this is not an attack on the notion described above, since y' and thereby the resulting transcript τ_1 is distributed completely independently of y. Nevertheless, the notion is still not completely satisfying since the output under tampering still depends on *one* of the inputs.

We may, thus, consider a strengthening of the above notion, where a party must receive either the correct transcript τ or $\perp$. This notion is achievable if the tampering function is not strong enough to cut off and impersonate one of the parties. It is easy to see that this notion is stronger than error detection: whether or not a party receives $\perp$ must not depend on the inputs (x, y), i.e. input dependent aborts must be prevented.[6]

Even this stronger definition may still not be quite as strong as one might hope. It is entirely conceivable that a tampering function could exist such that the marginal distributions of Alice' and Bob's individual outputs are independent from both inputs, but where the *product* distribution of the two is not. The final definition we settle for, is therefore one where we require that both parties receive either the correct transcript τ or $\perp$ and the product distribution of the two outputs must be (almost) independent of the inputs.

[6] This is similar in spirit to the definition of non-malleable codes where, whether or not the decoder gets $\perp$, can also not depend upon the original message m.

We do not explicitly model any secrecy requirements for the inputs (x, y). We view non-malleability of codes in the interactive setting as a separate property and as such it should be studied independently. However, our definitions of encodings work by defining them using simulators relative to an underlying protocol. This formalization ensures that any security properties such as secrecy of inputs of the underlying protocol are preserved under the encoding.

Relationship to Non-Malleable Codes. Consider the message transfer functionality where the transcript is simply the transferred message x. An interactive non-malleable coding protocol for this functionality gives the following guarantee: Bob either receives x from Alice or a value x' unrelated to x. It is easy to see that a one round interactive non-malleable coding protocol for this message transfer functionality is the same as a non-malleable code (encoding message x) for the same class of tampering functions. Indeed, the question that we consider in our work can be seen as generalizing non-malleable codes to more complex protocols potentially involving multiple rounds of interaction and both inputs x and y.

Our notion of INMCs is harder to achieve in one sense since more complex functionalities are involved, and yet, is easier to achieve in another sense since one is allowed multiple rounds of interaction and the order of messages introduces a natural limit on the power of an adversary, since she cannot tamper depending on "future" messages.

Similar to non-malleable codes, INMCs are impossible to achieve for arbitrary tampering functions. Very roughly, consider the first message of the protocol transcript which contains non-trivial information about the input x of Alice. The adversary at this point decodes and reconstructs this partial information about the input x, chooses a related input x' consistent with the partial information and simply executes the protocol honestly with Bob from this point onwards (cutting Alice off completely). A similar argument can also be made for the other direction. In fact, we even rule out INMCs for a more restricted class of threshold tampering functions using a very similar argument in Section 4. This suggests that, similar to non-malleable codes, we must focus on specific function classes for building INMCs.

One seemingly obvious approach of constructing INMCs even for multi-round protocols would be to directly use non-malleable codes. I.e., encode each message of an underlying protocol independently. The hope would be that this results in an INMC that allows at least independent tampering of each message under the same class of tampering functions as the original NMC. However, this naïve approach fails to produce INMCs for any meaningful class of functions.

As a counter example consider the following protocol: Alice has inputs (x, y) and sends these to Bob in two separate messages. Bob receives the messages and outputs (x, y). With the above approach, x and y would be encoded separately as $\mathsf{Enc}(x), \mathsf{Enc}(y)$. Let f be any tampering function, such that decoding $\mathsf{Dec}(f(\mathsf{Enc}(x)) \neq x$. Such functions exist within the class of tampering functions against which the NMC is supposed to be secure, unless the NMC is in fact error correcting. A valid tampering function against the supposed INMC could then tamper with the first message using f and not tamper with the second message at all. This would result in Bob receiving $z \neq x$ and y and outputting (z, y). Clearly (z, y) and (x, y) are related. Therefore, the protocol is not non-malleable. This counter example works even when more complex constructions such as the NMC against streaming space-bounded tamperings by Ball et al. [11] are used.

An interesting additional hurdle that needs to be overcome when constructing INMCs when compared to non-malleable codes is inherent leakage. Because messages in the protocol are tampered successively, a tampering function can use conditional aborts to communicate some information to future tampering functions. Let $\mathcal{F}$ be some class of tampering functions. Say a tampering function $f \in \mathcal{F}$ looks at message m_i sent in round i of the protocol and aborts unless m_i is "good" in some sense. In future rounds, even if the definition of $\mathcal{F}$ precludes f from having any knowledge of m_i, the tampering function still learns that m_i must have been "good", since the protocol would have otherwise aborted. We deal with this inherent leakage by bounding the leakage and using leakage resilient tools.

Relationship to Interactive Coding. Our notion can be seen as inspired by the notion of interactive coding (IC) [64, 65, 66]. Essentially, INMCs are to non-malleable codes what IC is to error correcting codes. In interactive coding, we require that the original transcript must remain preserved in face of an adversary tampering the message over the communication channel. INMCs only require something weaker, namely, that either the transcript must remain preserved or that the original transcript be destroyed and any possibly reconstructed transcript be independent of the inputs to the protocol.

An obvious advantage of such a weaker notion is that one could hope to achieve it for a larger class of tampering functions compared to ICs. Indeed, ICs are achievable only for threshold adversaries, namely, an adversary which only tampers with a fixed threshold number of bits of the communication (typically a constant fraction of the entire communication). All guarantees are lost in the case an adversary tampers with more bits than allowed by this threshold. However, as we discuss later, INMCs are achievable for adversaries which could potentially tamper with *every* bit going over the communication channel. For the specific case of threshold tampering functions, however, we are able to show that lower bounds on the fraction of the communication that can be tampered with transfer from ICss to INMCs.

1.1 Our Results and Techniques

In this work we initiate the study of INMCs. We formalize the tampering model and put forward a notion of securityfor INMC. Since achieving INMC for general adversaries is impossible, we turn our attention to specific classes of tampering functions.

We show both positive and negative results. We first establish a negative result for threshold tampering functions by showing that INMCs for threshold tampering imply ICs for the same class of tampering functions, thereby transferring lower bounds from interactive coding to INMCs. We then provide several positive results for specific classes of tampering functions by constructing general (unconditional) compilers Σ that can encode an arbitrary underlying protocol Π in a non-malleable fashion (for the appropriate class of tampering functions).

Threshold Tampering Functions. A threshold tampering function is not restricted in its knowledge of the protocol transcript or in its computational power, but can only modify a fixed fraction (say 1/4) of the bits in the transcript. For this class, lower bounds are known for the case of interactive coding. Specifically Braverman and Rao [18] showed that non-adaptive IC can tolerate tampering with at most 1/4 of the transcript, and Ghaffari, Haeupler, and Sudan [50] showed that an adaptive IC can tolerate tampering with at most 2/7 of the transcript. When looking for stronger classes of tampering functions, the first natural question to ask is therefore whether the weaker notion of INMCs might allow us to circumvent these lower bounds. However, it turns out that this is not the case.

We show that any INMC for a class of threshold tampering functions that allows only a negligible non-malleability error in fact implies an IC for the same class of functions in the *common reference string* (CRS) model and with parties running in super-polynomial time. While the resulting IC is not efficient and requires a CRS, it turns out that the lower bounds of Braverman and Rao [18] and Ghaffari, Haeupler, and Sudan [50] also apply in this setting, therefore ruling out the existence of such INMCs. This result can be found in Section 4. In fact, this impossibility even holds if we apply the notion of INMC to a weaker notion of encodings which does not imply knowledge-preservation. Recall that we are using a strong notion of protocol encoding that ensures that security guarantees of the underlying protocol are preserved. On the flip side, positive results for IC only translate to the positive result for this weaker notion of INMC. Getting meaningful positive result for our stronger INMC definition is an interesting open problem.

Interestingly (and fortunately), the above connection only holds for threshold tampering functions. Indeed, for the remaining families of tampering functions we consider in this paper, IC is naturally impossible and yet we are able to get positive results for INMC.

Bounded State Tampering Functions. For our first positive result we consider the class of tampering functions which can keep a bounded state. In more detail, the adversary is assumed to be arbitrarily computationally powerful, and we do not limit the size of the memory available for computing the tampering function. Instead, a limit is only placed on the size of the state that can be carried over from tampering one message to tampering with the next. That is, an adversary in this model can iteratively tamper with each message depending on some function of *all* previous messages, but the *size* of this information is limited to some fixed number of bits s. It is easy to see that achieving the notion of error correction is impossible for such a tampering function family since an adversary even with no storage can change every protocol message to an all zero string.

Adversaries with limited storage capabilities constitute a very natural model and similar adversaries have been considered before in many settings, starting with the work by Cachin and Maurer [19] on encryption and key exchange secure against computationally unbounded adversaries. In a seemingly related recent work, Faust et al. [39] studied non-malleable codes against space-bounded tampering. However in their setting, a limit is placed on the size of memory available to compute the tampering function (indeed it is meaningless to consider the state carried over from one message to the next in the non-interactive setting).

We give an unconditional positive result for this family of tampering functions: Any underlying protocol Π can be simulated by a protocol Σ which is an INMC against bounded state tampering functions. A naïve way of trying to construct such a compiler would be to try and encode each message of Π using a suitable (non-interactive) non-malleable code. However, this is doomed to fail. For a single message setting, our tampering adversary simply translates to an unbounded general adversary for which designing non-malleable codes is known to be impossible. Hence, getting a positive result inherently relies on making use of additional interaction.

The key technical tool we rely on to construct our compiler is the notion of seedless 2-non-malleable extractors introduced by Cheraghchi and Guruswami [25] as a natural generalization of seeded non-malleable extractors [34]. However, finding an explicit construction of such extractors was left as an open problem by Cheraghchi and Guruswami even for the case when both the sources are uniform. Such a construction was first given by Chattopadhyay, Goyal, and Li [22]. The construction in [22] requires one of the sources to be (almost) uniform, while the other source could have smaller min-entropy. We crucially rely upon a construction of seedless 2-non-malleable extractors where at least one of the sources could have small min-entropy. Our construction can be found in Section 5.

Split-State Tampering Functions. The second class we consider are split-state tampering functions where, very roughly, the transcript is divided into two disjoint sets of messages and each set is tampered independently. In more detail, the adversary can decide for each message of the protocol to be either in the first set or the second one. To compute an outgoing message, the tampering function takes all messages (so far) in any one set of its choice as input.

We are able to achieve interactive non-malleability for a strong class of these tampering functions, namely c-unbalanced split-state tampering functions. A c-unbalanced split-state tampering functions can split the transcript into two arbitrary sets, as long as each set contains at least a $1/c$ fraction of the messages (where c can be any polynomial parameter).

This notion is inspired by a corresponding notion in the non-interactive setting. Split-state tampering functions for non-interactive NMC are one of the most interesting and well studied classes of tampering functions in that setting. It was already introduced in the seminal work of Dziembowski, Pietrzak, and Wichs [36] and has since then been studied in a large number of works [60, 35, 3, 25, 24, 2, 26].

We give an unconditional positive result for this family of tampering functions: Any underlying protocol Π can be simulated by a protocol Σ which is an INMC against split-state tampering functions. The key technical tool we rely on in this case is a new notion of tamper evident n-out-of-n secret sharing we introduce in this work. Such a secret sharing scheme essentially guarantees that any detectable tampering with the shares can be detected when reconstructing the secret. Our construction can be found in Section 6.

Sliding Window Tampering Function. In the sliding window model, the tampering function "remembers" only the last w messages. In other words, the tampering function gets as input the last w (untampered) messages of the protocol transcript to compute the tampered message. The sliding window model is very natural and has been considered in a variety of contexts, such as error correcting codes [48] including convolution codes, streaming algorithms, and even in data transmission protocols such as TCP [55].

Our results in fact extend to a stronger model in which we can handle what we call *fragmented sliding window* tampering functions. Functions in this class are allowed to remember *any* w of the previous protocol messages (rather than just the w most recent ones). Thus in some sense, the window of message being stored by the tampering function is not continuous but "fragmented".

Comparing this class of functions with bounded-state tampering functions, we can see, that here the tampering function can no longer retain *some* information about *all* previous messages, but instead *all* of the information about *some* previous messages. Because there is no hard bound on the size of the state, but instead on the number of messages which potentially differ in length, this means that the two models are incomparable.

Comparing this class with c-unbalanced split-state tampering functions, we notice that here the maximum size of the window is fixed and does not scale with the number of messages in the protocol. On the other hand, however, the different sets of messages which the tampering can depend on are not required to be disjoint. E.g., the tampering of each single protocol messages could depend on the first message of the protocol, something that would not be possible in the case of split-state functions.

While this model has important conceptual differences to the our split state model, the techniques used to achieve both of them are almost identical. In particular, essentially the same protocol as in the case of c-unbalanced split-state tampering functions also works in this case, however the proof of security differs slightly. Our construction can be found in Section 7.

A Common Approach. A common theme in all of our constructions is the following: We only attempt to transfer *a single* message in a non-malleable way and then use this message to secure the rest of the protocol. In more detail, Alice and Bob essentially exchange a random key k possibly using multiple rounds of interaction such that the following holds. The two parties either agree on the correct key k or receive completely independent keys k_1 and k_2, (or, $\perp$ which leads them to abort the protocol). Subsequently, all future protocol messages will be encrypted with a one-time pad and authenticated with a one-time message authentication code using k (assuming k is long enough). This allows us to achieve non-malleability as long as we can ensure that the tampering function is not capable of predicting the exchanged key in any round. The reason is as follows: as long as the key remains (almost) uniformly distributed from the point of view of the tampering function f, the computation of f cannot depend on the encrypted messages, and any modification of the encrypted messages would be caught by the MAC and cause an abort independently of the inputs. The exact way in which we are able to prevent f from gaining any knowledge of k depends strongly upon the class of tampering functions. This leads to very different constructions of the key-exchange phase using different technical tools.

Given the common approach described above, it may be tempting to abstract a *non-malleable key-exchange* protocol as a new building block. Intuitively, this would allow us to easily extend our construction to new classes of tampering functions simply by designing a new key exchange protocol for said class. However, (maybe counter-intuitively) it turns out that it is very unclear how this abstraction would work. The class of tampering functions $\mathcal{F}_1$ allowed for the full INMC differs a lot from the class $\mathcal{F}_2$ the key-exchange would need to tolerate. Even worse, it is not clear how $\mathcal{F}_2$ can be generically identified from $\mathcal{F}_1$. Or, the other way round, given a key-exchange that is non-malleable relative to a class $\mathcal{F}_2$, it is not clear against which class of functions the full protocol would then be non-malleable. In fact, our constructions for split-state and for sliding-window show that $\mathcal{F}_1$ can be the result of a complex interplay between the properties of $\mathcal{F}_2$ and the round complexities of both the key-exchange and the original protocol itself.

1.2 Related Works

Non-malleable Codes. To the best of our knowledge, there has been no prior work studying non-malleable codes in the interactive setting. In the non-interactive setting, however, there exists a large body of works studying non-malleable codes for various classes of tampering functions as well as various variants of non-malleable codes. We provide a brief, but non-exhaustive, survey here.

The most well-studied class in the non-interactive setting are split-state tampering functions [60, 35, 3, 25, 24, 2, 26, 59, 57, 58, 4]. But other classes of tampering functions have been studied such as tampering circuits of limited size or depth [42, 10, 23, 11, 8], tampering functions computable by decision trees [12], memory-bounded tampering functions [39] where the size of the available memory is a priori bounded, bounded polynomial time tampering functions [9] 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 [40, 31, 29, 61, 38, 30, 4] and locally decodable and updatable non-malleable codes [33, 21, 32].

