Putting the Online Phase on a Diet: Covert Security from Short MACs
(Full Version)

Sebastian Faust[1], Carmit Hazay[2], David Kretzler[1], and Benjamin Schlosser[1]

[1] Technical University of Darmstadt, Germany
{first.last}@tu-darmstadt.de
[2] Bar-Ilan University, Israel
carmit.hazay@biu.ac.il

Abstract. An important research direction in secure multi-party computation (MPC) is to improve the efficiency of the protocol. One idea that has recently received attention is to consider a slightly weaker security model than full malicious security – the so-called setting of *covert security*. In covert security, the adversary may cheat but only is detected with certain probability. Several works in covert security consider the offline/online approach, where during a costly offline phase correlated randomness is computed, which is consumed in a fast online phase. State-of-the-art protocols focus on improving the efficiency by using a covert offline phase, but ignore the online phase. In particular, the online phase is usually assumed to guarantee security against malicious adversaries. In this work, we take a fresh look at the offline/online paradigm in the covert security setting. Our main insight is that by weakening the security of the online phase from malicious to covert, we can gain significant efficiency improvements during the offline phase. Concretely, we demonstrate our technique by applying it to the online phase of the well-known TinyOT protocol (Nielsen et al., CRYPTO '12). The main observation is that by reducing the MAC length in the online phase of TinyOT to t bits, we can guarantee covert security with a detection probability of $1 - \frac{1}{2^t}$. Since the computation carried out by the offline phase depends on the MAC length, shorter MACs result in a more efficient offline phase and thus speed up the overall computation. Our evaluation shows that our approach reduces the communication complexity of the offline protocol by at least 35% for a detection rate up to $\frac{7}{8}$. In addition, we present a new generic composition result for analyzing the security of online/offline protocols in terms of concrete security.

Keywords: Multi-Party Computation (MPC) · Covert Security · Offline/Online · Deterrence Composition

1 Introduction

Secure multi-party computation (MPC) allows a set of distrusting parties to securely compute an arbitrary function on private inputs. While originally MPC was mainly studied by the cryptographic theory community, in recent years many industry applications have been envisioned in areas such as machine learning [KVH+21], databases [VSG+19], blockchains [Zen] and more [ABL+18, MPC]. One of the main challenges for using MPC protocols in practice is their huge overhead in terms of efficiency. Over the last decade, tremendous progress has been made both on the protocol side as well as the engineering level to move MPC protocols closer to practice [DPSZ12, DKL+13, KOS16, KPR18, BCS19, CKR+20, Ors20].

Most efficient MPC protocols work in the honest-but-curious setting. In this setting, the adversary must follow the protocol specification but tries to learn additional information from the interaction with the honest parties. A much stronger security notion is to consider malicious security, where the corrupted parties may arbitrarily deviate from the specification in order to attack the protocol. Unfortunately, however, achieving malicious security is much more challenging and typically results into significant efficiency penalties [KOS16, DILO22].

An attractive middle ground between the efficient honest-but-curious model and the costly malicious setting is *covert security* originally introduced by Aumann and Lindell [AL07]. As in malicious security, the adversary may attack the honest parties by deviating arbitrarily from the protocol specification but may get detected in this process. Hence, in contrast to malicious security such protocols do not prevent cheating, but instead de-incentivize malicious behavior as an adversary may fear getting caught. The latter may lead to reputational damage or financial punishment, which for many real-world settings is a sufficiently strong countermeasure against attacks. Moreover, since covert security does not need to prevent cheating at the protocol level, it can lead to significantly improved efficiency. Let us provide a bit more detail on how to construct covert secure protocols.

The cut-and-choose technique. In a nutshell, all known protocols with covert security amplify the security of a semi-honest protocol by applying the cut-and-choose technique. In this technique, the semi-honest protocol is executed t times where $t-1$ of the executions are checked for correctness via revealing their entire private values. The remaining unchecked instance stays hidden and thus can be used for computing the output. Since in the protocol the $t-1$ checked instances are chosen uniformly at random, any cheating attempt is detected with probability at least $\frac{t-1}{t}$, which is called the *deterrence factor* of the protocol and denoted by ϵ. The overhead of the cut-and-choose approach is roughly a factor t compared to semi-honest protocols due to the execution of t semi-honest instances.

The offline/online paradigm. An important technique to construct efficient MPC protocols is to split the computation in an input independent offline phase and an input dependent online phase. The goal of this approach is to shift the bulk of the computational effort to the offline phase such that once the private inputs become available the evaluation of the function can be done efficiently. To this end, parties pre-compute correlated randomness during the offline phase, which is consumed during the online phase to speed up the computation. Examples for offline/online protocols are SPDZ [DPSZ12], authenticated garbling [WRK17a, WRK17b] and the TinyOT approach [NNOB12, LOS14, BLN$^+$21, FKOS15].

While traditionally the offline/online paradigm has been instantiated either in the honest-but-curious or malicious setting, several recent works have considered how to leverage the offline/online approach to speed-up covert secure protocols [DKL$^+$13, DOS20, FHKS21]. The standard approach is to take a covertly secure offline phase and combine it with a maliciously secure online phase. Since the offline phase is most expensive, this results into a significant efficiency improvement. Moreover, since the offline phase is input independent, it is particularly well suited for the cut-and-choose approach used for constructing covert secure protocols. In contrast to the offline phase, for the online phase we typically rely on a maliciously secure protocol. The common belief is that the main efficiency bottleneck is the offline phase, and hence optimizing the online phase to achieve covert security (which is also more challenging since we need to deal with the private inputs) is of little value. In our work, we challenge this belief and study the following question:

Can we improve the overall efficiency of a covertly secure offline/online protocol by relaxing the security of the online phase to covert security?

1.1 Contribution

Our main contribution is to answer the above question in the affirmative. Concretely, we show that significant efficiency improvements are possible by switching form a maliciously secure online phase to covert security.

To this end, we introduce a new paradigm to achieve covert security. Instead of amplifying semi-honest security using cut-and-choose, we start with a maliciously secure protocol and weaken its security. In malicious security, successful cheating of the adversary is only possible with negligible probability in the statistical security parameter. For protocol instantiations, this parameter is typically set to 40. The core idea is to show that in the setting of covert security, we can significantly reduce the value of the statistical security parameter *without* losing in security. We are the first to describe this new method of achieving covert security by weakening malicious security.

For achieving covert security of already efficient online protocols, the naive cut-and-choose approach is not a viable option due to its inherent overhead. In contrast, our approach is particularly interesting for these protocols. In addition, we observe that for several offline/online protocols, a reduction to covert security in the online phase reduces the amount of precomputation required. This results in an overall improved efficiency.

To illustrate the benefits of our paradigm, we apply it to the well-known TinyOT [NNOB12] protocol for two-party computation for boolean circuits based on the secret-sharing approach. This protocol is a good benchmark for oblivious transfer (OT)-based protocols and hasn't been considered before for the covert setting. The original TinyOT protocol consists of a maliciously secure offline and online phase where MACs ensure the correctness of the computation performed during the online phase. While the efficiency of the offline phase can be improved by making this phase covertly secure using the cut-and-choose approach, we apply our paradigm to the online phase to gain additional efficiency improvements. Our insight is that instead of using 40-bit MACs, which is typically done for an actively secure online phase, using t-bits MACs results in a covertly secure online phase with deterrence factor $1 - \frac{1}{2^t}$. We formally prove the covert security of this online protocol.

As touched on earlier, shortening the MAC length of the TinyOT online phase has a direct impact on the computation overhead carried out in the offline phase. In particular, the size of the oblivious transfers that need to be performed depend on the MAC length and thus this number can be reduced. Concretely, we compare the communication complexity of a cut-and-choose-based offline phase for different choices of MAC lengths. We can show that the communication complexity of the offline protocol reduces by at least 35% for a deterrence factor up to $\frac{7}{8}$.

While we chose the TinyOT protocol for demonstrating our new paradigm, we can apply our techniques also for other offline/online protocols in the two- and multi-party case, e.g., [LOS14, BLN$^+$21, FKOS15, WRK17a, WRK17b].

As a second major technical contribution, we show that the combination of a covert offline and covert online phase achieves the same deterrence factor as a covert offline phase combined with an active online phase. We show this result in a generic way by presenting a *deterrence replacement theorem*. Intuitively, when composing a covertly secure offline phase with a covertly secure online phase, the deterrence factor of the composed protocol needs to consider the worst deterrence of both phases. This is easy to see, since the adversary can always try to cheat in that phase where the detection probability is smaller. While easy at first sight, the formalization requires a careful analysis and adds restrictions on the class of protocols for which such composition can be shown. By applying our deterrence replacement theorem, we show for offline/online protocols that the overall detection probability is computed as the minimum of the detection probability of the offline phase and the detection probability of the online phase.