While most work on non-malleable codes deals with the information theoretic setting, there has also been recent work [1, 5, 6, 11] in the computational setting. In the computational setting, the work of Chandran et al. [20] on block-wise non-malleable codes may seem as most closely related to our setting; however, there are important differences. Firstly, Chandran et. al do not consider the setting where both parties may have inputs. Instead their notion is similar to the original notion of non-malleable codes where a single fixed message is encoded. Indeed, the entire communication is from the sender to the receiver (rather than running an interactive bi-directional protocol between two parties). Further, their definitions are weaker, as they inherently allow selective aborts whereas our definitions do not suffer from this problem.

Interactive Coding. Starting with the seminal work of Schulmann [64, 65, 66], a large body of works have studied IC schemes for two-party protocols (see, e.g., [18, 47, 15, 43, 50, 49, 54, 37, 45, 17, 44]). Most recently, several works have also studied IC for multiparty protocols [62, 56, 16, 7, 46] in various models.

Secure Computation without Authentication. We also mention a related work of Barak et. al. [13] on secure computation in a setting where the communication channel among the parties may be completely controlled by a polynomial-time adversary. The setting in their work is therefore inherently computational and their techniques rely on using bounded concurrent secure multi-party computation and are unrelated to ours. However, our setting can indeed be seen as being inspired by theirs.

2 Preliminaries

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

Notation. we denote by λ the security parameter. For a distribution D, we denote by $x \leftarrow_\$ D$ the process of sampling a random variable x according to D. By U_ℓ we denote the uniform distribution over $\{0,1\}^\ell$. For a set S, $x \leftarrow_\$ S$ denotes sampling from S uniformly at random. For a pair D_1, D_2 of distributions over a domain X, we denote their statistical distance by

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

If $\mathsf{SD}(D_1, D_2) \leq \epsilon$, we say that D_1, D_2 are ϵ-close. We denote by $\mathsf{replace}$ the function $\mathsf{replace} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ that behaves as follows: If the second input is a singular value s then it replaces any occurrence of same in the first input with s. If the second input is a tuple $(s_1, \ldots, s_n)$ then it replaces any occurrence of same_i in the first input with s_i. We will write $\mathsf{replace}(D, x)$ for some distribution D to denote the distribution defined by sampling $d \leftarrow_\$ D$ and applying $\mathsf{replace}(d, x)$.

Extractors In our constructions we make use of two types of extractors. We first recall the standard notion of strong two-source extractors. Two source extractors were first implicitly introduced by Chor and Goldreich [27]. An argument due to Barak [63] shows that any extractor with a small enough error ϵ is also a strong extractor. This means we can instantiate strong extractors for example with the two-source extractor due to Bourgain [14].

Definition 1 (Strong 2-source Extractor). *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ *is a strong 2-source extractor for sources with min-entropy k and with error ϵ if it satisfies the following property: If X and Y are independent sources of length n with min-entropy k then*

$$\Pr_{y \leftarrow_\$ Y}[\mathsf{SD}(\mathsf{Ext}(X, y), U_m) \geq \epsilon] \leq \epsilon \quad \textit{and} \quad \Pr_{x \leftarrow_\$ X}[\mathsf{SD}(\mathsf{Ext}(x, Y), U_m) \geq \epsilon] \leq \epsilon.$$

Seedless 2-non-malleable extractors were first defined by Cheraghchi and Guruswami [25] but their construction was left as an open problem. The definition was finally instantiated by Chattopadhyay et al. [22]. 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 [22].

Definition 2 (2-non-malleable Extractor). *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ *is a 2-non-malleable extractor for sources with min-entropy k and with error ϵ if it satisfies the following property: If X and 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* $\{0,1\}^m \cup \{\mathsf{same}\}$ *which is independent of sources X and Y, such that*

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

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

Tamper Evident Secret sharing We will define a new notion of tamper evident secret sharing in the following. Such tamper evident secret sharing schemes behave the same as regular secret sharing, except that we are guaranteed that the reconstruction algorithm is able to detect any *detectable tampering* of the shares that would lead to a different reconstructed message and will reject them if they have been tampered with.

Intuitively a tampering is detectable if it meets two criteria: First it must leave at least one of the shares unchanged, since otherwise the shares could simply be replaced by a completely independent sharing, which is trivially undetectable. Second, each tampered share must be independent of at least one of the untampered shares, except for some bounded leakage. This is formally defined in the following.

Definition 3 (n-out-of-n Secret Sharing). *A pair of algorithms* $(\mathsf{Share}, \mathsf{Reconstruct})$ *is a perfectly private, n-out-of-n secret sharing scheme with message space $\{0,1\}^\ell$ and share length ℓ', if all of the following hold.*

1. ***Correctness**: Given all shares, the secret can be reconstructed. I.e., for any secret $m \in \{0,1\}^\ell$, it holds that* $\Pr[\mathsf{Reconstruct}(\mathsf{Share}(m)) = m] = 1$.
2. ***Statistical Privacy**: Given any strict subset of shares, the secret remains perfectly hidden. I.e., for any two secrets $m_0, m_1 \in \{0,1\}^\ell$ and any set of indices $\mathcal{I} \subsetneq \{1,\ldots,n\}$ it holds that for any (computationally unbounded) distinguisher $\mathcal{D}$*

$$\Pr_{\vec{s}\leftarrow\mathsf{Share}(m_0)}[\mathcal{D}((s_i)_{i\in\mathcal{I}}) = 1] = \Pr_{\vec{s}\leftarrow\mathsf{Share}(m_1)}[\mathcal{D}((s_i)_{i\in\mathcal{I}}) = 1]\,.$$

Definition 4 (Detectable Tampering for Secret Sharing). *Let* $(\mathsf{Share}, \mathsf{Reconstruct})$ *be an n-out-of-n Secret Sharing scheme, let $m \in \{0,1\}^\ell$ be a message. A tampering function f for a secret sharing $(s_1,\ldots,s_n)$ of m with ν bits of leakage is described by functions $(f_1,\ldots,f_n)$, sets of indices $\mathcal{I}_1^{\mathsf{in}},\ldots,\mathcal{I}_n^{\mathsf{in}}$ and leakage functions* $(\mathsf{leak}_1,\ldots,\mathsf{leak}_n)$ *such that* $\mathsf{leak}_i : \{0,1\}^* \to \{0,1\}^\nu$ *and*

$$f(s_1,\ldots,s_n) = \Big(f_1\big((s_j)_{j\in\mathcal{I}_1^{\mathsf{in}}}, \mathsf{leak}_1\big((s_j)_{j\notin\mathcal{I}_1^{\mathsf{in}}}\big)\big),\ldots, f_n\big((s_j)_{j\in\mathcal{I}_n^{\mathsf{in}}}, \mathsf{leak}_n\big((s_j)_{j\notin\mathcal{I}_n^{\mathsf{in}}}\big)\big)\Big).$$

For any fixed secret sharing $\vec{s} \leftarrow \mathsf{Share}(m)$ let $\mathcal{M}$ be the set of indices i, such that $s'_i \neq s_i$ for $(s'_1,\ldots,s'_n) := f(s_1,\ldots,s_n)$. A tampering function f is called detectable for $\vec{s}$ if it holds that for all $i \in \mathcal{M}$ we have $\mathcal{M} \cup \mathcal{I}_i^{\mathsf{in}} \subsetneq \{1,\ldots,n\}$. We define the predicate $\mathsf{Dtct}(\vec{s}, f)$ to be 1 iff f is detectable for $\vec{s}$.

This now allows us to formally define tamper evident n-out-of-n secret sharing.

Definition 5 (Tamper Evident n-out-of-n Secret Sharing). *A perfectly private secret sharing scheme* $(\mathsf{Share}, \mathsf{Reconstruct})$ *is said to be $\epsilon(\lambda)$-tamper evident for up to ν bits of leakage if the reconstruction algorithm will reject shares with overwhelming probability if they have been tampered detectably with up to ν bits of leakage. I.e., for all $m \in \{0,1\}^\ell$ and all detectable tampering functions f with ν bits of leakage it holds that*

$$\Pr_{\vec{s}\leftarrow\mathsf{Share}(m)}[\mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \notin \{m, \perp\}] \leq \epsilon(\lambda)$$

We show how to instantiate this notion from XOR-based secret sharing and an information theoretic message authentication code in Appendix A. The concept of tamper evident secret sharing may seem superficially similar to non-malleable secret sharing [51] but the two concepts are in fact incomparable. The guarantee of tamper evident secret sharing is very strong, requiring that the secret cannot be changed except to $\perp$, but only holds against a weak class of tamperings that must leave at least one share unchanged. In contrast, NM-secret sharing provides a weaker guarantee, namely that a tampered secret must be unrelated, but against a stronger class of tampering functions.

3 Definitions

In this section we first formally define interactive protocols and encodings of interactive protocols. We then introduce our notions of non-malleability for encodings of interactive protocols.

3.1 Interactive Protocols

We consider protocols Π between a pair of parties P_0, P_1 (also called Alice and Bob, respectively, for convenience) for evaluating functionalities $g = (g_0, g_1)$ of the form $g_b : X \times Y \to Z$, where X, Y, Z are finite domains. Alice holds an input $x \in X$, and Bob holds $y \in Y$, and the goal of the protocol is to interactively evaluate the functionality, such that at the end of the protocol Alice outputs $g_0(x, y)$ and Bob outputs $g_1(x, y)$. The interactive protocol consists of r rounds, in each of which a single message is sent. Without loss of generality we assume that the parties in Π alternate in sending their messages and that Alice always sends the first message. Formally, an interactive protocol Π between two parties is described by a pair of "next message" functions π_0, π_1 (or π_A, π_B) and a pair of output functions out_A and out_B.The next message function π_A (π_B) takes the input x (y), round number i, and message sequence sent and received by Alice (Bob) so far trans_A (trans_B) and outputs the next message to be sent by Alice (Bob). For simplicity of notation, we assume π_A, π_B always output binary strings. Furthermore, we assume that each message output by π_A, π_B is always of the same length ℓ. The output function out_A (out_B) takes as input x (y) and the final message sequence sent and received by Alice (Bob) trans_A (trans_B) and outputs Alice's (Bob's) protocol output. We denote by $\mathsf{Trans}(x, y)$ the function mapping inputs x, y to the transcript of an honest execution of Π between $A(x)$ and $B(y)$. Note that in this setting we do not explicitly consider probabilistic protocols. However, this is not a limitation, since any probabilistic protocol can be written as a deterministic protocol with additional random tapes given as input to the two parties A and B.

This now allows us to define both correctness of a protocol as well as encodings of interactive protocols.

Definition 6 (Correctness). *A protocol Π, is said to ϵ-correctly evaluate a functionality (g_0, g_1) if it holds that without tampering the output of each party $\mathsf{out}_b(x_b, trans_b) = g_b(x_0, x_1)$ with probability $\geq 1 - \epsilon$.*

Definition 7 (Encoding of an Interactive Protocol). *An encoding Π' of a protocol $\Pi = (A, B)$ is defined by two simulators S_0, S_1 with black-box access to stateful oracles encapsulating the next message functions of A and B respectively. The protocol $\Pi' = (S_0^A, S_1^B)$ is an ϵ-correct encoding of protocol $\Pi = (A, B)$ if for all inputs x, y, $\Pi' = (S_0^{A(x)}, S_1^{B(y)})$ ϵ-correctly evaluates the functionality $(\mathsf{Trans}(x, y), \mathsf{Trans}(x, y))$.*

We note that, given a correct encoding Π' of protocol Π evaluating functionality (g_0, g_1) it is easy to also evaluate (g_0, g_1). To do so, simply run Π' resulting in output $\tau = \mathsf{Trans}(x, y)$ and then evaluate $\mathsf{out}_A(x, \tau)$ and $\mathsf{out}_B(y, \tau)$ respectively. Definition 7 slightly differs from the interactive coding literature [65, 15]. In most of the IC literature, encodings are not defined relative to a stateful oracle, but instead relative to a next-message function oracle as seen in Definition 8.

Definition 8 (Encoding of an Interactive Protocol). *An encoding Π' of a protocol $\Pi = (A, B)$ is defined by two simulators S_0, S_1 with black-box access to an oracle containing the next message function of A and B respectively. The protocol $\Pi' = (S_0^A, S_1^B)$ is an ϵ-correct encoding of protocol $\Pi = (A, B)$ if for all inputs x, y, $\Pi' = (S_0^{\pi_A}(x), S_1^{\pi_B}(y))$ ϵ-correctly evaluates the functionality $(\mathsf{Trans}(x, y), \mathsf{Trans}(x, y))$.*

This difference is significant, because, as observed by Chung et al. [28] in the context of IC, an encoding as defined in the IC literature can leak the parties' inputs under adversarial errors. I.e., security guarantees of Π are not necessarily preserved under Π'. In contrast, under Definition 7, any security guarantee of Π is preserved under Π'. This follows from the fact that the encoding is defined using a pair of simulators with only black-box access to A and B without the ability to know the inputs or rewind the participants of the underlying protocol. Therefore, access to this oracle is equivalent to communicating with an actual instance of A (or B respectively). Any attacker against Π – whether a man in the middle attacker or an attacker acting as either A or B – always has at least black-box access to the two parties. This means she can easily simulate Π' simply by running S_0, S_1 herself. Thus any attack against some arbitrary security property of Π' directly corresponds to an attack against the same property of Π, implying that security guarantees of Π are preserved under Π'.

Protocols under Tampering. It may appear tempting to try and define non-malleability in the interactive setting in the same manner as regular non-malleability by, e.g, considering tampering on the full transcript of the protocol. Split-state tampering for an r-round protocol would then for example mean that an adversary could separately tamper on the first $n/2$ and the second $n/2$ of the protocol messages. However, at least in the synchronous tampering setting we're focusing on such a definition would be very problematic. It would allow an adversary to tamper with the first message depending on future messages, which themselves could depend on the first message, therefore potentially causing an infinite causal loop, even if we allow such "time-travelling" adversaries. So instead we make the reasonable restriction that tampering on each message must happen separately and can only depend on past messages.

We formally describe the process of executing a protocol under tampering with a tampering function $f \in \mathcal{F}$, from some family of tampering functions $\mathcal{F}$. First, empty sequences of sent and received messages $\text{trans}_A = \text{trans}_B = \emptyset$ are initialized. Lets assume that it is Alice's turn to send a message in round i. The next message function π_A is evaluated to compute the next message $m_i := \pi_A(x, i, \text{trans}_A)$. Then m_i is added to Alice's transcript $\text{trans}_A := \text{trans}_A \| m_i$. Next the tampering function is applied to compute the tampered message $m'_i := f(m_1, \ldots, m_i)$ and m'_i is added to $\text{trans}_B := \text{trans}_B \| m'_i$. If it is Bob's turn the execution proceeds identically with reversed roles. Finally the output functions of Alice and Bob are evaluated respectively as $\mathsf{out}_A(x, \text{trans}_A), \mathsf{out}_B(y, \text{trans}_B)$. Note that due to tampering it does not necessarily hold for the sequences of messages $\text{trans}_A = m^A_1, \ldots, m^A_r$ and $\text{trans}_B = m^B_1, \ldots, m^B_r$ that $m^A_i = m^B_i$.

We note that this only models "synchronous" tampering, meaning that the adversary cannot drop or delay messages or desynchronize the two parties by first running the protocol with one party and then the other. This choice is partially inspired by the literature on interactive coding and helps keep our definitions simple. However, cryptographic primitives such as non-malleable commitments have been studied in the setting where there is a non-synchronizing man-in-the-middle adversary. We remark that even in these settings, getting a construction for the synchronous case is often the hardest (for example, there exist general compilers for non-malleable commitments to go from synchronous security to non-synchronous security [67]). We leave the study of more general tampering models for INMCs as an interesting topic for future work.

3.2 Interactive Non-malleable Codes

In the non-interactive setting, non-malleability intuitively means that after tampering the result should be either the original input, or the original input should be completely destroyed, i.e., the output should be independent of the original input. In the interactive setting, there are two different outputs and two different outputs and the question is which output (or pair of outputs) should be independent from which input(s). This leads to an entire space of possible notions, however we settle for the strongest possible – and arguably most natural – notion: In this notion we simply call *protocol-non-malleability*, we require that the output of Alice and Bob respectively are either the correct transcript $\mathsf{Trans}(x, y)$ or $\perp$ and that the product distribution over the two is (almost) completely independent of the two parties' respective inputs x and y. It is very important that the decisions whether to output $\perp$ or not must be made independently of x and y, since otherwise an adversary could potentially force selective aborts and thus learn at least one bit of information about the combined input. This means that protocol-non-malleability not only implies error detection, but is even stronger, since in error detection the output distribution over the real output and $\perp$ is not required to be independent of the inputs.

We note, that weaker definitions may still be meaningful and are not necessarily trivial. In Section 4 we will show that even for a much weaker notion of protocol-non-malleability strong lower bounds exist in the case of threshold tampering functions. We formally define protocol-non-malleability in the following.

Definition 9 (Protocol Non-malleability). *An encoding $\Pi' = (S_0^A, S_1^B)$, of protocol $\Pi = (A, B)$ is ϵ-protocol-non-malleable for a family $\mathcal{F}$ of tampering functions if the following holds: For each tampering function $f \in \mathcal{F}$ there exists a distribution D_f over $\{\perp, \mathsf{same}\}^2$ such that for all x, y, the product distribution of $S_0^{A(x)}$'s and $S_1^{B(y)}$'s outputs is ϵ-close to the distribution $\mathsf{replace}(D_f, \mathsf{Trans}(x, y))$.*

4 Lower Bounds for Threshold Tampering Functions

Threshold tampering functions are classes of tampering functions where the function is only limited in the fraction of the messages they can tamper with. For these classes of tampering functions, lower bounds are known in the case of interactive codes. Specifically Braverman and Rao [18] showed that non-adaptive interactive codes can tolerate tampering with at most 1/4 of the transcript, and Ghaffari, Haeupler, and Sudan [50] showed that an adaptive interactive code can tolerate tampering with at most 2/7 of the transcript. A natural question to ask is whether one can bypass these lower bounds in the case of non-malleable interactive codes. Unfortunately, we show in the following that the known lower bounds for interactive coding translate to identical lower bounds for $\mathsf{negl}(\ell)$-non-malleable interactive coding. In fact, we show that the lower bounds even apply to a much weaker form of protocol-non-malleability, where each party's output by itself (rather than the product distribution of both outputs) only needs to be independent of the *other* party's input.

The basic idea of this lower bound is essentially to show that a non-malleable interactive code is also a regular interactive code. In any encoded protocol, if the output of one party in the underlying protocol depends non-trivially on the other party's input (which should always be the case since otherwise the communication is completely unnecessary) then information theoretically, the transcript must leak this information. If the encoding was not error correcting, then that means that there is a way for a threshold tampering function to cause at least one of the parties to abort. Since the tampering function is unlimited in it's knowledge of the transcript, it can extract the information about one of the parties' input and depending on the function of the input thus revealed either cause the abort or not. This would be an input dependent abort which clearly means that the encoding is not non-malleable.

However, this straightforward approach does not work. The reason is, that the information about the input might only be revealed in say the ith message of protocol, while the threshold tampering function requires tampering with earlier messages to cause the abort. But there is a way around this problem. If we can cleanly define which message in the protocol is the first message that reveals information about the input, then we can construct another INMC in the CRS model, where all previous messages are pushed into the CRS. This is possible since those messages are "almost" independent of the actual input and it is possible for the INMC to (inefficiently) sample a consistent internal state, once it gets the input. This means that now the information about the input is revealed in the very first protocol message and thus the approach described above works.

For the lower bound to translate to INMC, we therefore need that the lower bounds for IC apply also to inefficient interactive encodings in the CRS model. Luckily, this follows easily from the structure of the results in [18] and [50]. We discuss the application of the bounds to the CRS model in a bit more detail in Appendix B.

As mentioned above, we can in fact show this lower bound for a much weaker form of non-malleability we formally define in the following.

Definition 10 (Weak Protocol Non-malleability). *An encoding $\Pi' = (S_0^A, S_1^B)$, of protocol $\Pi = (A, B)$ is ϵ-weakly-protocol-non-malleable for a family $\mathcal{F}$ of tampering functions if the following holds: For each tampering function $f \in \mathcal{F}$ and for each x (resp. y) there exists a distribution $D_{f,x}^A$ (resp. $D_{f,y}^B$) over*

$\{\bot, \mathsf{same}\} \cup \{0,1\}^n$ *such that for all y (resp. x), the output distribution of $S_0^{A(x)}$ (resp. $S_1^{B(y)}$) is ϵ-close to the distribution* $\mathsf{replace}(D^A_{f,x}, \mathsf{Trans}(x,y))$ *(resp.* $\mathsf{replace}(D^B_{f,y}, \mathsf{Trans}(x,y))$*).*

It is easy to see, that this notion is strictly weaker than protocol-non-malleability as defined in Definition 9. If a distribution D_f as required by Definition 9 exists, then $D^A_{f,x}$ and $D^B_{f,y}$ can easily be sampled by sampling from D_f and throwing away half of the output. On the other hand, since $D^A_{f,x}$ can depend on x, it does not help in sampling a distribution D_f that is required to be (almost) independent of x.

Theorem 1. *Let $\Pi = (A, B)$ be an r-round protocol with inputs $x, y \in \{0,1\}^\ell$ such that there exists at least one triple of inputs (x_1^*, x_2^*, y^*) or (x^*, y_1^*, y_2^*) such that $\mathsf{Trans}(x_1^*, y^*) \neq \mathsf{Trans}(x_2^*, y^*)$ or $\mathsf{Trans}(x^*, y_1^*) \neq \mathsf{Trans}(x^*, y_2^*)$ respectively. Let Π' be an $\delta(\ell)$-correct, $\mathsf{negl}(\ell)$-weakly-protocol-nonmalleable INMC for protocol Π for a family $\mathcal{F}$ of threshold tampering functions. Then there also exists an (computationally unbounded) interactive code $\overline{\Pi}$ in the CRS model for the same protocol Π and the same family of threshold tampering functions $\mathcal{F}$.*

Proof of Theorem 1 By assumption there exists at least one triple of inputs (x_0^*, x_1^*, y^*) or (x^*, y_0^*, y_1^*) such that $\mathsf{Trans}(x_0^*, y^*) \neq \mathsf{Trans}(x_1^*, y^*)$ or $\mathsf{Trans}(x^*, y_0^*) \neq \mathsf{Trans}(x^*, y_1^*)$ respectively. This implies that the full transcript of Π' must information theoretically reveal a noticeable amount of information about the inputs of at least one party. Therefore, at least one of the following must hold:

1. There exists a round i and a pair of Alice's inputs x_0^*, x_1^* such that for Bob's input y^*, uniformly sampled input $x \leftarrow\!\!\$\, \{0,1\}^\ell$, and a partial transcript $m_1, \ldots, m_i \leftarrow \langle S_1^{A(x)}, S_2^{B(y^*)} \rangle$ it holds that
$$|\Pr[x = x_0^* \,|\, x \in \{x_0^*, x_1^*\}] - \Pr[x = x_1^* \,|\, x \in \{x_0^*, x_1^*\}]| \geq \frac{1}{\mathsf{poly}(\ell)} \tag{1}$$
while for all previous rounds $j < i$ and all pairs of inputs for Alice x_0, x_1, all inputs y for Bob, a uniformly sampled $x \leftarrow\!\!\$\, \{0,1\}^\ell$, and a partial transcript $m_1, \ldots, m_j \leftarrow \langle S_1^{A(x)}, S_2^{B(y)} \rangle$ it holds that
$$|\Pr[x = x_0 \,|\, x \in \{x_0, x_1\}] - \Pr[x = x_1 \,|\, x \in \{x_0, x_1\}]| < \mathsf{negl}(\ell) \tag{2}$$
2. There exists a round k and a pair of Bob's inputs y_0^*, y_1^* such that for Alice's input x^*, uniformly sampled input $y \leftarrow\!\!\$\, \{0,1\}^\ell$, and a partial transcript $m_1, \ldots, m_i \leftarrow \langle S_1^{A(x^*)}, S_2^{B(y)} \rangle$ it holds that
$$|\Pr[y = y_0^* \,|\, y \in \{y_0^*, y_1^*\}] - \Pr[y = y_1^* \,|\, y \in \{y_0^*, y_1^*\}]| \geq \frac{1}{\mathsf{poly}(\ell)} \tag{3}$$
while for all previous rounds $j < k$ and all pairs of inputs for Bob y_0, y_1, all inputs x for Alice, a uniformly sampled $y \leftarrow\!\!\$\, \{0,1\}^\ell$, and a partial transcript $m_1, \ldots, m_j \leftarrow \langle S_1^{A(x)}, S_2^{B(y)} \rangle$ it holds that
$$|\Pr[y = y_0 \,|\, y \in \{y_0, y_1\}] - \Pr[y = y_1 \,|\, y \in \{y_0, y_1\}]| < \mathsf{negl}(\ell) \tag{4}$$

If a round i does not exist such that Equation 1 holds, then Equation 2 holds for all rounds $j \leq r$ and we define $i = r + 1$. Respectively, if a round k does not exist such that Equation 3 holds, then Equation 4 holds for all rounds $j \leq r$ and we define $k = r + 1$.

We use this property to construct a (computationally unbounded) INMC $\overline{\Pi}$ in the CRS model for the same protocol Π and the same family of tampering function $\mathcal{F}$ as follows: The CRS is computed by sampling random $x', y' \leftarrow\!\!\$\, \{0,1\}^\ell$ and computing a partial transcript $m_1, \ldots, m_{\min(i,k)-1} \leftarrow \langle S_1^{A(x')}, S_2^{B(y')} \rangle$. We then set $\mathsf{crs} = (m_1, \ldots, m_{\min(i,k)-1})$. We denote by $\mathsf{crs}(x', y')$ a crs computed based on inputs x' and y'.

The execution of $\overline{\Pi}$ works as follows: $\bar{S_1}^{A(x)}(\mathsf{crs})$ and $\bar{S_2}^{B(y)}(\mathsf{crs})$ uniformly sample – using their unbounded computational power – an internal state of S_1 and S_2 respectively consistent with x (respectively y) and the partial transcript present in the crs. Such a sampling will always be successful for $\mathsf{crs}(x, y)$. Further, Equation 2 and Equation 4 imply that for any pair of inputs x', y' the statistical distance $\mathsf{SD}(\mathsf{crs}(x, y), \mathsf{crs}(x', y'))$ is negligible. Therefore, for a CRS computed with uniformly sampled x', y' this sampling will be successful with probability $1 - \mathsf{negl}(\ell)$. Proceeding from this internal state, $\bar{S}_1^{A(x)}(\mathsf{crs})$ and $\bar{S}_2^{B(y)}(\mathsf{crs})$ then simply execute the original INMC $\mathit{\Pi}'$. Thus $\overline{\mathit{\Pi}}$ is $\delta + \mathsf{negl}(\ell)$ correct. Further, $\overline{\mathit{\Pi}}$ remains $\mathsf{negl}(\ell)$ non-malleable. This is clear since any valid threshold tampering function on $\overline{\mathit{\Pi}}$ would in particular be a valid threshold tampering function on $\mathit{\Pi}'$.

We will now argue that this implies that $\overline{\mathit{\Pi}}$ is also an $\delta + \mathsf{negl}(\ell)$-correct interactive code, i.e. provides error correction. Assume towards contradiction that this were not the case. I.e., assume there exists a threshold tampering function f that when applied successively to the messages exchanged in $\overline{\mathit{\Pi}}$ causes at least one of the parties to abort. Then we construct a threshold tampering function causing a selective abort as follows: The tampering function $f' = (f'_1, \ldots, f'_r)$ takes as input the CRS crs as well as the transcript so far and computes the probabilities $P_0 = \Pr[x = x_0^*]$ and $P_1 \Pr[x = x_1^*]$ conditioned on the transcript contained in the crs and the first message of the protocol and is then defined as

$$f'_i(\mathsf{crs}, m_1, \ldots, m_i) \begin{cases} m_i & \text{if } P_0 \geq P_1 \\ f_i(\mathsf{crs}, m_1, \ldots, m_i) & \text{otherwise} \end{cases}.$$

It remains to show that this causes an output distribution that cannot be approximated without knowledge of x, y. We denote by $\mathsf{Abort}(x, y, \mathsf{crs}(x', y'))$ the event that at least one of the parties aborts and outputs $\perp$ in a protocol execution of $\mathit{\Pi}'$ with inputs x and y and crs $\mathsf{crs}(x', y')$ that is being tampered by f'. If $i \leq k$ then it holds that

$$\left| \begin{array}{l} \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x_0^*, y^*, \mathsf{crs}(x, y))\right] \\ - \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x_1^*, y^*, \mathsf{crs}(x, y))\right] \end{array} \right| \tag{5}$$

$$\geq \left| \begin{array}{l} \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x_0^*, y^*, \mathsf{crs}(x, y)) \,\middle|\, x \in \{x_0^*, x_1^*\}\right] \\ - \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x_1^*, y^*, \mathsf{crs}(x, y)) \,\middle|\, x \in \{x_0^*, x_1^*\}\right] \end{array} \right| - \mathsf{negl}(\ell) \tag{6}$$

$$\geq \frac{1}{\mathsf{poly}(\ell)} - \mathsf{negl}(\ell) \tag{7}$$

where Equation 6 follows from Equation 2. As noted before, Equation 2 implies that the statistical distance between $\mathsf{crs}(x, y)$ and $\mathsf{crs}(x', y)$ is negligible and therefore Equation 6 must follow. Finally Equation 7 follows immediately from Equation 1. Since Equation 7 implies a statistical distance of $1/\mathsf{poly}(\ell)$ between the output distributions of $\mathit{\Pi}'$ with inputs x_0^*, y^* and x_1^*, y^*, clearly there cannot exist a distribution sampled independently of the inputs that has negligible statistical distance from both output distributions. Therefore $\mathit{\Pi}'$ cannot be $\mathsf{negl}(\ell)$-nonmalleable. Likewise, if $k < i$ then we have that

$$\left| \begin{array}{l} \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x^*, y_0^*, \mathsf{crs}(x, y))\right] \\ - \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x^*, y_1^*, \mathsf{crs}(x, y))\right] \end{array} \right| \tag{8}$$

$$\geq \left| \begin{array}{l} \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x^*, y_0^*, \mathsf{crs}(x, y)) \,\middle|\, x \in \{y_0^*, y_1^*\}\right] \\ - \Pr\left[x, y \leftarrow \{0,1\}^\ell : \mathsf{Abort}(x^*, y_1^*, \mathsf{crs}(x, y)) \,\middle|\, x \in \{y_0^*, y_1^*\}\right] \end{array} \right| - \mathsf{negl}(\ell) \tag{9}$$

$$\geq \frac{1}{\mathsf{poly}(\ell)} - \mathsf{negl}(\ell) \tag{10}$$

which follows equivalently from Equation 4 and Equation 3 and also implies that $\mathit{\Pi}'$ cannot be $\mathsf{negl}(\ell)$-nonmalleable. □

Applying the Lower Bound to Other Tampering Functions It is natural to ask whether the lower bound stated above also applies to other classes of functions. This would be unfortunate, since it would trivially rule out INMCs for most classes of tampering functions. However, fortunately, this is not the case.