While this result was proven by Aumann and Lindell [AL07] for a weak notion of covert security, the *failed-simulation formulation*, we are the first to formally present a proof in the strongest setting of covert security which is also mostly used in the literature. The definitional framework of the failed-simulation formulation and the one of all of the stronger notions are fundamentally different. In particular, the failed-simulation formulation relies on the ideal functionality defined for the malicious setting but allows for failed simulations. The stronger notions define a covert ideal functionality explicitly capturing the properties of the covert setting, i.e., the possible cheating attempts of the adversary. For this reason, it is not straightforward to translate the proof techniques from the failed-simulation formulation to the stronger notions.

1.2 Related Work

Short MACs. Hazay et al. [HOSS18] also considered short MAC keys for TinyOT, but in the context of concretely efficient large-scale MPC in the active security setting with a minority of honest parties. The main idea of their work is to distribute secret key material between all parties such that the security is based on the concatenation of all honest parties' keys. In contrast, we achieve more efficient covert security and the security is based on each party's individual key.

TinyOT extensions. In the two-party setting, the TinyOT protocol is extended by the TinyTables [DNNR17] and the MiniMac [DZ13] protocols. The former improves the online communication complexity by relying on precomputated scrambled truth tables. The precomputation of these works is based on the offline phase of TinyOT. Therefore, we believe that our techniques can be applied to the TinyTables protocol as well. We focus in our description on the original TinyOT protocol to simplify presentation.

The MiniMac protocol uses error correcting codes for authentication of bit vectors and is in particular interesting for "well-formed" circuits that allow for parallelization of computation. The sketched precomputation of MiniMac is based on the SPDZ-precomputation [DPSZ12]. In the SPDZ protocol, MACs represent field elements instead of binary strings as in TinyOT. Therefore, it is not straight-forward to apply our techniques to the MiniMac protocol. We leave it as an open question if our techniques can be adapted to this setting.

Larraia et al. and Burra et al. [LOS14, BLN$^+$21] show how to extend TinyOT to the multi-party setting. Our paradigm can be applied to these protocols as well as to the precomputation of [FKOS15].

Authenticated garbling. The authenticated garbling protocols [WRK17a, WRK17b, KRRW18, YWZ20] achieve constant round complexity and active security by utilizing an authenticated garbled circuit. For authentication, the protocols rely on a TinyOT-style offline phase. Hence, we believe that our approach can improve the efficiency of the authenticated garbling protocols as well (when moving to the setting of covert security).

Arithmetic computation. The family of SPDZ protocols [DPSZ12, DKL$^+$13, KOS16, KPR18, CDE$^+$18] provide means to perform multi-party computation with active security on arithmetic circuits. Damgård et al. [DKL$^+$13] have already considered the covert setting but only reduced the security of the offline phase to covert security. As already mentioned above in the context of MiniMac, we leave it as an interesting open question to investigate if our approach can be translated to the arithmetic setting of the SPDZ family in which MACs are represented as field elements.

Pseudorandom Correlation Generators. Recently, pseudorandom correlation generators (PCGs) were presented to compute correlated randomness with sublinear communication [BCG$^+$19, BCG$^+$20a, BCG$^+$20b]. While this is a promising approach, efficient constructions are based on variants of the learning parity with noise (LPN) assumption. These assumptions are still not fully understood, especially compared to oblivious transfer which is the base of TinyOT.

1.3 Technical Overview

Notions of covert security. The notion of *covert security with ϵ-deterrence factor* was proposed by Aumann and Lindell in 2007 [AL07], who introduced a hierarchy of three different variants. The weakest variant is called the *failed-simulation formulation*, the next stronger is the *explicit cheat formulation (ECF)* and the strongest variant is the *strong explicit cheat formulation (SECF)*. The last is also the most widely used variant of covert security. In the failed-simulation formulation, the adversary is able to cheat depending on the honest parties' inputs. This undesirable behavior is prevented in the stronger variants. In the ECF notion, the adversary learns the inputs of the honest parties even if cheating is detected. Finally, SECF prevents the adversary from learning anything in case cheating is detected.

In this work, we introduce on a new notion that lies between ECF and SECF. We call it *intermediate explicit cheat formulation (IECF)* (cf. Section 2), where we let the adversary learn the outputs of the corrupted parties even if cheating is detected. This is a strictly stronger security guarantee than ECF, where the adversary also learns the inputs of the honest parties. Our new notion captures protocols where an adversary learns its own outputs (which may depend on honest parties inputs) before the honest parties detect cheating. However, we emphasize that the adversary cannot prevent detection by the honest parties. In particular, it must make its decision on whether to cheat or not before learning its outputs. Moreover, notice that in case when the adversary does not cheat, it would anyway learn these outputs, and hence IECF is only a very mild relaxation of the SECF notion.

Composition of covert protocol. Composition theorems allow to modularize security proofs of protocols and thus are tremendously useful for protocol design. Aumann and Lindell presented two sequential composition theorems for protocols in the covert security model [AL07]. One for the failed-simulation formulation and one for the (S)ECF. In the following, we focus on the later theorem since these notions are closer to the IECF notion. The composition theorem presented in [AL07] allows to analyze the security of a protocol in a hybrid model where the parties have access to hybrid functionalities. In more detail, the theorem states that a protocol that is covertly secure with deterrence factor ϵ in a hybrid model where parties have access to a polynomial number of functionalities, which themselves have deterrence factors, then the protocol is also secure if the hybrid functionalities are replaced with protocols realizing the functionalities with the corresponding deterrence values. Note that the theorem states that a composed protocol using subprotocols instead of hybrid functionalities has the same deterrence factor as when analyzed with (idealized) hybrid functionalities.

Aumann and Lindell's theorem is very useful to show security of a complex protocol. Unfortunately, however, the theorem of Aumann and Lindell does not make any statement how the deterrence factor of hybrid functionalities influences the deterrence factor of the overall protocol. Instead, the deterrence factor of the overall protocol has to be determined depending on the concrete deterrence factors of the hybrid functionalities. We are looking for a composition theorem that goes one step further. In particular, we develop a theorem that allows to analyze a protocol's security and its deterrence factor in a simple model where no successful cheating in hybrid functionalities is possible, i.e., a deterrence factor of $\epsilon = 1$. Then, the theorem should help deriving the deterrence factor of the composed protocol when cheating in hybrid functionalities is possible with a fixed probability, i.e., $\epsilon < 1$.

Deterrence replacement theorem. Our deterrence replacement theorem fills the aforementioned gap (cf. Section 3). Let Hy_1 and Hy_2 be two hybrid worlds. In Hy_1 an offline functionality exists with deterrence factor 1. In Hy_2 the same offline functionality has deterrence factor $\epsilon^*_{\mathsf{off}}$. Our theorem states that a protocol, which is covertly secure with deterrence factor ϵ_{on} in Hy_1, is covertly secure with deterrence factor $\epsilon^*_{\mathsf{on}} := \mathsf{Min}(\epsilon_{\mathsf{on}}, \epsilon^*_{\mathsf{off}})$ in Hy_2. While we have to impose some restrictions on the protocols that our theorem can be applied on, practical offline/online protocols [DPSZ12, NNOB12, WRK17a, WRK17b] fulfill these restrictions or can easily be adapted to do so. The main benefit of our theorem is to simplify the analysis of a protocol's security by enabling the analysis in a model where successful cheating in the offline functionality does not occur. In addition, our theorem implies that the deterrence factor of the online phase can be as low as the deterrence factor of the offline phase without any security loss.

Achieving covert security. Most covertly secure protocols work by taking a semi-honest secure protocol and applying the cut-and-choose technique. In contrast, we present a new approach to achieve covert security where instead of amplifying semi-honest security, we downgrade malicious security. Our core idea is to obtain covert security by reducing the statistical security parameter of a malicious protocol.

As highlighted in the contribution, reducing the security of the online phase to covert has the potential to improve the efficiency of the overall protocol execution. This improvement does not come from a speed-up in the online phase, in fact the online phase can become slightly less efficient, but from lower requirements on the offline phase. Using the cut-and-choose approach to get a covertly secure online phase incurs an overhead to the semi-honest protocol that is linear in the number of executed instances. This overhead might exceed the efficiency gap between the semi-honest and the malicious protocol rendering the cut-and-choose-based covert offline phase significantly less efficient than the malicious online phase. In this case, the overhead of the online phase can vanish the gains of the faster offline phase. In contrast, our approach comes with a small constant overhead to the malicious protocol such that the overall efficiency gain is preserved. This makes our approach particularly interesting for actively secure protocols that are already very efficient such as information-theoretic online protocols, e.g., the online phase of TinyOT [NNOB12].

The TinyOT protocol. We illustrate the benefit of our new paradigm for achieving covert security by applying it to the maliciously secure online phase of TinyOT [NNOB12]. We start with a high-level overview of TinyOT.

The TinyOT protocol is a generic framework for computing Boolean circuits based on the secret sharing paradigm for two-party computation. The protocol splits the computation into an offline and an online phase. In the offline phase, the parties compute authenticated bits and authenticated triples. For instance, the authentication of a bit x known to a party $\mathcal{A}$ is achieved by having the other party $\mathcal{B}$ hold a global key $\Delta_{\mathcal{B}}$, a random t-bit key $K[x]$, and having $\mathcal{A}$ hold the bit x and a t-bit MAC $M[x] = K[x] \oplus x \cdot \Delta_{\mathcal{B}}$. In the online phase, parties evaluate the circuit with secret-shared wire values where each share is authenticated given the precomputed data. Due to the additive homomorphism of the MACs, addition gates can be computed non-interactively. For each multiplication gate, the parties interactively compute the results by consuming a precomputed multiplication triple. At the end of the circuit evaluation, a party learns its output, i.e., the value of an output wire, by receiving the other party's share on that wire. The correct behavior of all parties is verified by checking the MACs on the output wire shares.

Covert online protocol. The authors of TinyOT showed that successfully breaking security of the online phase is equivalent to guessing the global MAC key of the other party. In this work, we translate this insight to the covert setting. In particular, we show that the online phase of a TinyOT-like protocol with a reduced MAC length of t-bits implements covert security with a deterrence factor of $1 - (\frac{1}{2})^t$ (cf. Section 4).

The resulting protocol can be modified with small adjustments to achieve all known notions of covert security. In particular, the unmodified version of TinyOT implements a variant of covert security in which the adversary learns the output of the protocol, and, only then, decides on its cheating attempt. We achieve the IECF, i.e., the notion in which the adversary always learns the output of the corrupted parties, even in case of detected cheating, by committing to the outputs bits and MACs before opening them. Due to the commitments, the adversary needs to decide first if it wants to cheat and only afterwards it learns the output. However, since the adversary receives the opening on the commitment of the honest party first, it learns the output even if it committed to incorrect values or refuses to open its commitment, both of which are considered cheating. Finally, in order to achieve the SECF, we have to prevent the adversary from inserting incorrect values into the commitment. We can do so by generating the commitments as part of the function whose circuit is evaluated. Only after the parties checked both, correct behavior throughout the evaluation and correctness of the received outputs, i.e., the commitments, the parties exchange the openings of the commitments. This way, we ensure that the adversary only receives its output if it behaved honestly or cheated successfully which fulfills the SECF.

In this work, we focus on the IECF. On one hand, we assess the IECF to constitutes a minor loss of security compared to the SECF. This is due to the fact that we are in the security-with-abort setting, implying that the honest parties already approve the risk of giving the adversary its output while not getting an output themselves. On the other hand, the efficiency overhead of the IECF compared to the weaker variant of covert achieved by the unmodified protocol just consists out of a single commit-and-opening step accounting for 48 bytes per party (if instantiated via a hash function and with 128 bit security). In contrast, the SECF requires generating the commitments as part of the circuit which incurs a much higher efficiency overhead. Therefore, we assess the protocol achieving the IECF notion to depict a much better trade-off between efficiency overhead and security loss than the other notions.

Evaluation. Our result shows that we can safely reduce the security level of the online phase without compromising on the security of the overall protocol. As we show in the evaluation section (cf. Section 5), this improves the efficiency of the overall protocol. Concretely, the main improvements come from savings during the offline phase since using our techniques the online phase gets less demanding by relying on shorter MACs. We quantify these improvements by evaluating the communication complexity of the offline phase depending on the length of the generated MACs. More precisely, when using an actively secure online phase, the MAC length needs to be 40 Bits, while for achieving covert security, we can set the length of the MACs to a significantly lower value t. This results into a deterrence factor of $1 - \frac{1}{2^t}$. Our evaluation shows that we can reduce the communication complexity of the offline protocol by at least 35% for a deterrence factor of up to $\frac{7}{8}$.

2 Covert Security

A high-level comparison between the notions of covert security presented by Aumann and Lindell [AL07] is stated in Section 1.3. Next, we present details about the *explicit cheat formulation (ECF)* and the *strong explicit cheat formulation (SECF)*. Afterwards, we present our new notion which lies strictly between the ECF and the SECF.

The ECF and the SECF consider an ideal functionality where the adversary explicitly sends a cheat_i command for the index i of a corrupted party to the functionality which then decides if cheating is detected with probability ϵ. In the ECF, the adversary learns the honest parties' inputs even if cheating is detected, which is prevented by the SECF. In addition, the adversary can also send a $\mathsf{corrupted}_i$ or abort_i command, which is forwarded to the honest parties. The $\mathsf{corrupted}_i$ command models a blatant cheat option, where the adversary cheats in a way that will always be detected, and the abort_i command models an abort of a corrupted party. Later, Faust et al. [FHKS21] proposed to extract the *identifiable abort* property as it can be considered orthogonal and of independent interest (cf. [IOZ14]). For the covert notion, this means that if a corrupted party aborts, the ideal functionality only sends abort to the honest parties instead of abort_i for i being the index of the aborting party.

In the following, we present a new notion for covert security called the *intermediate explicit cheat formulation (IECF)*. We follow the approach of [FHKS21] and present our notion without the identifiable abort property. In addition, we clean up the definition by merging the blatant cheat option, where cheating is always detected, with the cheat attempt that is only detected with a fixed probability. To this end, if the adversary sends the cheat-command, we allow the adversary to specify any detection probability between the deterrence factor and 1. Furthermore, we enable the adversary to force a cheating detection or abort even if the ideal functionality signals undetected cheating. This additional action does not provide further benefit to the adversary and thus does not harm the security provided by our notion. Since the decision solely depends on the adversary, the change also does not restrict the adversary.

Finally, and most important, our notion allows the adversary to learn the outputs of the corrupted parties but nothing else if cheating is detected. Therefore, it lies between the ECF, where the adversary learns the inputs of all parties even if cheating is detected, and the SECF, where the adversary learns nothing if cheating is detected. Since our notion is strictly between the ECF and the SECF, we call it the IECF.

Next, we present the IECF in full details in the following and state the difference to the SECF afterwards.

Intermediate explicit cheat formulation. As in the standalone model, the notions are defined in the real world/ideal world paradigm. This means, the security of a protocol is shown by comparing the real-world execution with an ideal-world execution. In the *real world*, the parties jointly compute the desired function f using a protocol π. Let n be the number of parties and let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where $f = (f_1, \ldots, f_n)$ is the function computed by π. We define for every vector of inputs $\bar{x} = (x_1, \ldots, x_n)$ the vector of outputs $\bar{y} = (f_1(\bar{x}), \ldots, f_n(\bar{x}))$ where party P_i with input x_i obtains the output $f_i(\bar{x})$. During the execution of π, the adversary Adv can corrupt a subset $\mathcal{I} \subset [n]$ of all parties. We define $\mathsf{REAL}_{\pi,\mathsf{Adv}(z),\mathcal{I}}(\bar{x}, 1^\kappa)$ as the output of the protocol execution π on input $\bar{x} = (x_1, \ldots, x_n)$ and security parameter κ, where Adv on auxiliary input z corrupts parties $\mathcal{I}$. We further specify $\mathsf{OUTPUT}_i(\mathsf{REAL}_{\pi,\mathsf{Adv}(z),\mathcal{I}}(\bar{x}, 1^\kappa))$ to be the output of party P_i for $i \in [n]$.

In contrast, in the *ideal world*, the parties send their inputs to the uncorruptible ideal functionality $\mathcal{F}$ which computes function f and returns the result. Hence, the computation in the ideal world is correct by definition. The security of π is analyzed by comparing the ideal-world execution with the real-world execution. The ideal world in covert security is slightly changed compared to the standard model of secure computation. In particular, in covert security, the ideal world allows the adversary to cheat, and cheating is detected at least with some fixed probability ϵ which is called the *deterrence factor*. Let $\epsilon : \mathbb{N} \to [0,1]$ be a function. The execution in the ideal world in our *IECF* notion is defined as follows:

Inputs: Each party obtains an input, where the i^{th} party's input is denoted by x_i. We assume that all inputs are of the same length and call the vector $\bar{x} = (x_1, \ldots, x_n)$ balanced in this case. The adversary receives an auxiliary input z. In case there is no input, the parties will receive $x_i = \mathsf{ok}$.

Send inputs to ideal functionality: Any honest party P_j sends its received input x_j to the ideal functionality. The corrupted parties, controlled by ideal world adversary $\mathcal{S}$, may either send their received input, or send some other input of the same length to the ideal functionality. This decision is made by $\mathcal{S}$ and may depend on the values x_i for $i \in \mathcal{I}$ and the auxiliary input z. Denote the vector of inputs sent to the ideal functionality by $\bar{x}$. In addition, $\mathcal{S}$ can send a special cheat or abort message w.

Abort options: If $\mathcal{S}$ sends $w = \mathsf{abort}$ to the ideal functionality as its input, then the ideal functionality sends abort to all honest parties and halts.