In the proof of Theorem 1, we explicitly use that the tampering function at any point has complete knowledge of the full transcript so far and is completely unbounded in the resources necessary to compute the tampering. It then follows that if the transcript information theoretically reveals *anything* about the inputs, then the tampering function can extract this information and cause a conditional abort, thus allowing for the proof to go through. In each of the classes of tampering functions we consider in the following sections, however, the tampering functions are restricted in one way or another in its view of the full transcript. This means that the proof no longer applies, since even when the full transcript contains information about the inputs, the tampering function is no longer capable of extracting it.

In fact, we explicitly exploit this observation in each of our protocols. Our protocols consist of an initial input-independent phase, where key material is established. This phase is constructed in such a way that in any future round, the established key material will be almost uniform from the point of view of the tampering function. Using information theoretically secure encryption and authentication we can then execute the underlying protocol in such a way that the transcript of that execution is remains independent of the input *from the point of view of the tampering function.*

5 Bounded State Tampering

The first class of tampering functions we consider are tampering functions with bounded state. This is a very natural model in which adversaries are assumed to be arbitrarily powerful, but there exists an a priori upper bound on the size of the state they can hold. Similar adversaries have been considered before in many settings, starting with the work by Cachin and Maurer [19] on encryption and key exchange secure against computationally unbounded adversaries. Recently, in related work, Faust et al. [39] studied non-malleable codes against space-bounded tampering. However, the notion of bounded state tampering we introduce in this section is stronger than one would expect from naïvely extending the notion to interactive non-malleable codes. In particular we do not limit the size of the memory available for computing the tampering function. Instead, a limit is only placed on the size of the state that can be carried over from tampering one message to tampering with the next. I.e., the idea is, that an adversary in this model can iteratively tamper with each message depending on some function of *all* previous messages, *but* the size of this information is limited to some fixed number of bits s. We formally define this in terms of a tampering function in the following.

Definition 11 (Bounded State Tampering Functions). *Functions of the class of s-bounded state tampering functions $\mathcal{F}^s_{bounded}$ for an r-round interactive protocols are defined by an r-tuple of pairs of functions $((g_1, h_1), \ldots, (g_r, h_r))$ where the range of the functions h_i is $\{0,1\}^s$. Let $m_1, \ldots, m_i$ be the messages sent by the participants of the protocol in a partial execution.The tampering function for the ith message is then defined as*

$$f_i(m_1, \ldots, m_i) := g_i\big(m_i, h_{i-1}\big(m_{i-1}, h_{i-2}(m_{i-2}, \ldots)\big)\big).$$

5.1 Interactive Non-Malleable Code for Bounded State Tampering

We devise a generic protocol-non-malleable encoding Π for bounded state tampering for any two-party protocol Π_0. The basic idea is to first run a key exchange phase in which Alice and Bob exchange enough key material that they can execute the original protocol encrypted under one-time pad and authenticated with information theoretically secure MACs. The main challenge is to craft the key-exchange phase in such a way, that the adversary's limitations, i.e., having bounded state, preclude her from both, learning any meaningful information about the exchanged key material, as well as influencing the key material in a meaningful way. For bounded state tampering functions, we achieve this using 2-non-malleable extractors. The idea behind this is that each party chooses two random sources that are significantly longer than the size of the bounded state and sends it to the other party. Both parties then apply a 2-non-malleable extractor to each pair of sources and thus extract a key they can use to secure the following communication using information theoretic authenticated encryption. A tampering function with bounded state will not be able to "remember" enough information about the two sources to predict the exchanged key with a any significant probability and thus will not be able to change the authenticated ciphertexts without being caught. Formally this is stated in the following theorem.

Algorithm 1: Protocol Π against bounded state tampering functions

We compile Π_0 into Π below. Let Ext and Π_0 be as in Theorem 2. The communication proceeds in three phases, a key exchange phase, a key confirmation phase and a protocol execution phase. All messages in the following protocol have a fixed length. Whenever a party in the protocol aborts, she outputs $\perp$ instead of a transcript.

Key Exchange Phase: Alice chooses two strings α_1, α_2 and Bob chooses two strings β_1, β_2 all of length n. The two parties then alternatingly send the two strings.

1. First Alice then sends α_1, then Bob sends β_1, Alice sends α_2, and Bob finally sends β_2.
2. Both parties use the extractor to extract $k_1 := \mathsf{Ext}(\alpha_1, \alpha_2)$ and $k_2 := \mathsf{Ext}(\beta_1, \beta_2)$ and set $k := k_1 \oplus k_2$. They then split $k = k_A \| k_B \| k_1^{\mathsf{auth}} \| k_1^{\mathsf{enc}} \| \ldots \| k_r^{\mathsf{auth}} \| k_r^{\mathsf{enc}}$ into substrings, where $|k_A| = |k_B| = \left|k_i^{\mathsf{auth}}\right| = 2\lambda$ and $|k_i^{\mathsf{enc}}| = \ell$.

Key Confirmation Phase: Alice and Bob verify that they agree on the exchanged key.

1. Bob chooses a random challenge $c_B \leftarrow\$ \{0,1\}^\lambda$ and sends it to Alice.
2. Alice computes $t_B := \mathsf{MAC}(k_B, c_B)$, chooses a challenge $c_A \leftarrow\$ \{0,1\}^\lambda$, and sends t_B, c_A to Bob.
3. If $\mathsf{Vf}(k_B, c_B, t_B) = 1$, then Bob sends $t_A := \mathsf{MAC}(k_A, c_A)$ to Alice. Otherwise he aborts.
4. If $\mathsf{Vf}(k_A, c_A, t_A) = 1$ then Alice proceeds to the next phase. Otherwise she aborts.

Protocol Execution Phase: Both parties initialize their view of the underlying protocol as an empty list $\mathrm{trans}_A = \emptyset$ and $\mathrm{trans}_B = \emptyset$. Starting with Alice's first message Alice and Bob proceed as follows for each message:

1. In the ith round, if it is Alice's (resp. Bob's) turn to send a message she invokes the next-message function of the underlying protocol $m_i := \pi_A^0(i, x, \mathrm{trans}_A)$ (resp. $m_i := \pi_B^0(i, y, \mathrm{trans}_B)$) and adds the message to her view $\mathrm{trans}_A := \mathrm{trans}_A \| m_i$ (resp. $\mathrm{trans}_B := \mathrm{trans}_B \| m_i$).
2. Next the party computes the one-time pad encryption $c_i := m_i \oplus k_i^{\mathsf{enc}}$ of m_i as well as an authentication tag $t_i := \mathsf{MAC}(k_i^{\mathsf{auth}}, c_i)$ and sends c_i, t_i to the other party.
3. If the authentication tag verifies, i.e., $\mathsf{Vf}(k_i^{\mathsf{auth}}, c_i, t_i) = 1$ the other party decrypts $m_i := c_i \oplus k_i^{\mathsf{enc}}$ and adds the message to their view, i.e., $\mathrm{trans}_A := \mathrm{trans}_A \| m_i$ or $\mathrm{trans}_B := \mathrm{trans}_B \| m_i$.
4. Finally the underlying protocol terminates and both parties output their respective transcripts trans_A or trans_B or $\perp$ if they aborted at any point during the protocol.

Theorem 2. *Let Π_0 denote a correct, r-round protocol, with length-ℓ messages. We assume wlog that Alice sends both the first and last message in Π_0 Let $s \in \mathbb{N}$ be any bound as defined in Definition 11. Let λ' be the target security parameter, then we set $\lambda = \max(\ell, \lambda')$. Let $\mathsf{MAC} : \{0,1\}^{2\lambda} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a $2^{-\lambda}$-secure information theoretic message authentication code. Let $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{r\ell+(2r+4)\lambda}$ be a 2-non-malleable extractor for sources with min-entropy $n - (s + \lambda)$ and with error ϵ. Then there exists a $r + 7$-round encoding Π of Π_0 that is $5\epsilon + 4 \cdot 2^{-\lambda}$-protocol-non-malleable against $\mathcal{F}^s_{bounded}$.*

Note that the required extractor can be instantiated using the construction of Chattopadhyay et al. [22], while the MAC can be instantiated with a family of pair-wise independent hash functions.

Proof of Theorem 2. The protocol Π is specified in Algorithm 1. We need to argue that the protocol is correct and protocol-non-malleable.

Correctness: The correctness of Π follows from the fact that the extractor is deterministic and the message authentication code is correct. Since the extractor is deterministic, both parties will extract the same string k. The correctness of the message authentication code then implies neither party will ever abort during the protocol. Further, since the one-time pad is correct it follows that messages of the underlying protocol will always be decrypted correctly and thus both parties are faithfully executing an honest instance of Π_0. Thus at the end of the protocol the collected transcripts correspond to an honest execution of Π_0.

Protocol-non-malleability: Let f be an s-bounded state tampering function described by $((g_1, h_1), \ldots, (g_r, h_r))$. To prove that the coding scheme is protocol-non-malleable, we need to prove that a distribution D_f as in Definition 9 exist.

The distribution D_f When sampling from D_f we need to deal with the problem that in addition to the s bits of state f can keep by design, it can learn additional information by making use of conditional aborts. I.e., in round i the function g_i can force an abort in the protocol unless the message sent in round i is "good". In any future round $j > i$, even if it's s bit state does not retain any information about m_i the function g_j therefore "remembers" that m_i must have been "good", since otherwise the protocol would have aborted.

Algorithm 2: Sampler of distribution D_f for Algorithm 1

```
1. Sample four strings $\alpha_1, \alpha_2, \beta_1, \beta_2 \leftarrow_\$ \{0,1\}^n$.
2. Apply the tampering function to the messages as $\alpha_1' := f_1(\alpha_1)$, $\beta_1' := f_2(\alpha_1, \beta_1)$, $\alpha_2' := f_3(\alpha_1, \beta_1, \alpha_2)$,
   $\alpha_2' := f_4(\alpha_1, \beta_1, \alpha_2, \beta_2)$ and extract $k_1 := \mathsf{Ext}(\alpha_1, \alpha_2)$ and $k_2 := \mathsf{Ext}(\beta_1, \beta_2)$ as well as $k_1' := \mathsf{Ext}(\alpha_1', \alpha_2')$ and
   $k_2' := \mathsf{Ext}(\beta_1', \beta_2')$. Set $k := k_1 \oplus k_2'$ and $k' := k_1' \oplus k_2$.
3. If $k' \neq k$ output $(\perp, \perp)$ and stop.
4. If $k' = k$, then simulate a protocol execution tampered with $f$ as follows
   (a) Replace all messages with random strings of appropriate length and apply the tampering function to those
       messages.
   (b) If for any index $7 < i < r + 7$ it holds that $m_i \neq f_i(m_1, \ldots, m_i)$ output $(\perp, \perp)$ and stop.
   (c) If it holds that $m_{r+7} \neq f_{r+7}(m_1, \ldots, m_{r+7})$ output $(\mathsf{same}, \perp)$ and stop.
5. If the simulated interaction completed successfully, output $(\mathsf{same}, \mathsf{same})$.
```

Technically the tampering function can use conditional aborts to leak an arbitrary amount of information. However, this comes at the expense of having to abort with high probability. Let $1 - \delta(\lambda)$ be the probability of f causing either party to abort *before* the last message in the protocol is sent. Then this allows the tampering function to leak at most $\log \delta^{-1}(\lambda)$ additional bits to future rounds. Note that causing an abort by tampering with the very last message cannot add any additional leakage, since there are no more future rounds to consider. Further note, that either party aborting *before* the last message is sent automatically causes both parties to output $\perp$ in the synchronized setting.

We use the above observation to sample from D_f by sampling differently depending on $\delta(\lambda)$. If $\delta(\lambda) \leq 2^{-\lambda}$, the distribution D_f is sampled by simply outputting $(\perp, \perp)$. Clearly this distribution is $2^{-\lambda}$ close to the real distribution, since f causes both Alice and Bob to abort and output $\perp$ with probability at least $1 - 2^{-\lambda}$. If $\delta > 2^{-\lambda}$, the distribution D_f is sampled as shown in Algorithm 2. The difference between D_f and the real tampered transcript distribution is captured by the event in which the sampler aborts the execution in steps 4b or 4c, but the real execution continues. To see why D_f is close to the tampered transcript distribution, consider the four cases.

1 The tampering function did not change (α_1, α_2) **or** (β_1, β_2): This is the simplest case. Note that the tampering function may store a bounded function of the messages seen so far. That is, the tampering function stores $\gamma = h_4(\beta_2, h_3(\alpha_2, h_2(\beta_1, h_1(\alpha_1))))$ where h_i denotes a memory bounded function as described above. We claim that given γ and up to $\log \delta^{-1}(\lambda) = \lambda$ many bits of additional leakage due to conditional aborts, (k_1, k_2) and hence k is 2ϵ-close to uniform. This follows from the property of strong extractors. Conditioned on γ and the leakage, the sources (α_1, α_2) are still independent and have sufficient min-entropy. This may not be immediately apparent, since future tampering can depend on γ, which technically constitutes joint leakage over (α_1, α_2). However, we can see that this particular joint leakage is not an issue for a 2-nonmalleable extractor by switching to a different but equivalent viewpoint. If we fix $h_1(\alpha_1)$, then α_1 is no longer uniformly distributed but it is still a source with a distribution with at least $n - s$ bits of min-entropy. This is ensured by the fixed upper bound on the size of the leakage. From this viewpoint, since $h_1(\alpha_1)$ is fixed, γ is no longer joint leakage over (α_1, α_2) but merely bounded leakage over α_2. The same applies to additional potential leakage due to conditional aborts, leaving us with a source α_1 with at least $n - (s + \lambda)$ bits of min-entropy. Similarly, the same holds for sources (β_1, β_2).

Now it follows that if the tampering function changes any message in the protocol execution phase, the MAC verification will fail (up to the error $2^{-\lambda}$) causing the receiving party to abort. Unless the tampered message was the one sent in round $r + 7$ this in turn automatically causes the other party to abort as well (corresponding to step 4b). If the tampered message was the one sent in round $r + 7$ then only Bob would abort (corresponding to step 4c). Furthermore, by the property of one-time pads, the probability of the tampering function changing any message is independent of the message itself.

2 The tampering function changed (α_1, α_2) (i.e., changed at least one of them) but not (β_1, β_2): We claim that $k_1 := \mathsf{Ext}(\alpha_1, \alpha_2)$ is ϵ-close to uniform given γ and up to λ many bits of additional leakage due to conditional aborts, $k_1' := \mathsf{Ext}(\alpha_1', \alpha_2')$, and (β_1, β_2). This follows from the fact that k_1 is ϵ-close to uniform given k_1', γ and λ bits of leakage (by the property of 2-non-malleable extractors), and, that (β_1, β_2) are independent of (α_1, α_2). This also implies that k_1 is ϵ-close to uniform given $\gamma, k_1', (\beta_1, \beta_2)$, k_2, and λ bits of leakage since k_2 is entirely determined by (β_1, β_2). This in turn implies that k_1 is ϵ-close to uniform given $\gamma, k'_1, (\beta_1, \beta_2), k_2, k'_2$, and λ bits of leakage since $k'_2 = k_2$. This implies that $k = k_1 \oplus k'_2$ is ϵ-close to uniform conditioned on $\gamma, k'_1, (\beta_1, \beta_2), k_2$ and leakage. This finally implies that k is ϵ-close to uniform conditioned on $\gamma, k' = k'_1 \oplus k_2$ and leakage. Thus, the MAC verification will fail for Alice in the key confirmation phase (up to the error $2^{-\lambda}$) causing both parties to output $\perp$.

3 The tampering function changed (β_1, β_2) but not (α_1, α_2): This case is symmetric to the previous case.

4 The tampering function changed both (α_1, α_2) **and** (β_1, β_2): The only difference between this case and case 2 is that now k'_2 may not be equal to k_2. As in the previous case, k_1 is almost uniform given $\gamma, k'_1, (\beta_1, \beta_2), k_2$ and leakage. But note that k'_2 is entirely determined by (β_1, β_2), γ and the (fixed) tampering function. Hence, k_1 is almost uniform given $\gamma, k'_1, (\beta_1, \beta_2), k_2, k'_2$ and leakage.

Overall using a union bound over the errors of the extractor and the MAC, we get an upper bound on the statistical distance between D_f and the outputs of a real execution of $5\epsilon + 4 \cdot 2^{-\lambda}$. □

6 Split-State Tampering

Split-state tampering functions are one of the most interesting and well studied families of tampering functions for regular non-malleable codes and were already considered by Dziembowski, Pietrzak, and Wichs [36] in their seminal paper. A 2-split-state tampering function independently tampers on two fixed disjoint parts of a codeword. Transferring this idea to the interactive setting is straightforward. We can divide the transcript of a protocol into two disjoint sets of messages and allow the tampering function to tamper independently on those two sets.

However, we are actually able to achieve protocol non-malleability for a stronger class, namely c-unbalanced split-state tampering functions. In the regular split state setting, the encoding scheme determines the "split". In contrast, a c-unbalanced split-state tampering function can split the transcript into two arbitrary sets, as long as each set contains at least a $1/c$ fraction of the messages.

Definition 12 (c-Unbalanced Split-State Tampering Functions). *Functions of the class of c-unbalanced 2-split-state tampering functions $\mathcal{F}^c_{strong\text{-}split}$ for an r-round interactive protocols are defined by an r-tuple of functions $(g_1, \ldots, g_r)$ and two disjoint sets $\mathcal{I}_0, \mathcal{I}_1$ such that $\min(|\mathcal{I}_0|, |\mathcal{I}_1|) \geq r/c$ and $\mathcal{I}_0 \cup \mathcal{I}_1 = \{1, \ldots, r\}$. Let $m_1, \ldots, m_i$ denote the messages sent by the participants of the protocol in a partial execution. The tampering function for message m_i is then*

$$f_i(m_1, \ldots, m_i) := \begin{cases} g_i((m_j)_{j \in \mathcal{I}_0, j \leq i}) & \text{if } i \in \mathcal{I}_0 \\ g_i((m_j)_{j \in \mathcal{I}_1, j \leq i}) & \text{if } i \in \mathcal{I}_1 \end{cases}$$

As a special case functions in $\mathcal{F}^2_{\text{strong-split}}$ must split the messages into two equal size sets. These functions are also alternatively simply called split-state tampering functions, since the split is not unbalanced.

6.1 INMC for Split-State Tampering

We devise a generic protocol-non-malleable encoding $\mathit{\Pi}$ for c-unbalanced split-state tampering functions for any two-party protocol $\mathit{\Pi}_0$. The basic idea of the encoding will seem similar to the protocol for bounded state tampering functions, however the instantiation is quite different. We again first run a key exchange phase in which enough key material is exchanged to execute the original protocol encrypted under one-time pad and authenticate all messages with information theoretically secure MACs. The main difference is in the implementation of the key exchange phase. Unlike before, where we relied on non-malleable extractors, we use a notion of tamper-evident n-out-of-n secret sharing in this case. The idea behind this is that both parties contribute to the key material $k = \mathsf{Ext}(k_1, k_2)$ and share their part of the key-material into many shares that are sent in separate messages. If we are able to enforce that the tampering function must jointly tamper with almost all of the messages in the key-exchange phase to be able to predict the key with any significant probability, then we can scale the key exchange phase to make sure that such a function would not be c-unbalanced. The tamper-evidence of the secret sharing scheme allows us to ensure that either party's shares must be tampered with jointly to learn anything about the reconstructed secret. However, this is not enough. We must also ensure that the *other party's* messages must also be tampered jointly. We achieve this via a use of MACs with "successively revealed keys." I.e., each message must be authenticated using a key that is only revealed if one has knowledge of *all* of the other party's previous messages. In this way, each message is "chained" to the other party's previous messages and any successful tampering must necessarily tamper with the full key-exchange phase in a joint manner.

Theorem 3. *Let Π_0 denote a correct, r-round protocol, with length-ℓ messages. Let $(\mathsf{Share}, \mathsf{Reconstruct})$ be a $\lceil((c-1)(r+5)+1)/2\rceil$-out-of-$\lceil((c-1)(r+5)+1)/2\rceil$ perfectly private, ϵ'-tamper evident secret sharing scheme for up to $\lambda/2$ bits of leakage with message length ℓ'' and share length ℓ' Let λ' be the target security parameter, then we set $\lambda = \max(\ell, \ell', \lambda')$. Let $\mathsf{MAC} : \{0,1\}^{2\lambda} \times \{0,1\}^{\lambda}$ be a $2^{-\lambda}$-secure information theoretic message authentication code. Let $\mathsf{Ext} : \{0,1\}^{\ell''} \times \{0,1\}^{\ell''} \to \{0,1\}^{r\ell+(2r+4)\lambda}$ be a strong two-source extractor for sources with min-entropy $\ell'' - \lambda/2$ with error ϵ''. We assume without loss of generality that Alice sends both the first and last message in Π_0 Then for any c there exists a $c(r+5)$-round encoding Π of Π_0 that is $\epsilon(\lambda) = 2\epsilon' + 3\epsilon'' + (c-1)(r+5)+3) \cdot 2^{-\lambda/2} + 2^{-\lambda+1}$-non-malleable against $\mathcal{F}^{c}_{strong\text{-}split}$.*

The tamper evident secret sharing scheme can be instantiated using the construction described in Appendix A, the MAC can be instantiated with a family of pairwise-independent hash functions and the strong 2-source extractor can be instantiated with the extractor due to Bourgain [14].

Proof of Theorem 3 The protocol Π is specified in Algorithm 3. We need to argue that the protocol is correct and protocol-non-malleable.

Correctness: The correctness of Π follows from the correctness of the secret sharing scheme and the message authentication code. The correctness of the secret sharing scheme implies that when no tampering takes place, Bob and Alice will both reconstruct the correct string k_1 or k_2 respectively. Thus, they will compute the same key k. Combined with the correctness of the message authentication code, this means that neither party will ever abort during the protocol. Further, since the one-time pad is correct it follows that messages of the underlying protocol will always be decrypted correctly and thus both parties are faithfully executing an honest instance of Π_0. Thus at the end of the protocol the collected transcripts correspond to an honest execution of Π_0.

Protocol Non-Malleability: Let f be a c-unbalanced split state tampering function described by $(g_1, \ldots, g_{c(r+5)})$ and $\mathcal{I}_0, \mathcal{I}_1$ (refer to Definition 12). To prove that the coding scheme is protocol-non-malleable, we show that a distributions D_f as in Definition 9 exists.

The distribution D_f: When sampling from D_f we again need to deal with the problem that the tampering function can communicate information through conditional aborts. I.e., in round i with $i \in \mathcal{I}_b$, the function g_i can force an abort in the protocol unless the message sent in round i is "good". In any future round $j > i$, even if $j \in \mathcal{I}_{1-b}$ the function g_j therefore has the information that the message in round i must have been "good". This implies leakage between the two split states. To deal with this problem we sample differently depending on the probability of f causing an abort during a protocol execution. Let $1 - \delta(\lambda)$ be the probability of f causing either party to abort *before* the last message in the protocol is sent. If $\delta(\lambda) \leq 2^{-\lambda/2}$, the distribution D_f is sampled by simply outputting $(\perp, \perp)$. Clearly this distribution is $2^{-\lambda/2} \leq \epsilon(n)$ close to the real distribution, since f causes both parties to abort and output $\perp$ with probability at least $1 - 2^{-\lambda/2}$. If $\delta > 2^{-\lambda/2}$, the distribution D_f is sampled as shown in Algorithm 4.

Analysis. It remains to show that D_f is $2\epsilon' + 3\epsilon'' + (c-1)(r+5)+3) \cdot 2^{-\lambda/2} + 2^{-\lambda+1}$ close to the tampered transcript distribution. We first note that the protocol Π overall has $((c-1)(r+5)+1)+r+4 = c(r+5)$ rounds, of which $(c-1)(r+5)+2$ form the key exchange phase, 3 the key confirmation phase, and r the protocol execution phase. We therefore have that $|\mathcal{I}_b| \leq (1-1/c) \cdot c(r+5) \leq (c-1)(r+5)$. As noted above,

Algorithm 3: Protocol Π against c-unbalanced split-state tampering functions

We compile Π_0 into Π below. Let $(\mathsf{Share}, \mathsf{Reconstruct})$, and Π_0 be as in Theorem 3. The communication proceeds in three phases, a key exchange phase, a key confirmation phase, and a protocol execution phase. All messages in the following protocol have a fixed length. Whenever a party in the protocol aborts, she outputs $\perp$ instead of the transcript.

Key Exchange Phase: The number of rounds in the key exchange phase depends on the number of rounds r of the underlying protocol and on the parameter c that determines how unbalanced the states are allowed to be. Let $d = \lceil ((c-1)(r+5)+1)/2 \rceil$.

1. Alice and Bob choose ℓ''-bit strings $k_1, k_2 \leftarrow\$ \{0,1\}^{\ell''}$ respectively and secret share them into d shares each as $s_1^A, \ldots, s_d^A \leftarrow \mathsf{Share}(k_1)$ and $s_1^B, \ldots, s_d^B \leftarrow \mathsf{Share}(k_2)$.
2. Alice chooses d random strings $r_{1,1}^A, \ldots, r_{1,d}^A \leftarrow\$ \{0,1\}^{2\lambda}$ and sends $m_1^A = (r_{1,1}^A, \ldots, r_{1,d}^A)$ to Bob.
3. For every $1 \leq i \leq d$ Alice and Bob proceed as follows
   (a) Bob chooses $d-i+1$ random string $r_{i,i}^B, \ldots, r_{i,d}^B \leftarrow\$ \{0,1\}^{2\lambda}$, computes the tag $t_i^B := \mathsf{MAC}(r_{1,i}^A \oplus \ldots \oplus r_{i,i}^A, s_i^B)$ and sends $m_i^B = (s_i^B, r_{i,i}^B, \ldots, r_{i,d}^B, t_i^B)$ to Alice.
   (b) Alice verifies that $\mathsf{Vf}(r_{1,i}^A \oplus \cdots \oplus r_{i,i}^A, s_i^B, t_i^B) = 1$ and aborts otherwise.
   (c) Alice chooses $d-i$ random strings $r_{i+1,i+1}^A, \ldots, r_{i+1,d}^A \leftarrow\$ \{0,1\}^{2\lambda}$ (note that once $i = d$ this means no random string at all), computes the tag $t_i^A := \mathsf{MAC}(r_{1,i}^B \oplus \ldots \oplus r_{i,i}^B, s_i^A)$ and sends $m_{i+1}^A = (s_i^A, r_{i+1,i+1}^A, \ldots, r_{i+1,d}^A, t_i^A)$ to Bob.
   (d) Bob verifies that $\mathsf{Vf}(r_{1,i}^B \oplus \cdots \oplus r_{i,i}^B, s_i^A, t_i^A) = 1$ and aborts otherwise.
4. Once all the shares have been exchanged, Alice reconstructs $k_2' := \mathsf{Reconstruct}(s_1^B, \ldots, s_d^B)$. If $k_2' = \perp$, she aborts. Otherwise she extracts $k = \mathsf{Ext}(k_1, k_2')$. Bob reconstructs $k_1' := \mathsf{Reconstruct}(s_1^A, \ldots, s_d^A)$. If $k_1' = \perp$, he aborts. Otherwise he extracts $k = \mathsf{Ext}(k_1', k_2)$.
5. Both parties then split $k = k_A \| k_B \| k_1^{\mathsf{auth}} \| k_1^{\mathsf{enc}} \| \ldots \| k_r^{\mathsf{auth}} \| k_r^{\mathsf{enc}}$ into substrings, where $|k_A| = |k_B| = \left|k_i^{\mathsf{auth}}\right| = 2\lambda$ and $|k_i^{\mathsf{enc}}| = \ell$.

Key Confirmation Phase: Alice and Bob verify that they agree on the exchanged key.

1. Bob chooses a random challenge $c_B \leftarrow\$ \{0,1\}^{\ell}$ and sends it to Alice.
2. Alice computes $t_B := \mathsf{MAC}(k_B, c_B)$, chooses a challenge $c_A \leftarrow\$ \{0,1\}^{\ell}$, and sends t_B, c_A to Bob.
3. If $\mathsf{Vf}(k_B, c_B, t_B) = 1$, Bob computes $t_A := \mathsf{MAC}(k_A, c_A)$ and sends t_A to Alice. Otherwise he aborts.
4. If $\mathsf{Vf}(k_A, c_A, t_A) = 1$, Alice proceeds to the next phase. Otherwise she aborts.

Protocol Execution Phase: Both parties initialize their view of the underlying protocol as a empty lists $\mathrm{trans}_A = \mathrm{trans}_B = \emptyset$. For each protocol message the parties then proceed as follows:

1. In the ith round, if it is Alice's (resp. Bob's) turn to send a message she invokes the next-message function of the underlying protocol $m_i := \pi_A^0(i, x, \mathrm{trans}_A)$ (resp. $m_i := \pi_B^0(i, y, \mathrm{trans}_B)$) and adds the message to her view $\mathrm{trans}_A := \mathrm{trans}_A \| m_i$ (resp. $\mathrm{trans}_B := \mathrm{trans}_B \| m_i$).
2. Next the party computes the one-time pad encryption $c_i := m_i \oplus k_i^{\mathsf{enc}}$ of m_i as well as an authentication tag $t_i := \mathsf{MAC}(k_i^{\mathsf{auth}}, c_i)$ and sends c_i, t_i to the other party.
3. If $\mathsf{Vf}(k_i^{\mathsf{auth}}, c_i, t_i) = 1$ the other party decrypts $m_i := c_i \oplus k_i^{\mathsf{enc}}$ and adds the message to their view, i.e., $\mathrm{trans}_A := \mathrm{trans}_A \| m_i$ or $\mathrm{trans}_B := \mathrm{trans}_B \| m_i$.

Finally the underlying protocol terminates and both parties output their respective transcripts trans_A or trans_B or $\perp$ if they aborted at some point.

we need to deal with leakage due to conditional aborts for every message being tampered. I.e., the tampered message $\bar{m}_i$ in round i with $i \in \mathcal{I}_b$ can, in addition to all previous messages in $\mathcal{I}_b$, also depend on some joint leakage over all previous messages in $\mathcal{I}_{1-b}$ due to conditional aborts, simply by observing that the protocol has *not* aborted.