Attempted cheat option: If $\mathcal{S}$ sends $w = (\mathsf{cheat}_i, \epsilon_i)$ for $i \in \mathcal{I}$ and $\epsilon_i \geq \epsilon$, the ideal functionality proceeds as follows:

1. With probability ϵ_i, the ideal functionality sends $\mathsf{corrupted}_i$ to all honest parties. In addition, the ideal functionality computes $(y_1, \ldots, y_n) = f(\bar{x})$ and sends $(\mathsf{corrupted}_i, \{y_j\}_{j \in \mathcal{I}})$ to $\mathcal{S}$.
2. With probability $1 - \epsilon_i$, the ideal functionality sends $\mathsf{undetected}$ to $\mathcal{S}$ along with the honest parties' inputs $\{x_j\}_{j \notin \mathcal{I}}$. Then, $\mathcal{S}$ sends output values $\{y_j\}_{j \notin \mathcal{I}}$ of its choice for the honest parties to the ideal functionality. Then, for every $j \notin \mathcal{I}$, the ideal functionality sends y_j to P_j. The adversary may also send abort or $\mathsf{corrupted}_i$ for $i \in \mathcal{I}$, in which case the ideal functionality sends abort or $\mathsf{corrupted}_i$ to every P_j for $j \notin \mathcal{I}$.

The ideal execution ends at this point. Otherwise, if no w equals abort or $(\mathsf{cheat}_i, \cdot)$ the ideal execution proceeds as follows.

Ideal functionality answers adversary: The ideal functionality computes $(y_1, \ldots, y_n) = f(\bar{x})$ and sends y_i to $\mathcal{S}$ for all $i \in I$.

Ideal functionality answers honest parties: After receiving its outputs, the adversary sends abort, $\mathsf{corrupted}_i$ for some $i \in \mathcal{I}$, or $\mathsf{continue}$ to the ideal functionality. If the ideal functionality receives $\mathsf{continue}$ then it sends y_j to all honest parties P_j ($j \notin \mathcal{I}$). Otherwise, if it receives abort resp. $\mathsf{corrupted}_i$, it sends abort resp. $\mathsf{corrupted}_i$ to all honest parties.

Outputs: An honest party always outputs the message it obtained from the ideal functionality. The corrupted parties output nothing. The adversary $\mathcal{S}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial inputs $\{x_i\}_{i \in \mathcal{I}}$, the auxiliary input z, and the messages obtained from the ideal functionality.

We denote by $\mathsf{IDEALC}^{\epsilon}_{f, \mathcal{S}(z), \mathcal{I}}(\bar{x}, 1^{\kappa})$ the output of the honest parties and the adversary in the execution of the ideal model as defined above, where $\bar{x}$ is the input vector and the adversary $\mathcal{S}$ runs on auxiliary input z.

Definition 1 (Covert security - intermediate explicit cheat formulation). *Let f, π, and ϵ be as above. A protocol π* securely computes f in the presence of covert adversaries with ϵ-deterrence *if for every non-uniform probabilistic polynomial-time adversary* Adv *in the real world, there exists a non-uniform probabilistic polynomial-time adversary* $\mathcal{S}$ *for the ideal model such that for every* $\mathcal{I} \subseteq [n]$*, every balanced vector* $\bar{x} \in (\{0,1\}^*)^n$*, and every auxiliary input* $z \in \{0,1\}^*$*:*

$$\{\mathsf{IDEALC}^{\epsilon}_{f,\mathcal{S}(z),\mathcal{I}}(\bar{x}, 1^{\kappa})\}_{\kappa \in \mathbb{N}} \overset{c}{\equiv} \{\mathsf{REAL}_{\pi,\mathsf{Adv}(z),\mathcal{I}}(\bar{x}, 1^{\kappa})\}_{\kappa \in \mathbb{N}}$$

The *SECF* notions follows the IECF notion with one single change. Instead of sending $(\mathsf{corrupted}_i, \{y_j\}_{j \in \mathcal{I}})$ to $\mathcal{S}$ in case of detected cheating, the ideal functionality only sends $(\mathsf{corrupted}_i)$. This means that in the SECF the ideal adversary does not learn the output of corrupted parties in case cheating is detected.

3 Offline/Online Deterrence Replacement

Offline/online protocols split the computation of a function f into two parts. In the offline phase, the parties compute correlated randomness independent of the actual inputs to f. In the online phase, the function f is computed on the private inputs of all parties while the correlated randomness from the offline phase is consumed to accelerate the execution. When considering covert security, the adversary may cheat in both the offline and the online phase. The cheating detection probability might differ in these two phases. Intuitively, the deterrence factor of the overall protocol needs to consider the worst-case detection probability. This is easy to see, since the adversary can always choose to cheat during that phase where the detection probability is smaller.

While the above is easy to see at a high level, the outlined intuition needs to be formally modeled and proven. We take the approach of describing offline/online protocols within a hybrid model. This means, the offline phase is formalized as a hybrid functionality to which the adversary can signal a cheat attempt. This hybrid functionality is utilized by the online protocol during which the adversary can cheat, too. We formally describe the hybrid model in Section 3.1.

Next, we present our offline/online deterrence replacement theorem in Section 3.2. Let π_{on} be an online protocol that is covertly secure with deterrence factor ϵ_{on} while any cheat attempt during the offline phase is detected with probability $\epsilon_{\mathsf{off}} = 1$[3]. Then, our theorem shows that if the detection probability during the offline phase is reduced to $\epsilon'_{\mathsf{off}} < 1$, π_{on} is also covertly secure with a deterrence factor of $\epsilon'_{\mathsf{on}} = \min(\epsilon_{\mathsf{on}}, \epsilon'_{\mathsf{off}})$. This means, the new deterrence factor is the minimum of the detection probability of the old online protocol, in which successful cheating during the offline phase is not possible, and the detection probability of the new offline phase. Intuitively, our theorem quantifies the effect on the deterrence factor of the online protocol when replacing the deterrence

[3] Covert security with deterrence factor 1 can be realized by a maliciously secure protocol as shown by Asharov and Orlandi [AO12].

factor of the offline hybrid functionality with a different value. This is why we call Theorem 1 the deterrence replacement theorem.

The main purpose of our theorem is to allow the analysis of the security of an online protocol in a simple setting where $\epsilon_{\mathsf{off}} = 1$. Since in this setting cheating during the offline phase is always detected, the security analysis and the calculation of the online deterrence factor ϵ_{on} are much simpler. Once the security of π_{on} has been proven in the hybrid world, in which the offline phase is associated with deterrence factor 1, and ϵ_{on} has been determined, our theorem allows to derive security of π_{on} in the hybrid world, in which the offline phase is associated with deterrence factor ϵ'_{off}, and determines the deterrence factor to be $\epsilon'_{\mathsf{on}} = \min(\epsilon'_{\mathsf{off}}, \epsilon_{\mathsf{on}})$.

While the effect of deterrence replacement was already analyzed by Aumann and Lindell [AL07] for a weak variant of covert security, we are the first to consider deterrence replacement in a widely adopted and strong variant of covert security. We discuss the relation to [AL07] in Appendix C.

3.1 The Hybrid Model

We consider a hybrid model to formalize the execution of offline/online protocols. Within such a model, parties exchange messages between each other but also have access to hybrid functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$. These hybrid functionalities work like trusted parties to compute specified functions. The hybrid model is thus a combination of the real model, in which parties exchange messages according to the protocol description, and the ideal model, in which parties have access to an idealized functionality.

A protocol in a hybrid model consists of standard messages sent between the parties and calls to the hybrid functionalities. These calls instruct the parties to send inputs to the hybrid functionality, which delivers back the output according to its specification. After receiving the outputs from the hybrid functionality, the parties continue the execution of the protocol. When instructed to send an input to the hybrid functionality, all honest parties follow this instruction and wait for the return value before continuing the protocol execution.

The interface provided by a hybrid functionality depends on the security model under consideration. Since we deal with covert security, the adversary is allowed to send special commands, e.g., cheat, to the hybrid functionality. In case the functionality receives cheat from the adversary, the functionality throws a coin to determine whether or not the cheat attempt will be detected by the honest parties. The detection probability is defined by the deterrence factor of this functionality. We use the notation $\mathcal{F}_f^\epsilon$ to denote a hybrid functionality computing function f with deterrence factor ϵ. The notation of a $(\mathcal{F}_{f_1}^{\epsilon_1}, \ldots, \mathcal{F}_{f_\ell}^{\epsilon_\ell})$-hybrid model specifies the hybrid functionalities accessible by the parties.

The hybrid model technique is useful to modularize security proofs. Classical composition theorems for passive and active security [Can00] as well as for covert security [AL07] build the foundation for this proof technique. Informally, these theorems state that if a protocol π is secure in the hybrid model where the parties use a functionality $\mathcal{F}_f$ and there exists a protocol ρ that securely realizes $\mathcal{F}_f$, then the protocol π is also secure in a model where $\mathcal{F}_f$ is replaced with ρ.

3.2 Our Theorem

We start by assuming an online protocol π_{on} that realizes an online functionality $\mathcal{F}_{f_{\mathsf{on}}}^{\epsilon_{\mathsf{on}}}$ in the $\mathcal{F}_{f_{\mathsf{off}}}^{1}$-hybrid world. This means the deterrence factor of π_{on} is ϵ_{on} and the deterrence factor of the offline functionality is 1 which means that every cheating attempt in the offline phase will be detected. Next, our theorem states that replacing the deterrence factor 1 of the offline hybrid functionality with any $\epsilon'_{\mathsf{off}} \in [0,1]$ results in a deterrence factor of the online protocol of $\epsilon'_{\mathsf{on}} = \min(\epsilon_{\mathsf{on}}, \epsilon'_{\mathsf{off}})$, i.e., the minimum of the previous deterrence factor of the online protocol and the new deterrence of the offline hybrid functionality.

Formally, we model the composition of an offline and an online phase via the hybrid model. Let $f_{\mathsf{off}} : (\{\perp\}_{j \notin \mathcal{I}}, \{x_i^{\mathsf{off}}\}_{i \in \mathcal{I}}) \to (y_1^{\mathsf{off}}, \ldots, y_n^{\mathsf{off}})$ be an n-party probabilistic polynomial-time function representing the offline phase, where $\mathcal{I}$ denotes the set of corrupted parties. We model the offline functionality in such a way that the honest parties provide no input, the adversary may choose the randomness used by the corrupted parties and the functionality produces outputs which depend on the randomness of the corrupted parties and further random choices. The n-party probabilistic polynomial-time online function is denoted by $f_{\mathsf{on}} : (x_1, \ldots, x_n) \to (y_1^{\mathsf{on}}, \ldots, y_n^{\mathsf{on}})$. We use the abbreviation $\mathcal{F}_{\mathsf{off}}^{\epsilon_{\mathsf{off}}}$ and $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ for $\mathcal{F}_{f_{\mathsf{off}}}^{\epsilon_{\mathsf{off}}}$ and $\mathcal{F}_{f_{\mathsf{on}}}^{\epsilon_{\mathsf{on}}}$.

Our composition theorem puts some restrictions on the online protocol π_{on} that we list below and discuss in more technical depth in Appendix A. First, we require that $\mathcal{F}_{\mathsf{off}}^{\epsilon}$ is called only once during the execution of π_{on} and this call happens at the beginning of the protocol before any other messages are exchanged. Second, we require that if $\mathcal{F}_{\mathsf{off}}^{\epsilon}$ returns $\mathsf{corrupted}_i$ to the parties, then π_{on} instructs the parties to output $\mathsf{corrupted}_i$. Practical offline/online protocols [DPSZ12, NNOB12, WRK17a, WRK17b] either directly fulfill theses requirements or can easily be adapted to do so. We are now ready to formally state our deterrence replacement theorem.

Theorem 1 (Deterrence replacement theorem). *Let f_{off} and f_{on} be n-party probabilistic polynomial-time functions and π_{on} be a protocol that securely realizes $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid model according to Definition 1, where f_{off}, f_{on} and π_{on} are defined as above. Then, π_{on} securely realizes $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid model according to Definition 1, where $\epsilon'_{\mathsf{on}} = \min(\epsilon_{\mathsf{on}}, \epsilon'_{\mathsf{off}})$.*

Remarks. Our theorem focuses on the offline/online setting where only a single hybrid functionality is present. Nevertheless, it can be extended to use additional hybrid functionalities with fixed deterrence factors. In addition, we present our theorem for the intermediate explicit cheat formulation to match the definition given in Section 2. We emphasize that our theorem is also applicable to the strong explicit cheat formulation. For this variant of covert security, our theorem can also be extended to consider an offline hybrid functionality that takes inputs from all parties, in contrast to the definition of the offline function we specified above.

Proof sketch. We present a proof sketch together with the simulator here and defer the full indistinguishability proof to Appendix B.

On a high level, we prove our theorem by constructing a simulator $\mathcal{S}$ for the protocol π_{on} in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world. In the construction, we exploit the fact that π_{on} is covertly secure in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world with deterrence factor ϵ_{on}, which means that a simulator $\mathcal{S}_1$ for the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$-ideal world exists. Next, we state the full simulator description.

0. Initially, $\mathcal{S}$ calls $\mathcal{S}_1$ to obtain a random tape used for the execution of Adv.
1. In the first step, $\mathcal{S}$ receives the messages sent from Adv to $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$, i.e., a set of inputs for the corrupted parties $\{x_i^{\mathsf{off}}\}_{i\in\mathcal{I}}$ together with additional input from the adversary $m \in \{\bot, \mathsf{abort}, (\mathsf{cheat}_i, \epsilon_i)\}$, where $i \in \mathcal{I}$ and $\epsilon_i \geq \epsilon'_{\mathsf{off}}$. $\mathcal{S}$ distinguishes the following cases:
   (a) If $m \in \{\bot, \mathsf{abort}\}$, $\mathcal{S}$ sends $\{x_i^{\mathsf{off}}\}_{i\in\mathcal{I}}$ and m to $\mathcal{S}_1$ and continues the execution exactly as $\mathcal{S}_1$. The latter is done by forwarding all messages received from $\mathcal{S}_1$ to Adv or $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and vice versa.
   (b) If $m = (\mathsf{cheat}_\ell, \epsilon_\ell)$ for some $\ell \in \mathcal{I}$, $\mathcal{S}$ samples dummy inputs $\{\hat{x}_i^{\mathsf{on}}\}_{i\in\mathcal{I}}$ for the corrupted parties, sends $\{\hat{x}_i^{\mathsf{on}}\}_{i\in\mathcal{I}}$ together with $(\mathsf{cheat}_\ell, \epsilon_\ell)$ to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and distinguishes the following cases:
      i. If $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ replies $(\mathsf{corrupted}_i, \{\hat{y}_i^{\mathsf{on}}\}_{i\in\mathcal{I}})$, $\mathcal{S}$ computes the probabilistic function $f_{\mathsf{off}} : (\{\bot\}_{i\notin\mathcal{I}}, \{x_i^{\mathsf{off}}\}_{i\in\mathcal{I}}) \to (\hat{y}_1^{\mathsf{off}}, \ldots, \hat{y}_n^{\mathsf{off}})$ using fresh randomness, sends $(\mathsf{corrupted}_i, \{\hat{y}_i^{\mathsf{off}}\}_{i\in\mathcal{I}})$ to Adv and returns whatever Adv returns.
      ii. Otherwise, if $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ replies $(\mathsf{undetected}, \{x_j^{\mathsf{on}}\}_{j\notin\mathcal{I}})$, $\mathcal{S}$ sends $\mathsf{undetected}$ to Adv and gets back the value y defined as follows:
         – If $y \in \{\mathsf{abort}, \mathsf{corrupted}_\ell\}$ for $\ell \in \mathcal{I}$, $\mathcal{S}$ sends y to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and returns whatever Adv returns.
         – If $y = \{y_j^{\mathsf{off}}\}_{j\notin\mathcal{I}}$ with $y_j^{\mathsf{off}} \in \{0,1\}^*$ for $j \notin \mathcal{I}$, $\mathcal{S}$ interacts with Adv to simulate the rest of the protocol. To this end, $\mathcal{S}$ takes x_j^{on} as the input of the honest party P_j and y_j^{off} as P_j's output of the offline phase for every $j \notin \mathcal{I}$. When the protocol ends with an honest party's output y_j^{on} for $j \notin \mathcal{I}$, $\mathcal{S}$ forwards these outputs to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and returns whatever Adv returns. Note that y_j^{on} can also be abort or $\mathsf{corrupted}_\ell$ for $\ell \in \mathcal{I}$.

Recall that due to first restriction on π_{on}, the call to the hybrid functionality $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ is the first message sent in the protocol. Via this message, the adversary Adv decides if it sends cheat to the hybrid functionality or not. Since this message is the first one, the cheat decision depends only on the adversary's code and its random tape. The cheat decision is equally distributed in the hybrid and the ideal world, as it depends only on the random tape and input of Adv which is the same in the ideal world and in the hybrid world.

In the ideal world, the hybrid functionality is simulated by the simulator $\mathcal{S}$ and hence $\mathcal{S}$ gets the message of Adv. Depending on Adv's decision to cheat, $\mathcal{S}$ distinguishes between two cases.

On the one hand, in case the adversary *does not cheat*, $\mathcal{S}$ internally runs $\mathcal{S}_1$ for the remaining simulation. Since the case of no cheating might also appear in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world, $\mathcal{S}_1$ is able to produce an indistinguishable view in the ideal world. We formally show via a reduction to the assumption that π_{on} is covertly secure in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world that the views are indistinguishable in this case.