Claim 4. *The tampered message $\bar{m}_i$ in round i with $i \in \mathcal{I}_b$ can depend on at most $\lambda/2$ bits of joint leakage over $\{m_j | j \in \mathcal{I}_{1-b} \wedge j \leq i\}$.*

Proof. We know that f does *not* cause an abort with probability at least $\delta(\lambda) = 2^{-\lambda/2}$. Therefore, the tampering function g_i learns at most $\log \delta^{-1}(\lambda) = \log 2^{\lambda/2} = \lambda/2$ bits of joint leakage over previous messages in $\mathcal{I}_{1-b}$. □

We will argue that conditioned on the protocol not having aborted and the complete view of any tampering function g_i in the key confirmation and protocol execution phase the key $k = \mathsf{Ext}(k_1, \bar{k}_2)$ computed by Alice in the key exchange phase remains ϵ'' close to uniform. For this we first note that up to step 5 in Algorithm 4 the sampler acts identically to a real execution of the protocol.

Algorithm 4: Sampler of distribution D_f for Algorithm 3

```
1. Sample $k_1, k_2 \leftarrow_{\$} \{0,1\}^{\ell''}$ and share them as $s_1^A, \dots, s_d^A \leftarrow \mathsf{Share}(k_1)$ and $s_1^B, \dots, s_d^B \leftarrow \mathsf{Share}(k_2)$.
2. Sample $d^2 + d$ strings $r_{1,1}^A, \dots, r_{1,d}^A, r_{2,2}^A, \dots, r_{2,d}^A, \dots, r_{d,d}^A \leftarrow_{\$} \{0,1\}^{2\lambda}$
                            $r_{1,1}^B, \dots, r_{1,d}^B, r_{2,2}^B, \dots, r_{2,d}^B, \dots, r_{d,d}^B \leftarrow_{\$} \{0,1\}^{2\lambda}$.
3. Let $m_i^A := (r_{1,1}^A, \dots, r_{1,d}^A)$ and apply the tampering function as $\bar{m}_i^A = (\bar{r}_{1,1}^A, \dots, \bar{r}_{1,d}^A) := g_1(m_1^A)$.
4. For $1 \leq i \leq d$ perform the following steps
    (a) Compute $t_i^B := \mathsf{MAC}(\bar{r}_{1,i}^A \oplus \dots \oplus \bar{r}_{i,i}^A, s_i^B)$ and let $m_i^B := (s_i^B, r_{i,i}^B, \dots, r_{i,d}^B, t_i^B)$.
    (b) Apply the tampering function as $\bar{m}_i^B = (\bar{s}_i^B, \bar{r}_{i,i}^B, \dots, \bar{r}_{i,d}^B, \bar{t}_i^B) := g_{2i}(m_1^A, m_1^B, m_2^A, \dots, m_i^B)$.
    (c) If $\mathsf{Vf}(r_{1,i}^A \oplus \dots \oplus r_{i,i}^A, \bar{s}_i^B, \bar{t}_i^B) = 0$, output $(\bot, \bot)$.
    (d) Compute $t_i^A := \mathsf{MAC}(\bar{r}_{1,i}^B \oplus \dots \oplus \bar{r}_{i,i}^B, s_i^A)$ and let $m_{i+1}^A := (s_i^A, r_{i+1,i+1}^A, \dots, r_{i+1,d}^A, t_i^A)$.
    (e) Apply the tampering function as $\bar{m}_{i+1}^A = (\bar{s}_i^A, \bar{r}_{i+1,i+1}^A, \dots, \bar{r}_{i+1,d}^A, \bar{t}_i^A) := g_{2i+1}(m_1^A, m_1^B, \dots, m_{i+1}^A)$.
    (f) If $\mathsf{Vf}(r_{1,i}^B \oplus \ldots \oplus r_{i,i}^B, \bar{t}_i^A, \bar{s}_i^A) = 0$, output $(\bot, \bot)$.
5. Reconstruct $\bar{k}_1 := \mathsf{Reconstruct}(\bar{s}_1^A, \dots, \bar{s}_d^A)$ and $\bar{k}_2 := \mathsf{Reconstruct}(\bar{s}_1^B, \dots, \bar{s}_d^B)$. If $\bar{k}_1 = \bot$ or $\bar{k}_2 = \bot$, output $(\bot, \bot)$.
6. If $\mathsf{Ext}(k_1, k_2) \neq \mathsf{Ext}(\bar{k}_1, k_2)$ or $\mathsf{Ext}(k_1, k_2) \neq \mathsf{Ext}(k_1, \bar{k}_2)$, stop and output $(\bot, \bot)$.
7. Else, if $\mathsf{Ext}(\bar{k}_1, k_2) = \mathsf{Ext}(k_1, \bar{k}_2) = \mathsf{Ext}(k_1, k_2)$, simulate a protocol execution tampered with $f$
    (a) Replace all messages with random strings of appropriate length and apply the tampering function to those messages.
    (b) If for any index $2d + 4 < i < c(r + 5)$ it holds that $m_i \neq f_i(m_1, \dots, m_i)$ then output $(\bot, \bot)$.
    (c) If $m_{c(r+5)} \neq f_{c(r+5)}(m_1, \dots, m_{c(r+5)})$ then output $(\mathsf{same}, \bot)$, otherwise output $(\mathsf{same}, \mathsf{same})$.
```

Lemma 5. *If Alice, or respectively D_f, does not abort during the key exchange phase, then $\bar{k}_2 = k_2$ except with probability $\epsilon' + (d+1) \cdot 2^{-\lambda/2}$.*

Proof. Assume that $\bar{k}_2 \neq k_2$. We will show that Alice, or respectively D_f, aborts with probability $1 - \epsilon' - (d+1) \cdot 2^{-\lambda/2}$. The tampering function f induces a tampering function g on Bob's secret shares $(s_1^B, \dots, s_d^B)$. If this tampering is *detectable*, then the execution aborts except with probability ϵ'. Recall that we have $\bar{k}_2 \neq k_2$. If $\bar{k}_2 = \bot$ then Alice aborts with probability 1. If however, $\bar{k}_2 \neq \bot$, then by Definition 5 Alice aborts with probability $1 - \epsilon'$.

It remains to deal with the case when the tampering is *not* detectable. We will prove that this does not occur except with probability $(d+1) \cdot 2^{-\lambda/2}$. Let f be non-detectable. We then prove a series of claims.

Claim 6. *Let $i \in \mathcal{M}$ be an arbitrary index. Let $\mathcal{I}_b$ denote the set of indices from the description of f, such that $2i \in \mathcal{I}_b$.*[7] *If Alice does not abort then $\{1, 3, \dots, 2i-1\} \subseteq \mathcal{I}_b$ except with probability $2^{-\lambda/2}$.*

Proof. We have by definition that $\bar{s}_i^B \neq s_i^B$, since $i \in \mathcal{M}$. However, Alice aborts unless $\mathsf{Vf}(r_{1,i}^A \oplus \dots \oplus r_{i,i}^A, \bar{s}_i^B, \bar{t}_i^B) = 1$. The key of this information theoretic MAC is perfectly secret shared across all of Alice's previous messages, i.e., messages $1, 3, \dots, 2i-1$ of the protocol. Therefore, if $\{1, 3, \dots, 2i-1\} \nsubseteq \mathcal{I}_b$ then by Claim 4 the tampering function can learn at most $\lambda/2$ bits of the key and it would hold that $\Pr\left[\mathsf{Vf}(r_{1,i}^A \oplus \dots \oplus r_{i,i}^A, \bar{s}_i^B, \bar{t}_i^B) = 1\right] \leq 2^{-\lambda/2}$. Thus, except with probability $2^{-\lambda/2}$ it must hold that $\{1, 3, \dots, 2i-1\} \subseteq \mathcal{I}_b$. □

Claim 7. *It holds that $d \in \mathcal{M}$.*

Proof. Since the tampering is *not* detectable, by Definition 4 and since Claim 4 guarantees the bound on joint leakage, there exists an $i \in \mathcal{M}$ such that $\mathcal{M} \cup \mathcal{I}_i^{\mathsf{in}} = \{1, \dots, d\}$. However, for any j, $\mathcal{I}_j^{\mathsf{in}} \subseteq \{1, \dots, j\}$ and in particular for all $j < d$, $d \notin \mathcal{I}_j^{\mathsf{in}}$. This holds since s_d^B is only revealed *after* the rest of the shares have been tampered with and received by Alice. For an i as required above to exist, it must therefore hold that $d \in \mathcal{M}$. □

Claim 8. *There exists $b \in \{0,1\}$ such that $\{2i | i \in \mathcal{M}\} \subseteq \mathcal{I}_b$ except with probability $d \cdot 2^{-\lambda/2}$.*

[7] Note that s_i^B is sent in round $2i$.

Proof. From Claim 6 it follows that for any $i \in \mathcal{M}$ it holds that $2i \in \mathcal{I}_b$, such that $1 \in \mathcal{I}_b$ except with probability $2^{-\lambda/2}$. By a union bound over all $i \in \mathcal{M}$ and the observation that $|\mathcal{M}| \leq d$ the claim thus follows immediately. □

Claim 9. *There exists $b \in \{0, 1\}$ such that $\{1, \ldots, 2d\} \subseteq \mathcal{I}_b$ except with probability $(d+1) \cdot 2^{-\lambda/2}$.*

Proof. We have from Claim 8 that there exists $b \in \{0, 1\}$ such that $\{2i | i \in \mathcal{M}\} \subseteq \mathcal{I}_b$ except with probability $d \cdot 2^{-\lambda/2}$. We also have from Claim 6 and Claim 7, that $\{1, 3, \ldots, 2d-1\} \subseteq \mathcal{I}_b$ except with probability $2^{-\lambda/2}$. And finally, as previously observed there must exist an $i \in \mathcal{M}$ such that $\mathcal{M} \cup \mathcal{I}_i^{\mathsf{in}} = \{1, \ldots, d\}$. Since it must naturally hold that $\{2j | j \in \mathcal{I}_i^{\mathsf{in}}\} \subseteq \mathcal{I}_b$, we can thus with another union bound conclude that $\{1, \ldots, 2d\} \subseteq \mathcal{I}_b$ except with probability $(d+1) \cdot 2^{-\lambda/2}$. □

From Claim 9 it thus follows that, if the induced tampering is non-detectable, then except with probability $(d+1) \cdot 2^{-\lambda/2}$ we have $|\mathcal{I}_b| \geq 2d \geq ((c-1)(r+5)+1) \geq (c-1)(r+5)$ which would contradict the fact that f is c-unbalanced. Therefore, the induced tampering can only be non-detectable with probability $(d+1) \cdot 2^{-\lambda/2}$. Lemma 5 then immediately follows using an additional union bound. □

A completely symmetric argument can be made for $\bar{k}_1 = k_1$, where otherwise Bob aborts with probability $1 - \epsilon' - (d+1) \cdot 2^{-\lambda/2}$, causing Alice to also abort. This means that if Alice does not abort, we have that $k = \mathsf{Ext}(k_1, \bar{k}_2) = \mathsf{Ext}(\bar{k}_1, k_2) = \mathsf{Ext}(k_1, k_2)$ with probability at least $1 - 2(\epsilon' - (d+1) \cdot 2^{-\lambda/2})$.[8]

Now, consider how much information about k_1 and k_2 a tampering function g_i can learn. Let $\mathcal{I}_b$ be the set of indices, such that $i \in \mathcal{I}_b$. Clearly, g_i has complete knowledge of all shares s_j^B with $2j \in \mathcal{I}_b$ and all shares s_j^A with $2j+1 \in \mathcal{I}_b$. Further, g_i receives joint leakage over shares in $\mathcal{I}_{1-b}$ simply by observing the fact that the protocol has not yet aborted. This leakage is however bounded by Claim 4 by $\lambda/2$ bits. By the perfect privacy of the secret sharing scheme, it follows that $\lambda/2$ bits of joint leakage over all shares can reveal at most $\lambda/2$ bits of the secret.

Since a set of indices with $|\mathcal{I}_b| \geq 2d+1$ would be too large for a c-unbalanced split state tampering function, $\mathcal{I}_b$ cannot possibly contain all the shares. Thus, the maximum amount of information the tampering function g_i can gain about k_1 and k_2 is exactly one of the two strings and $\lambda/2$ bits of the other string. Since Ext is a strong 2-source extractor for sources with min-entropy $\ell'' - \lambda/2$, this implies that in this case with probability at least $1 - \epsilon''$ the extracted key-material remains ϵ'' close to uniform. Overall, this means that with probability at least $1 - 2 \cdot (\epsilon' + (d+1) \cdot 2^{-\lambda/2}) - \epsilon''$, k remains ϵ'' close to uniform from the point of view of any tampering function g_i.

To recap, if any of the key-shares are tampered with in such a way that the original keys are not reconstructed, then the sampling algorithm will always output $(\perp, \perp)$, while the parties in the real protocol will do so with probability at least $1 - 2 \cdot (\epsilon' + (d+1) \cdot 2^{-\lambda/2})$. If the shares were not tampered with and thus $k = \mathsf{Ext}(\bar{k}_1, k_2) = \mathsf{Ext}(k_1, \bar{k}_2) = \mathsf{Ext}(k_1, k_2)$, then since k is distributed ϵ''-close to uniform – the random messages in the simulated protocol execution phase are distributed ϵ'' close to a real protocol execution. Now, if f tampers with any message of the key-confirmation or protocol-execution phase except for the very last one, then the sampling algorithm always outputs $(\perp, \perp)$, whereas if only the very last message is tampered with the sampling algorithm outputs $(\mathsf{same}, \perp)$. In a real protocol execution when tampering with any message, the information theoretic MAC must be computed almost independently of k, since k remains ϵ'' close to uniform. Therefore, if any message is tampered with in a real protocol execution, the receiving party will abort with probability $1 - 2^{-\lambda} - \epsilon''$, causing both parties to output $\perp$, except if it only happens in the very last message, where only Bob will abort with probability $1 - 2^{-\lambda} - \epsilon''$ and output $\perp$ and Alice will retain the correct transcript. On the other hand, if no message is tampered with, the sampling algorithm outputs $(\mathsf{same}, \mathsf{same})$ and both Alice and Bob in a real protocol execution retain the correct transcript. This follows since in this case Alice and Bob agree on a key. Overall a union bound then gives us an upper bound on the statistical distance between D_f and the distribution of both parties' outputs in a real execution of $2\epsilon' + 3\epsilon'' + 2(d+1) \cdot 2^{-\lambda/2} + 2^{-\lambda-1}$. With $d = \lceil ((c-1)(r+5)+1)/2 \rceil$, this leads to the claimed bound of $\epsilon(\lambda) = 2\epsilon' + 3\epsilon'' + ((c-1)(r+5)+3) \cdot 2^{-\lambda/2} + 2^{-\lambda+1}$. □

[8] Note that the tampering function cannot influence the values k_1, k_2 at all since they are sampled independently of the protocol transcript.

7 Fragmented Sliding Window Tampering

The sliding window model is a very natural restriction of algorithms and is considered in a variety of contexts, in particular also for error correcting codes [48]. The idea of the sliding window is that an adversary can only watch a stream of data through a window of fixed size. In the context of interactive non-malleable codes this means that the tampering function "remembers" only the last w messages. That is, the tampering function gets as input the last w (untampered) messages of the protocol transcript to compute the tampered message.