On the other hand, in case the adversary *tries to cheat*, $\mathcal{S}$ cannot use $\mathcal{S}_1$. This is due to the fact that the scenario of undetected cheating can occur in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world, while it cannot happen in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world. Thus, $\mathcal{S}$ needs to be able to simulate undetected cheating which is not required from $\mathcal{S}_1$. Instead of using $\mathcal{S}_1$, $\mathcal{S}$ simulates the case of cheating on its own. To this end, $\mathcal{S}$ asks the ideal functionality whether or not cheating is detected. If cheating is detected, the remaining simulation is mostly straightforward. One subtlety we like to highlight here is that $\mathcal{S}$ needs to provide the output values of the corrupted parties of $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ to Adv. $\mathcal{S}$ obtains these values by computing the offline function f_{off}. Since this function is independent of the inputs of honest parties, $\mathcal{S}$ is indeed able to compute values that are indistinguishable to the values in the hybrid world execution.

If cheating is undetected, $\mathcal{S}$ needs to simulate the remaining steps of π_{on}. Note that if cheating is undetected, $\mathcal{S}$ obtains the inputs of the honest parties from the ideal functionality. Moreover, the adversary provides to $\mathcal{S}$ the potentially corrupted output values of the hybrid functionality for the honest parties. Now, $\mathcal{S}$ knows all information to act exactly like honest parties do in the hybrid world execution and therefore the resulting view is indistinguishable as well.

We finally give the idea about the deterrence factor of π_{on} in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world. We know that cheating during all steps after the call to the hybrid functionality is detected with probability ϵ_{on}. This is due to the fact that π_{on} is covertly secure with deterrence factor ϵ_{on} in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world. Now, any cheat attempt in the hybrid functionality is detected only with probability ϵ'_{off}. Since the adversary can decide when he wants to cheat, the detection probability of π_{on} in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world is $\epsilon'_{\mathsf{on}} = \min(\epsilon_{\mathsf{on}}, \epsilon'_{\mathsf{off}})$.

4 Covert Online Protocol

In this section, we demonstrate the applicability of our new paradigm to achieve covert security. To this end, we construct a covertly secure online phase for the TinyOT protocol [NNOB12]. We refer to Section 1.3 for the intuition and high-level idea of TinyOT. Here, we present the exact specification of our covertly secure online protocol. We present our protocol in a hybrid world where the offline phase is modeled via a hybrid functionality and show its covert security under the intermediate explicit cheat formulation (IECF) (cf. Definition 1) in the random oracle model.

In the following, we first present the notation we use to describe our protocol. Then, we state the building blocks of our protocol, especially, an ideal commitment functionality and the offline functionality, which are both used as hybrid functionalities. Next, we present the exact specification of our two-party online protocol and afterwards prove its security.

We remark that we focus on the two-party setting, since this setting is sufficient to show applicability and the benefit of our paradigm. Nevertheless, we believe our protocol can easily be extended to the multi-party case following the multi-party extensions of TinyOT ([LOS14, BLN$^+$21, FKOS15, WRK17b]).

Notation. We use the following notation to describe secret shared and authenticated values. This notation follows the common approach in the research field [NNOB12, DPSZ12, WRK17a, WRK17b]. For covert security parameter t, both parties have a global key, $\Delta_{\mathcal{A}}$ resp. $\Delta_{\mathcal{B}}$, which are bit strings of length t. A bit x is authenticated to a party $\mathcal{A}$ by having the other party $\mathcal{B}$ hold a random t-bit key, $K[x]$, and having $\mathcal{A}$ hold the bit x and a t-bit MAC $M[x] = K[x] \oplus x \cdot \Delta_{\mathcal{B}}$. We denote an authenticated bit x known to $\mathcal{A}$ as $\langle x \rangle^{\mathcal{A}}$ which corresponds to the tuple $(x, K[x], M[x])$ in which x and $M[x]$ is known by $\mathcal{A}$ and $K[x]$ by $\mathcal{B}$. A public constant c can be authenticated to $\mathcal{A}$ non-interactively by defining $\langle c \rangle^{\mathcal{A}} := (c, c \cdot \Delta_b, 0^\kappa)$. Authenticated bits known to $\mathcal{B}$ are authenticated and denoted symmetrically.

A bit z is secret shared by having $\mathcal{A}$ hold a value x and $\mathcal{B}$ hold a value y such that $z = x \oplus y$. The secret shared bit is authenticated by authenticating the individual shares of $\mathcal{A}$ and $\mathcal{B}$, i.e., by using $\langle x \rangle^{\mathcal{A}}$ and $\langle y \rangle^{\mathcal{B}}$. We denote the authenticated secret sharing $(\langle x \rangle^{\mathcal{A}}, \langle y \rangle^{\mathcal{B}}) = (x, K[x], M[x], y, K[y], M[y])$ by $\langle z \rangle$ or $\langle x|y \rangle$.

Observe that this kind of authenticated secret sharing allows linear operations, i.e., addition of two secret shared values as well as addition and multiplication of a secret shared value with a public constant. In order to calculate $\langle \gamma \rangle := \langle \alpha \rangle \oplus \langle \beta \rangle$ with $\langle \alpha \rangle = \langle a_{\mathcal{A}} | a_{\mathcal{B}} \rangle$, $\langle \beta \rangle = \langle b_{\mathcal{A}} | b_{\mathcal{B}} \rangle$, parties compute the authenticated share of γ of $\mathcal{A}$ as $\langle c_{\mathcal{A}} \rangle^{\mathcal{A}} := (a_{\mathcal{A}} \oplus b_{\mathcal{A}}, K[a_{\mathcal{A}}] \oplus K[b_{\mathcal{A}}], M[a_{\mathcal{A}}] \oplus M[b_{\mathcal{A}}])$. The authenticated share of γ of $\mathcal{B}$, $\langle c_{\mathcal{B}} \rangle^{\mathcal{B}}$, is calculated symmetrically. It follows that $\langle \gamma \rangle = \langle c_{\mathcal{A}} | c_{\mathcal{B}} \rangle$ is an authenticated sharing of $\alpha \oplus \beta$. In order to calculate $\langle \gamma \rangle := \langle \alpha \rangle \oplus \beta$ for a public constant β and α defined as above, parties first create authenticated constants bits $\langle \beta \rangle^{\mathcal{A}}$ and $\langle 0 \rangle^{\mathcal{B}}$ and define $\langle \beta \rangle := \langle \beta | 0 \rangle$. In order to calcualte $\langle \gamma \rangle := \langle \alpha \rangle \cdot \beta$ for a public constant β and α defined as above, parties set $\langle \gamma \rangle := \langle \alpha \rangle$ if $b = 1$ and $\langle \gamma \rangle := \langle 0|0 \rangle$ if $b = 0$.

Finally, we use the notation $[n]$ to denote the set $\{1, \ldots, n\}$. We consider any sets to be ordered, e.g., $\{x_i\}_{i \in [n]} := [x_1, x_2, \ldots, x_n]$, and for a set of indices $\mathcal{I} = \{x_i\}_{i \in [n]}$ we denote the i-th element of $\mathcal{I}$ as $\mathcal{I}[i]$. Note, that $M[x]$ always denotes a MAC for bit x and we only denote the i-th element for sets of indices which we denote by $\mathcal{I}$.

Ideal commitments. The protocol uses an hybrid commitment functionality $\mathcal{F}_{\mathsf{Commit}}$ that is specified as follows:

Functionality $\mathcal{F}_{\mathsf{Commit}}$: Commitments

The functionality interacts with two parties, $\mathcal{A}$ and $\mathcal{B}$.

– Upon receiving (Commit, x_P) from party $P \in \{\mathcal{A}, \mathcal{B}\}$, check if Commit was not received before from P. If the check holds, store x_P and send ($\mathsf{Committed}, P$) to party $\bar{P} \in \{\mathcal{A}, \mathcal{B}\} \setminus P$.
– Upon receiving (Open) from party $P \in \{\mathcal{A}, \mathcal{B}\}$, check if Commit was received before from P. If the check holds, send (Open, P, x_P) to party $\bar{P} \in \{\mathcal{A}, \mathcal{B}\} \setminus P$.

Offline functionality. The online protocol uses an hybrid offline functionality $\mathcal{F}^{\epsilon}_{f_{\mathsf{off}}}$ to provide authenticated bits and authenticated triples. Function f_{off} is defined as follows.

Functionality f_{off}: Precomputation

The function receives inputs by two parties, $\mathcal{A}$ and $\mathcal{B}$. W.l.o.g., we assume that if any party is corrupted it is $\mathcal{A}$. The function is parametrized with a number of authenticated bits, n_1, a number of authenticated triples n_2 and the deterrence parameter t.
Inputs: $\mathcal{A}$ provides either input ok or $(\Delta_{\mathcal{A}}, \{r_i, K[s_i], M[r_i]\}_{i \in [n_1 + 3 \cdot n_2]})$ where $\Delta_{\mathcal{A}}, K[\cdot], M[\cdot]$ are t-bit strings and r_i is a bit for $i \in [n_1 + 3 \cdot n_2]$. An honest $\mathcal{A}$ will always provide input ok. $\mathcal{B}$ provides input ok.
Computation: The function calculates authenticated bits and authenticated shared triples as follows:

– Sample $\Delta_{\mathcal{B}} \in_R \{0,1\}^t$. Do the same for $\Delta_{\mathcal{A}}$ if not provided as input.
– For each $i \in [n_1 + 3 \cdot n_2]$, sample $s_i \in_R \{0,1\}$. If not provided as input, sample $r_i \in_R \{0,1\}$ and $K[s_i], M[r_i] \in_R \{0,1\}^t$. Set $K[r_i] := M[r_i] \oplus r_i \cdot \Delta_B$ and $M[s_i] := K[s_i] \oplus s_i \cdot \Delta_A$. Define $\langle r_i \rangle^{\mathcal{A}} := (r_i, K[r_i], M[r_i])$ and $\langle s_i \rangle^{\mathcal{B}} = (s_i, K[s_i], M[s_i])$.
– For each $i \in [n_2]$, set $j = n_1 + 3 \cdot i$ and define $x := r_j \oplus (r_{j-1} \oplus s_{j-1}) \cdot (r_{j-2} \oplus s_{j-2})$, $K[x] := K[s_j]$, and $M[x] := K[x] \oplus x \cdot \Delta_A$ and $\langle x \rangle^{\mathcal{B}} := (x, K[x], M[x])$. Then, define the multiplication triple $\langle \alpha_i \rangle := \langle r_{j-2} | s_{j-2} \rangle$, $\langle \beta_i \rangle := \langle r_{j-1} | s_{j-1} \rangle$, and $\langle \gamma_i \rangle := \langle r_j | x \rangle$.

Output: Output global keys $(\Delta_{\mathcal{A}}, \Delta_{\mathcal{B}})$, authenticated bits $\{(\langle r_i \rangle^{\mathcal{A}}, \langle s_i \rangle^{\mathcal{B}})\}_{i \in [n_1]}$, and authenticated shared triples $\{(\langle \alpha_i \rangle, \langle \beta_i \rangle, \langle \gamma_i \rangle)\}_{i \in [n_2]}$, and assign $\mathcal{A}$ and $\mathcal{B}$ their respective shares, keys and macs.

We present a protocol instantiating $\mathcal{F}^{\epsilon}_{f_{\mathsf{off}}}$ in Appendix D.
Online protocol. The online protocol works in four steps. First, the parties obtain authenticated bits and triples from the hybrid offline functionality. Second, the parties secret share their inputs and use authenticated bits to obtain authenticated shares of the inputs wires of the circuit. Third, the parties evaluate the boolean circuit on the authenticated values. While XOR-gates are computed locally, AND-gates require communication between the parties and the consumption of a precomputed authenticated triple for each gate. Finally, in the output phase each party verifies the MACs on the computed values to check for correct behavior of the other party. If no cheating was detected, the parties exchange their shares on the output wires to recompute the actual outputs.

We modified the original TinyOT online phase in two aspects. First, the original TinyOT protocol uses one-sided authenticated precomputation data, e.g., one-sided authenticated triples where the triple is not secret shared but known to one party. In contrast, we focus on a simplification [WRK17a] where the authenticated triples are secret shared among all parties. This allows us to use a single two-sided authenticated triple for each AND gate instead of two one-sided authenticated triples with additional data. Second, we integrate commitments in the output phase. In detail, the parties first commit on their shares for the output wires together with the corresponding MACs and only afterwards reveal the committed values. By using commitments, the adversary needs to decide first if it wants to cheat and only afterwards it learns the output. However, since the adversary can commit on incorrect values, it still can learn its output even if the honest parties detect its cheating afterwards. We show the security of this protocol under the IECF of covert security.

To prevent the adversary from inserting incorrect values into the commitment, the generation of the commitments can be part of the circuit evaluation. By checking the correct behavior of the entire evaluation, honest parties detect cheating with the inputs to the commitments with a fixed probability. This way, we can achieve the strong explicit cheat formulation (SECF). Since computing the commitments as part of the circuit reduces the efficiency, we opted for the less expensive protocol.

Protocol Π_{on}: TinyOT-style online protocol

The protocol is executed between parties $\mathcal{A}$ and $\mathcal{B}$ and uses of a hash function H (modeled as non-programmable random oracle), the hybrid commitment functionality $\mathcal{F}_{\mathsf{Commit}}$, and the hybrid covert functionality $\mathcal{F}^{1}_{f_{\mathsf{off}}}$, in the following denoted as $\mathcal{F}_{\mathsf{off}}$. f_{off} is instantiated with the same public parameters as the protocol. When denoting a particular party with P, we denote the respective other party with $\bar{P}$.

Public parameters: The deterrence parameter t and the number of input bits and output bits per party n_1. A function $f(\{x_{(i,\mathcal{A})}\}_{i\in[n_1]}, \{x_{(i,\mathcal{B})}\}_{i\in[n_1]}) = (\{z_{(i,\mathcal{A})}\}_{i\in[n_1]}, \{z_{(i,\mathcal{B})}\}_{i\in[n_1]})$ with $x_{(*,\mathcal{A})}, x_{(*,\mathcal{B})}, z_{(*,\mathcal{A})}, z_{(*,\mathcal{B})} \in \{0,1\}$ and a boolean circuit $\mathcal{C}$ computing f with n_2 AND gates. $\{z_{(i,\mathcal{A})}\}_{i\in[n_1]}$ resp. $\{z_{(i,\mathcal{B})}\}_{i\in[n_1]}$ is the output of $\mathcal{A}$ resp. $\mathcal{B}$. The set of indices of input wires resp. output wires of each party $P \in \{\mathcal{A},\mathcal{B}\}$ is denoted by $\mathcal{I}^{\mathsf{in}}_P$ resp. $\mathcal{I}^{\mathsf{out}}_P$. Without loss of generality, we assume that the wire values are ordered in topological order.

Inputs: $\mathcal{A}$ has input bits $\{x_{(i,\mathcal{A})}\}_{i\in[n_1]}$ and $\mathcal{B}$ has input bits $\{x_{(i,\mathcal{B})}\}_{i\in[n_1]}$.

Pre-computation phase:

1. Each party $P \in \{\mathcal{A},\mathcal{B}\}$ defines ordered sets $\mathcal{M}^P_P := \emptyset$, $\mathcal{M}^P_{\bar{P}} := \emptyset$, sends (ok) to $\mathcal{F}_{\mathsf{off}}$ and receives its shares of $(\{(\langle r_{(i,\mathcal{A})}\rangle^{\mathcal{A}}, \langle r_{(i,\mathcal{B})}\rangle^{\mathcal{B}})\}_{i\in[n_1]}, \{(\langle\alpha_j\rangle, \langle\beta_j\rangle, \langle\gamma_j\rangle)\}_{j\in[n_2]})$. If $\mathcal{F}_{\mathsf{off}}$, returns $m \in \{\mathsf{abort}, \mathsf{corrupted}_{\bar{P}}\}$, P outputs m and aborts.

Input phase:

2. For each $i \in [n_1]$, each party $P \in \{\mathcal{A},\mathcal{B}\}$ sends $d_{(i,P)} := x_{(i,P)} \oplus r_{(i,P)}$. Then, the parties define $\langle x_{(i,\mathcal{A})}\rangle := \langle r_{(i,\mathcal{A})}|0\rangle \oplus d_{(i,\mathcal{A})}$ and $\langle x_{(i,\mathcal{B})}\rangle := \langle 0|r_{(i,\mathcal{B})}\rangle \oplus d_{(i,\mathcal{B})}$ For each party $P \in \{\mathcal{A},\mathcal{B}\}$ and each $j \in [n_1]$ with $i := \mathcal{I}^{\mathsf{in}}_P[j]$, the parties assign $\langle x_{(j,P)}\rangle$ to $\langle w_i\rangle$.