We in fact consider a stronger class of functions that we call *fragmented* sliding window. Functions with a fragmented window of size w can depend on *any* w previous messages, not just the *last* w. In a sense the adversary is still watching the transcript through a fixed size window, it can freely choose which fragments of the window remain transparent and which ones become opaque.

Comparing this class with c-unbalanced split-state tampering functions, we note that the size of the window is now fixed and does not scale with the number of messages. On the other hand the different sets of messages tampering can depend on are no longer required to be disjoint. E.g., the tampering of each single message could depend on the first message of the protocol, something that would not be possible in the case of split-state functions.

Definition 13 (Fragmented Sliding Window Tampering Functions).
Functions of the class of w-size fragmented sliding window tampering functions $\mathcal{F}^w_{frag}$ for an r-round interactive protocols are defined by an r-tuple of functions $(g_1, \ldots, g_r)$ and an r-tuple of sets $(S_1, \ldots, S_r)$ such that $S_1 = \emptyset$, $S_i \subseteq S_{i-1} \cup \{i-1\}$ and $|S_i| \leq w$ for $1 < i \leq r$. Let $m_1, \ldots, m_i$ be the messages sent by the participants of the protocol in a partial execution. The tampering function for message m_i is then defined as $f_i(m_1, \ldots, m_i) := g_i\big(m_i, (m_j)_{j \in S_i}\big)$.

7.1 INMC for Fragmented Sliding Window Tampering

Even though there are important conceptual differences between fragmented sliding window tampering functions and c-unbalanced split-state tampering functions, essentially identical protocol can be used to achieve protocol-non-malleability for fragmented sliding window tampering functions. The difference is how the key exchange phase scales. The window-size is fixed and does not depend on the round complexity of the protocol. This means that d – the number of shares Alice and Bob split their keys into – must scale with w instead of the underlying protocol's round complexity.

Theorem 10. *Let Π_0 denote a correct, r-round protocol, with length-ℓ messages. Let $(\mathsf{Share}, \mathsf{Reconstruct})$ be a $w+2$-out-of-$w+2$ perfectly private, ϵ'-tamper evident secret sharing scheme for up to $\lambda'/2$ bits of leakage with message length ℓ'' and share length ℓ' Let λ' be the target security parameter, then we set $\lambda = \max(\ell, \ell', \lambda')$. Let $\mathsf{MAC} : \{0,1\}^{2\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ be a $2^{-\lambda}$-secure information theoretic message authentication code. Let $\mathsf{Ext} : \{0,1\}^{\ell''} \to r\ell + (2r+4)\lambda$ be a strong two-source extractor for sources with min-entropy $\ell'' - \lambda/2$ with error ϵ''. We assume wlog that Alice sends both the first and last message in Π_0. Then for any w there exists a $r + 2w + 8$-round encoding Π of Π_0 that is $\epsilon(\lambda) = 3 \cdot 2^{-\lambda} + 2\epsilon'(\lambda) + 2\epsilon''$-protocol non-malleable against $\mathcal{F}^w_{frag}$.*

Proof for Theorem 10 The protocol for this case is identical to the case of split-state tampering functions. The only difference is in the number of round of the key exchange phase. Specifically we choose d in the description of Π in Algorithm 3 to be $d = w + 2$ leading to a key exchange phase with $2w + 5$ rounds. Otherwise Π remains unaltered.

Correctness therefore follows in exactly the same way as in Theorem 3. However, the arguments in the proof of protocol non-malleability are slightly different, because we need to exploit different properties of the family of tampering functions.

Let f be a w-size fragmented sliding window tampering function described by $(g_1, \ldots, g_r)$ and $(S_1, \ldots, S_r)$ as in Definition 13. To prove that the coding scheme is non-malleable, we need to prove a distributions D_f as in Definition 9 exists.

The distribution D_f: When sampling from D_f we again need to deal with the problem that the tampering function can communicate information through conditional aborts. To deal with this problem we again sample differently depending on the probability of f causing an abort in a protocol execution. Let $1 - \delta(\lambda)$ be the probability of f causing either party to abort *before* the last message in the protocol is sent. If $\delta(\lambda) \leq 2^{-\lambda/2}$, the distribution D_f is sampled by simply outputting $\perp$. Clearly this distribution is $2^{-\lambda/2} \leq \epsilon(n)$ close to the real distribution, since f causes both parties to abort and output $\perp$ with probability at least $1 - 2^{-\lambda/2}$.

If $\delta > 2^{-\lambda/2}$, the distribution D_f is sampled identically to the case of split-state tampering function in Algorithm 4. It remains to show that D_f is $2\epsilon'(\lambda) + 2\epsilon'' + 3 \cdot 2^{-\lambda}$ close to the tampered transcript distribution. We first note that the protocol Π overall has $2d + 4 + r = 2w + 8 + r$ rounds, of which $2w + 5$ form the key exchange phase, 3 the key confirmation phase, and r the protocol execution phase.

Claim 11. *The tampered message $\bar{m}_i$ in round i can depend on at most $2^{-\lambda/2}$ bits of joint leakage over $\{m_j | j \notin S_i \wedge j \leq i\}$.*

The above claim follows identically to Claim 4.

We will argue that conditioned on the protocol not having aborted and the complete view of any tampering function g_i in the key confirmation and protocol execution phase the key $k = \mathsf{Ext}(k_1, \bar{k}_2)$ computed by Alice in the key exchange phase remains ϵ'' close to uniform.

Lemma 12. *If Alice, or respectively D_f, does not abort during the key exchange phase, then $\bar{k}_2 = k_2$ except with probability $\epsilon' + 2^{-\lambda/2}$.*

Proof. Assume that $\bar{k}_2 \neq k_2$. We will show that Alice, or respectively D_f, aborts with probability $1 - \epsilon' - 2d \cdot 2^{-\lambda/2}$. The tampering function f induces a tampering function g on Bob's secret shares $(s_1^B, \ldots, s_d^B)$. If this tampering is *detectable*, then the execution aborts except with probability ϵ'. Recall that we have $\bar{k}_2 \neq k_2$. If $\bar{k}_2 = \perp$ then Alice aborts with probability 1. If however, $\bar{k}_2 \neq \perp$, then by Definition 5 Alice aborts with probability $1 - \epsilon'$.

It remains to deal with the case when the tampering is *not* detectable. We will prove that this does not occur except with probability $(d + 1) \cdot 2^{-\lambda/2}$. Let f be non-detectable. We then prove a series of claims.

Claim 13. *Let $i \in \mathcal{M}$ be an arbitrary index. If Alice does not abort then $\{1, 3, \ldots, 2i - 1\} \subseteq S_{2i}$ except with probability $2^{-\lambda/2}$.*[9]

Proof. We have by definition that $\bar{s}_i^B \neq s_i^B$, since $i \in \mathcal{M}$. However, Alice aborts unless $\mathsf{Vf}(r_{1,i}^A \oplus \cdots \oplus r_{i,i}^A, \bar{s}_i^B, \bar{t}_i^B) = 1$. The key of this information theoretic MAC is perfectly secret shared across all of Alice's previous messages, i.e., messages $1, 3, \ldots, 2i - 1$ of the protocol. Therefore, if $\{1, 3, \ldots, 2i - 1\} \subsetneq S_{2i}$ then by Claim 11 the tampering function can learn at most $\lambda/2$ bits of the key and it would hold that $\Pr\left[\mathsf{Vf}(r_{1,i}^A \oplus \cdots \oplus r_{i,i}^A, \bar{s}_i^B, \bar{t}_i^B) = 1\right] \leq 2^{-\lambda/2}$. Thus, except with probability $2^{-\lambda/2}$ it must hold that $\{1, 3, \ldots, 2i - 1\} \subseteq S_{2i}$. □

Claim 14. *It holds that $d \in \mathcal{M}$.*

Proof. Since the tampering is *not* detectable, by Definition 4 and since Claim 11 guarantees the bound on joint leakage, there exists an $i \in \mathcal{M}$ such that $\mathcal{M} \cup \mathcal{I}_i^{\in} = \{1, \ldots, d\}$. However, for any j, $\mathcal{I}_j^{\mathsf{in}} \subseteq \{1, \ldots, j\}$ and in particular for all $j < d$, $d \notin \mathcal{I}_j^{\mathsf{in}}$. This holds since s_d^B is only revealed *after* the rest of the shares have been tampered with and received by Alice. For an i as required above to exist, it must therefore hold that $d \in \mathcal{M}$. □

Combining Claim 13 and Claim 14 it follows directly, that $\{1, 3, \ldots, 2i - 1\} \subseteq S_{2d}$ and thereby $|S_{2d}| \geq d = w + 2$ except with probability $2^{-\lambda/2}$, which contradicts the fact that f is a w-size fragmented sliding window tampering function. Therefore, the induced tampering can only be non-detectable with probability $2^{-\lambda/2}$ as claimed. Lemma 12 then immediately follows using an additional union bound. □

[9] Note that s_i^B is sent in round $2i$.

Again a completely symmetric argument can be made for $\bar{k}_1 = k_1$, where otherwise Bob aborts with probability $1 - \epsilon' - 2^{-\lambda/2}$, causing Alice to also abort. This means that if the parties do not abort, we have that $k = \mathsf{Ext}(k_1, \bar{k}_2) = \mathsf{Ext}(\bar{k}_1, k_2) = \mathsf{Ext}(k_1, k_2)$ with probability at least $1 - 2(\epsilon' - 2^{-\lambda/2})$.[10]

Now, consider how much information about k_1 and k_2 a tampering function g_i can learn. Clearly, g_i has complete knowledge of all shares s_j^B with $2j \in S_i$ and all shares s_j^A with $2j + 1 \in S_i$. Further, g_i receives joint leakage over shares not in S_i simply by observing the fact that the protocol has not yet aborted. This leakage is however bounded by Claim 11 by $\lambda/2$ bits. By the perfect privacy of the secret sharing scheme, it follows that $\lambda/2$ bits of joint leakage over all shares can reveal at most $\lambda/2$ bits of the secret.

Since a set of indices with $|S_i| \geq 2d = 2w + 4$ would be too large for a w-unbalanced split state tampering function, S_i cannot possibly contain all the shares. Thus, the maximum amount of information the tampering function g_i can gain about k_1 and k_2 is $\lambda/2$ bits of joint leakage. Since Ext is a strong 2-source extractor for sources with min-entropy $\ell'' - \lambda/2$, this implies that in this case with probability at least $1 - \epsilon''$ the extracted key-material remains ϵ'' close to uniform. Overall, this means that with probability at least $1 - 2 \cdot (\epsilon' - 2^{-\lambda/2}) - \epsilon''$, k remains ϵ'' close to uniform from the point of view of any tampering function g_i.

To recap, if any of the key-shares are tampered with in such a way that the original keys are not reconstructed, then the sampling algorithm will always output $(\perp, \perp)$, while the parties in the real protocol will do so with probability at least $1 - 2 \cdot (\epsilon' + (d + 1) \cdot 2^{-\lambda/2})$. If the shares were not tampered with and thus $k = \mathsf{Ext}(\bar{k}_1, k_2) = \mathsf{Ext}(\bar{k}_1, k_2) = \mathsf{Ext}(k_1, k_2)$, then since k is distributed ϵ''-close to uniform, the random messages in the simulated protocol execution phase are distributed ϵ'' close to a real protocol execution. Now, if f tampers with any message of the key-confirmation or protocol-execution phase, then the sampling algorithm always outputs $(\perp, \perp)$, except, if the tampered message is the very last one, in which case it outputs $(\mathsf{same}, \perp)$. In a real protocol execution when tampering with any message, the information theoretic MAC must be computed almost independently of k, since k remains ϵ'' close to uniform. Therefore, if any message is tampered with in a real protocol execution, the receiving party will abort with probability $1 - 2^{-\lambda} - \epsilon''$, causing Bob to output $\perp$ and Alice to also output $\perp$, unless the tampered message is sent in the very last round, in which case Alice retains the correct transcript. On the other hand, if no message is tampered with, the sampling algorithm outputs $(\mathsf{same}, \mathsf{same})$ and both parties in a real protocol execution retain the correct transcript. This follows since in this case Alice and Bob agree on a key.

Overall using a union bound this gives us an upper bound on the statistical distance between D_f and Alice's transcript in a real execution of $2\epsilon' + 2\epsilon'' + 2 \cdot 2^{-\lambda/2}) + 2^{-\lambda}$ as claimed. □

Acknowledgments

We would like to thank the anonymous reviewers for TCC 2019 for suggesting a stronger and more natural notion of non-malleability. We would also like to thank Ran Gelles for helpful comments on an earlier version of our writeup.

References

1. Aggarwal, D., Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Optimal computational split-state non-malleable codes. In: TCC 2016-A. pp. 393–417 (2016)
2. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. In: 47th ACM STOC. pp. 459–468 (2015)
3. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: 46th ACM STOC. pp. 774–783 (2014)
4. Aggarwal, D., Döttling, N., Nielsen, J.B., Obremski, M., Purwanto, E.: Continuous non-malleable codes in the 8-split-state model. In: EUROCRYPT 2019. pp. 531–561 (2019)
5. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Explicit non-malleable codes against bit-wise tampering and permutations. In: CRYPTO 2015. pp. 538–557 (2015)

[10] Note that the tampering function cannot influence the values k_1, k_2 at all since they are sampled independently of the protocol transcript.

6. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In: TCC 2015. pp. 375–397 (2015)
7. Alon, N., Braverman, M., Efremenko, K., Gelles, R., Haeupler, B.: Reliable communication over highly connected noisy networks. In: 35th ACM PODC. pp. 165–173 (2016)
8. Ball, M., Dachman-Soled, D., Guo, S., Malkin, T., Tan, L.Y.: Non-malleable codes for small-depth circuits. In: 59th FOCS. pp. 826–837 (2018)
9. Ball, M., Dachman-Soled, D., Kulkarni, M., Lin, H., Malkin, T.: Non-malleable codes against bounded polynomial time tampering. In: EUROCRYPT 2019. pp. 501–530 (2019)
10. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes for bounded depth, bounded fan-in circuits. In: EUROCRYPT 2016. pp. 881–908 (2016)
11. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In: EUROCRYPT 2018. pp. 618–650 (2018)
12. Ball, M., Guo, S., Wichs, D.: Non-malleable codes for decision trees. Cryptology ePrint Archive, Report 2019/379 (2019)
13. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. In: CRYPTO 2005. pp. 361–377 (2005)
14. Bourgain, J.: More on the sum-product phenomenon in prime fields and its applications. International Journal of Number Theory 1(01), 1–32 (2005)
15. Brakerski, Z., Kalai, Y.T.: Efficient interactive coding against adversarial noise. In: 53rd FOCS. pp. 160–166 (2012)
16. Braverman, M., Efremenko, K., Gelles, R., Haeupler, B.: Constant-rate coding for multiparty interactive communication is impossible. In: 48th ACM STOC. pp. 999–1010 (2016)
17. Braverman, M., Gelles, R., Mao, J., Ostrovsky, R.: Coding for interactive communication correcting insertions and deletions. In: ICALP 2016. pp. 61:1–61:14 (2016)
18. Braverman, M., Rao, A.: Towards coding for maximum errors in interactive communication. In: 43rd ACM STOC. pp. 159–166 (2011)
19. Cachin, C., Maurer, U.M.: Unconditional security against memory-bounded adversaries. In: CRYPTO'97. pp. 292–306 (1997)
20. Chandran, N., Goyal, V., Mukherjee, P., Pandey, O., Upadhyay, J.: Block-wise non-malleable codes. In: ICALP 2016. pp. 31:1–31:14 (2016)
21. Chandran, N., Kanukurthi, B., Raghuraman, S.: Information-theoretic local non-malleable codes and their applications. In: TCC 2016-A. pp. 367–392 (2016)
22. Chattopadhyay, E., Goyal, V., Li, X.: Non-malleable extractors and codes, with their many tampered extensions. In: 48th ACM STOC. pp. 285–298 (2016)
23. Chattopadhyay, E., Li, X.: Non-malleable codes and extractors for small-depth circuits, and affine functions. In: 49th ACM STOC. pp. 1171–1184 (2017)
24. Chattopadhyay, E., Zuckerman, D.: Non-malleable codes against constant split-state tampering. In: 55th FOCS. pp. 306–315 (2014)
25. Cheraghchi, M., Guruswami, V.: Non-malleable coding against bit-wise and split-state tampering. In: TCC 2014. pp. 440–464 (2014)
26. Cheraghchi, M., Guruswami, V.: Capacity of non-malleable codes. IEEE Transactions on Information Theory 62(3), 1097–1118 (Mar 2016)
27. Chor, B., Goldreich, O.: Unbiased bits from sources of weak randomness and probabilistic communication complexity (extended abstract). In: 26th FOCS. pp. 429–442 (1985)
28. Chung, K.M., Pass, R., Telang, S.: Knowledge-preserving interactive coding. In: 54th FOCS. pp. 449–458 (2013)
29. Coretti, S., Dodis, Y., Tackmann, B., Venturi, D.: Non-malleable encryption: Simpler, shorter, stronger. In: TCC 2016-A. pp. 306–335 (2016)
30. Coretti, S., Faonio, A., Venturi, D.: Rate-optimizing compilers for continuously non-malleable codes. Cryptology ePrint Archive, Report 2019/055 (2019)
31. Coretti, S., Maurer, U., Tackmann, B., Venturi, D.: From single-bit to multi-bit public-key encryption via non-malleable codes. In: TCC 2015. pp. 532–560 (2015)
32. Dachman-Soled, D., Kulkarni, M., Shahverdi, A.: Tight upper and lower bounds for leakage-resilient, locally decodable and updatable non-malleable codes. In: PKC 2017. pp. 310–332 (2017)
33. Dachman-Soled, D., Liu, F.H., Shi, E., Zhou, H.S.: Locally decodable and updatable non-malleable codes and their applications. In: TCC 2015. pp. 427–450 (2015)
34. Dodis, Y., Wichs, D.: Non-malleable extractors and symmetric key cryptography from weak secrets. In: 41st ACM STOC. pp. 601–610 (2009)

35. Dziembowski, S., Kazana, T., Obremski, M.: Non-malleable codes from two-source extractors. In: CRYPTO 2013. pp. 239–257 (2013)
36. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: ICS 2010. pp. 434–452 (2010)
37. Efremenko, K., Gelles, R., Haeupler, B.: Maximal noise in interactive communication over erasure channels and channels with feedback. In: ITCS 2015. pp. 11–20 (2015)
38. Faonio, A., Nielsen, J.B., Simkin, M., Venturi, D.: Continuously non-malleable codes with split-state refresh. In: ACNS 18. pp. 121–139 (2018)
39. Faust, S., Hostáková, K., Mukherjee, P., Venturi, D.: Non-malleable codes for space-bounded tampering. In: CRYPTO 2017. pp. 95–126 (2017)
40. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: Continuous non-malleable codes. In: TCC 2014. pp. 465–488 (2014)
41. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: A tamper and leakage resilient von neumann architecture. In: PKC 2015. pp. 579–603 (2015)
42. Faust, S., Mukherjee, P., Venturi, D., Wichs, D.: Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: EUROCRYPT 2014. pp. 111–128 (2014)
43. Franklin, M.K., Gelles, R., Ostrovsky, R., Schulman, L.J.: Optimal coding for streaming authentication and interactive communication. In: CRYPTO 2013. pp. 258–276 (2013)
44. Gelles, R., Haeupler, B.: Capacity of interactive communication over erasure channels and channels with feedback. SIAM J. Comput. 46(4), 1449–1472 (2017)
45. Gelles, R., Haeupler, B., Kol, G., Ron-Zewi, N., Wigderson, A.: Towards optimal deterministic coding for interactive communication. In: 27th SODA. pp. 1922–1936 (2016)
46. Gelles, R., Kalai, Y.T.: Constant-rate interactive coding is impossible, even in constant-degree networks. Electronic Colloquium on Computational Complexity (ECCC), TR17-095 (2017)
47. Gelles, R., Moitra, A., Sahai, A.: Efficient and explicit coding for interactive communication. In: 52nd FOCS. pp. 768–777 (2011)
48. Gelles, R., Ostrovsky, R., Roytman, A.: Efficient error-correcting codes for sliding windows. In: SOFSEM 2014. pp. 258–268 (2014)
49. Ghaffari, M., Haeupler, B.: Optimal error rates for interactive coding II: Efficiency and list decoding. In: 55th FOCS. pp. 394–403 (2014)
50. Ghaffari, M., Haeupler, B., Sudan, M.: Optimal error rates for interactive coding I: adaptivity and other settings. In: 46th ACM STOC. pp. 794–803 (2014)
51. Goyal, V., Kumar, A.: Non-malleable secret sharing. In: 50th ACM STOC. pp. 685–698 (2018)
52. Goyal, V., Kumar, A.: Non-malleable secret sharing for general access structures. In: CRYPTO 2018. pp. 501–530 (2018)
53. Goyal, V., Pandey, O., Richelson, S.: Textbook non-malleable commitments. In: 48th ACM STOC. pp. 1128–1141 (2016)
54. Haeupler, B.: Interactive channel capacity revisited. In: 55th FOCS. pp. 226–235 (2014)
55. Jacobson, V., Braden, R., Borman, D.: RFC1323: TCP extensions for high performance, `http://www.ietf.org/rfc/rfc1323.txt`
56. Jain, A., Kalai, Y.T., Lewko, A.B.: Interactive coding for multiparty protocols. In: ITCS 2015. pp. 1–10 (2015)
57. Kanukurthi, B., Obbattu, S.L.B., Sekar, S.: Four-state non-malleable codes with explicit constant rate. In: TCC 2017. pp. 344–375 (2017)
58. Kanukurthi, B., Obbattu, S.L.B., Sekar, S.: Non-malleable randomness encoders and their applications. In: EUROCRYPT 2018. pp. 589–617 (2018)
59. Li, X.: Improved non-malleable extractors, non-malleable codes and independent source extractors. In: 49th ACM STOC. pp. 1144–1156 (2017)
60. Liu, F.H., Lysyanskaya, A.: Tamper and leakage resilience in the split-state model. In: CRYPTO 2012. pp. 517–532 (2012)
61. Ostrovsky, R., Persiano, G., Venturi, D., Visconti, I.: Continuously non-malleable codes in the split-state model from minimal assumptions. In: CRYPTO 2018. pp. 608–639 (2018)
62. Rajagopalan, S., Schulman, L.J.: A coding theorem for distributed computation. In: 26th ACM STOC. pp. 790–799 (1994)
63. Rao, A.: An exposition of bourgain's 2-source extractor. Electronic Colloquium on Computational Complexity (ECCC), TR07-034 (2007)
64. Schulman, L.J.: Communication on noisy channels: A coding theorem for computation. In: 33rd FOCS. pp. 724–733 (1992)
65. Schulman, L.J.: Deterministic coding for interactive communication. In: 25th ACM STOC. pp. 747–756 (1993)

66. Schulman, L.J.: Coding for interactive communication. IEEE Transactions on Information Theory 42(6), 1745–1756 (Nov 1996)
67. Wee, H.: Black-box, round-efficient secure computation via non-malleability amplification. In: 51st FOCS. pp. 531–540 (2010)

A Instantiating Tamper-Evident Secret Sharing

In this section we instantiate tamper evident secret sharing following Definition 5 based on regular XOR-based secret sharing and an information theoretic message authentication code. The basic idea of the construction is that each share contains a key to verify the integrity of every other share. Intuitively, this means that to change one of the shares in a meaningful way, a tampering function must either know each of the keys embedded into the other shares, to allow it to recompute the message authentication codes, or the other share must also be modified, replacing the relevant key with a different one.

Algorithm 5: $\mathsf{Share}(m)$

```
1. $(s'_1, \ldots, s'_{n-1}) \leftarrow \{0,1\}^{|m|}$
2. $s'_n := s'_1 \oplus \ldots \oplus s'_{n-1} \oplus m$
3. $k_1^1, \ldots, k_1^n, k_2^1 \ldots, k_n^n \leftarrow \{0,1\}^{\max(\lambda,|m|)}$
4. For $1 \leq i \leq n$ and $1 \leq j \leq n$:
   (a) $t_i^j := \mathsf{MAC}(k_i^j, s_j)$
5. For $1 \leq i \leq n$
   (a) $s_i := (s'_i, k_i^1, \ldots k_i^n, t_1^i, \ldots t_n^i)$
6. Output $(s_1, \ldots, s_n)$
```

Algorithm 6: $\mathsf{Reconstruct}(s_1, \ldots, s_n)$

```
1. For $1 \leq i \leq n$
   (a) Parse $s_i$ as $(s'_i, k_i^1, \ldots k_i^n, t_1^i, \ldots t_n^i)$
2. For $1 \leq i \leq n$ and $1 \leq j \leq n$:
   (a) If $\mathsf{Vf}(k_i^j, s'_j, t_i^j) = 0$ output $\perp$
3. Output $(s'_1 \oplus \ldots \oplus s'_n)$
```

Theorem 15. *The secret sharing scheme described in Algorithms 5 and 6 is an n-out-of-n $2^{-\ell}$ private, $2^{-\ell+\nu}$-tamper evident secret sharing scheme for up to ν bits of leakage with message space ℓ and share length $3n \cdot \max(\lambda, \ell) + \ell$.*

Proof of Theorem 15 We need to argue that the scheme described above is correct, private and tamper evident.

Correctness and Privacy Correctness and privacy both trivially follow from the correctness and privacy of the underlying regular (non-tamper evident) XOR secret sharing.

Tamper Evidence Consider an arbitrary tampering function f and an arbitrary fixed message m. We have that

$$\begin{aligned}
&\Pr_{\vec{s}\leftarrow\mathsf{Share}(m)}[\mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \notin \{m, \perp\}] \\
&= \Pr_{\vec{s}\leftarrow\mathsf{Share}(m)}[\mathsf{Reconstruct}(f(\vec{s})) \neq \perp \mid \mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \neq m] \\
&\quad \cdot \Pr_{\vec{s}\leftarrow\mathsf{Share}(m)}[\mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \neq m] \\
&\leq \Pr_{\vec{s}\leftarrow\mathsf{Share}(m)}[\mathsf{Reconstruct}(f(\vec{s})) \neq \perp \mid \mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \neq m] \,.
\end{aligned} \tag{11}$$

Observe that the scheme described above will either output $\perp$, or $\bar{s}'_1 \oplus \ldots \oplus \bar{s}'_n$. Since $\mathsf{Reconstruct}(f(\vec{s})) \neq m$ it must therefore clearly hold that for at least one i, $\bar{s}'_i \neq s'_i$. Fix any such i. $\mathsf{Reconstruct}(f(\vec{s})) \neq \perp$ would then imply that for all $j \in \{1, \ldots, n\}$, $\mathsf{Vf}(\bar{k}_j^i, \bar{s}'_i, \bar{t}_j^i) = 1$. However, f is detectable for $\vec{s}$, therefore there exists a $j \in \{1, \ldots, n\}$, such that $j \notin \mathcal{M} \cup \mathcal{I}_i^{\mathsf{in}}$ and f receives only up to ν bits of joint leakage about $j \notin \mathcal{I}_i^{\mathsf{in}}$. I.e., it holds that $\bar{k}^i_j = k^i_j$, since $j \notin \mathcal{M}$, and $\bar{t}^i_j$ is depending on at most ν bits of k^i_j, since $j \notin \mathcal{I}^{\mathsf{in}}_i$. Therefore, by the information theoretic security of the message authentication code, it holds that $\Pr\left[\mathsf{Vf}(\bar{k}^i_j, \bar{s}'_i, \bar{t}^i_j) = 1\right] \leq 2^{-\lambda+\nu}$. Thus, using Equation 11 we have

$$\begin{aligned} 2^{-\lambda+\nu} &\geq \Pr_{\vec{s} \leftarrow \mathsf{Share}(m)}[\mathsf{Reconstruct}(f(\vec{s})) \neq \bot \,|\, \mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \neq m] \\ &\geq \Pr_{\vec{s} \leftarrow \mathsf{Share}(m)}[\mathsf{Dtct}(\vec{s}, f) = 1 \wedge \mathsf{Reconstruct}(f(\vec{s})) \notin \{m, \bot\}] \end{aligned}$$

and the claim thus follows. □

B Lower Bounds for Interactive Coding with CRS

To see why the known lower bounds for interactive coding also apply in the CRS model, we recall the arguments for those lower bounds. Specifically we focus on the bound of Braverman and Rao [18] for non-adaptive interactive coding.

This lower bound on the tolerable error rate stems from a very simple argument. Consider an r-round protocol Π with single bit inputs $x, y \in \{0, 1\}$ and assume without loss of generality that Alice sends a message in $n \leq r/2$ of the rounds and let Bob have input $y = 0$. Now consider the tampering function f that leaves Bob's messages intact and tampers with Alice's messages to achieve the following. The first $\lfloor n/2 \rfloor$ messages of Alice are tampered to be consistent with input $y = 0$. The remaining $\lceil n/2 \rceil$ messages of Alice are tampered to be consistent with input $y = 1$. Note that since Alice's untampered messages are already consistent with either $y = 0$ or $y = 1$ this requires modifying at most $\lceil n/2 \rceil \leq \lceil r/4 \rceil$ messages. Therefore f is a threshold tampering function, modifying at most an $\lceil r/4 \rceil$ fraction of the transcript. From Bob's point of view, the execution is consistent with an execution of Π with inputs $x = 0, y = 0$ and $\lceil n/2 \rceil$ errors, but it is also consistent with an execution of the protocol with $x = 1, y = 0$ and $\lfloor n/2 \rfloor$ errors. If an error-rate of greater than $1/4$ is allowed, then clearly it is impossible for Bob to tell, which of the two is correct and therefore an error rate of greater than $1/4$ cannot be tolerated by any non-adaptive protocol.

What is important to note is, that this argument goes through even if there exists a CRS, since the CRS is sampled independently from the inputs and does not give Bob any additional information that would allow him to resolve the dilemma. Thus the argument – and therefore Braverman and Rao's lower bound – applies without modification to interactive coding in the CRS model.

It was noted by Ghaffari, Haeupler, and Sudan [50] that the bound from [18] does not apply to *adaptive* protocols, where the parties can decide on the fly which party should speak in the next round. This is because in an adaptive setting, in the scenario described above, Bob could have chosen to yield some of his rounds to Alice once he noticed that messages were inconsistent. Thereby allowing Alice to send messages in more than $r/2$ rounds, thus thwarting the attack and avoiding the dilemma.

However, Ghaffari, Haeupler, and Sudan are able to prove a very similar lower bound, ruling out non-adaptive interactive coding capable of tolerating an error rate of $2/7$. While the proof for this bound is much more elaborate because it needs to deal with the parties' ability to adapt, it still works in essentially the same manner. I.e., it uses tampering to cause transcripts that are explainable with more than one input given the allowed error rate. Again, this bound is completely unaffected by the presence of an input-independent CRS.