Circuit evaluation phase:

3. Repeat till all wire values are assigned. Let j be the smallest index of an unassigned wire. Let l and r be the indices of the left resp. right input wire of the gate computing w_j. Dependent on the gate type, $\langle w_j \rangle$ is calculated as follows:

- **XOR-Gate:** $\langle w_j \rangle := \langle w_l \rangle \oplus \langle w_r \rangle$
- **AND-Gate:** For the i-th AND gate, the parties define $(\langle \alpha \rangle, \langle \beta \rangle, \langle \gamma \rangle) := (\langle \alpha_i \rangle, \langle \beta_i \rangle, \langle \gamma_i \rangle)$, calculate $\langle e \rangle = \langle e^{\mathcal{A}} | e^{\mathcal{B}} \rangle := \langle \alpha \rangle \oplus \langle w_l \rangle$ and $\langle d \rangle = \langle d^{\mathcal{A}} | d^{\mathcal{B}} \rangle := \langle \beta \rangle \oplus \langle w_r \rangle$, open e and d by publishing $e^{\mathcal{A}}, e^{\mathcal{B}}, d^{\mathcal{A}}, d^{\mathcal{B}}$ respectively, and compute $\langle w_j \rangle := \langle \gamma \rangle \oplus e \cdot \langle w_r \rangle \oplus d \cdot \langle w_l \rangle \oplus e \cdot d$.
  Further, each party $P \in \{\mathcal{A}, \mathcal{B}\}$ appends $(M[e^P], M[d^P])$ to $\mathcal{M}_P^P$ and $((K[e^{\bar{P}}] \oplus e^{\bar{P}} \cdot \Delta_P), (K[d^{\bar{P}}] \oplus d^{\bar{P}} \cdot \Delta_P))$ to $\mathcal{M}_{\bar{P}}^P$.

Output phase:

4. Party $P \in \{\mathcal{A}, \mathcal{B}\}$ computes $\mathcal{M}^1_{(P,P)} := H(\mathcal{M}_P^P)$ and $\mathcal{M}^1_{(P,\bar{P})} = H(\mathcal{M}_{\bar{P}}^P)$ and sends $\mathcal{M}^1_{(P,P)}$.
5. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$, upon receiving $\mathcal{M}^1_{(\bar{P},\bar{P})}$, verifies that $\mathcal{M}^1_{(\bar{P},\bar{P})} = \mathcal{M}^1_{(P,\bar{P})}$. If not, P outputs $\mathsf{corrupted}_{\bar{P}}$ and aborts. Otherwise, P computes $\mathcal{M}^2_{(P,P)} := H(\{M[w_i^P]\}_{i \in \mathcal{I}_{\bar{P}}^{\mathsf{out}}})$, and sends $(\mathsf{Commit}, (\{w_i^P\}_{i \in \mathcal{I}_{\bar{P}}^{\mathsf{out}}}, \mathcal{M}^2_{(P,P)}))$ to $\mathcal{F}_{\mathsf{Commit}}$.
6. Upon receiving, $(\mathsf{Committed}, \bar{P})$ from $\mathcal{F}_{\mathsf{Commit}}$, P sends (Open) to $\mathcal{F}_{\mathsf{Commit}}$.
7. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$, upon receiving $(\mathsf{Opened}, \bar{P}, (\{w_i^{\bar{P}}\}_{i \in \mathcal{I}_P^{\mathsf{out}}}, \mathcal{M}^2_{(\bar{P},\bar{P})}))$ from $\mathcal{F}_{\mathsf{Commit}}$, re-defines $\mathcal{M}_{\bar{P}}^P := \{K[w_i^{\bar{P}}] \oplus w_i^{\bar{P}} \cdot \Delta_P\}_{i \in \mathcal{I}_P^{\mathsf{out}}}$ and verifies that $\mathcal{M}^2_{(\bar{P},\bar{P})} = H(\mathcal{M}_{\bar{P}}^P)$. If not, P outputs $\mathsf{corrupted}_{\bar{P}}$ and aborts. Otherwise, P outputs $\{w_i^P \oplus w_i^{\bar{P}}\}_{i \in \mathcal{I}_P^{\mathsf{out}}}$.

Handle aborts:

8. If a party P does not receive a timely message before executing Step 6, it outs abort and aborts. If a party P does not receive a timely message after having executed Step 6, it outputs $\mathsf{corrupted}_{\bar{P}}$ and aborts.

Security. Intuitively, successful cheating in the context of the online protocol is equivalent to correctly guessing the global key of the other party. Let us assume $\mathcal{A}$ is corrupted. It is evident that $\mathcal{A}$ can only behave maliciously by flipping the bits sent during the evaluation phase and the output phase – flipping a bit during the input phase is not considered cheating as the adversary, $\mathcal{A}$, is allowed to pick its input arbitrarily. For each of those bits, there is a MAC check incorporated into the protocol. Hence, $\mathcal{A}$ needs to guess the correct MACs for the flipped bits ($\mathcal{A}$ knows the ones of the unflipped bits) in order to cheat successfully. As a MAC $M[b^{\mathcal{A}}]$ for a bit $b^{\mathcal{A}}$ known to $\mathcal{A}$ is defined as $K[b^{\mathcal{A}}] \oplus b^{\mathcal{A}} \cdot \Delta_{\mathcal{B}}$, a MAC $\widetilde{M}[\tilde{b}^{\mathcal{A}}]$ of a flipped bit $\tilde{b}^{\mathcal{A}}$ is correct iff $\widetilde{M}[\tilde{b}^{\mathcal{A}}] = M[b^{\mathcal{A}}] \oplus \Delta_{\mathcal{B}} = K[b^{\mathcal{A}}] \oplus (b^{\mathcal{A}} \oplus 1) \cdot \Delta_{\mathcal{B}}$. It follows that $\mathcal{A}$ has to guess the global key of $\mathcal{B}$ and apply it to the MACs of all flipped bits in order to cheat successfully. As the global key has t bits, the chance of guessing the correct global key is $\frac{1}{2^t}$. It follows that the deterrence factor ϵ equals $1 - \frac{1}{2^t}$. More formally, we state the following theorem and prove its correctness in Appendix E:

Theorem 2. *Let H be a (non-programmable) random oracle, $t \in \mathbb{N}$, and $\epsilon = 1 - \frac{1}{2^t}$. Then, protocol Π_{on} securely implements $\mathcal{F}_f^\epsilon$ (i.e., constitutes a covertly secure protocol with deterrence factor ϵ) in the presence of a rushing adversary according to the intermediate explicit cheat formulation as defined in Definition 1 in the $(\mathcal{F}_{\mathsf{off}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid world.*

On the usage of random oracles. As explained above, successful cheating is equivalent to guessing the global key of the other party. However, a malicious party can also cheat inconsistently, i.e., it guesses different global keys for the flipped bits, or even provide incorrect MACs for unflipped bits. In this case, the adversary has no chance of cheating successfully, which needs to be detected by the simulator. As the simulator only receives a hash of a all MACs, it needs some trapdoor to learn the hashed MACs and check for consistency. To provide such a trapdoor, we model the hash function as a random oracle. The requirement of a random oracle can be removed if the parties send all MACs in clear instead of hashing them first. However, this increases the communication complexity.

Another alternative is to bound the deterrence parameter t such that the simulator can try out all consistent ways to compute the MACs of flipped bits, i.e., each possible value for the guessed global key, hash those and compare them to the received hash. In this case, it is sufficient to require collision resistance of the hash function. As the number of possible values for the global key grows exponentially with the deterrence parameter t, i.e., 2^t, this approach is only viable if we bound t. Nevertheless, the probability of successful cheating also declines exponentially with t, i.e., $\frac{1}{2^t}$. Hence, for small values of t, the simulator runs in reasonable time.

5 Evaluation

In Section 4, we showed the application of our new paradigm to achieve covert security on the example of the TinyOT online phase. By shortening the MAC length in the online phase, we also reduced the amount of precomputation required from the offline phase. In order to quantify the efficiency gain that can be achieved by generating shorter MACs, we compare the communication complexity of a covert offline phase generating authenticated bits and triples with short MACs to the covert offline phase generating bits and triples with long MACs.

The offline protocol. To the best of our knowledge, there is no explicit covert protocol for the precomputation of TinyOT-style protocols. Therefore, we rely on generic transformations from semi-honest to covert security based on the cut-and-choose paradigm, similar to the transformations proposed by [DOS20, FHKS21, SSS22]. However, semi-honest precomputation protocols do not consider authentication of bits and triples, since semi-honest online protocols do not need authentication. Hence, it is necessary to first extend the semi-honest protocol to generate MACs, and then, apply the generic transformation. We first specify a semi-honest protocol to generate authenticated bits and triples as well as the covert protocol that can be derived via the cut-and-choose approach.

Both protocols are presented in Appendix D. Then, we take the resulting covert protocol to evaluate the communication complexity for different MAC lengths.

ϵ	# triples	λ-bit MACs (state-of-the-art)	Short MACs (our approach)	Improvement
$\frac{1}{2}$	10 K	531	333	37,19%
	100 K	5 211	3 258	37,47%
	1 M	52 011	32 508	37,50%
	1 B	52 000 011	32 500 008	37,50%
$\frac{3}{4}$	10 K	1 062	677	36,24%
	100 K	10 422	6 617	36,51%
	1 M	104 022	66 017	36,54%
	1 B	104 000 022	66 000 017	36,54%
$\frac{7}{8}$	10 K	2 124	1 374	35,29%
	100 K	20 844	13 434	35,55%
	1 M	208 044	134 034	35,57%
	1 B	208 000 044	134 000 034	35,58%

Table 1: Concrete communication complexity of the covert offline phase generating the precomputation required for a maliciously secure TinyOT online phase (as applied by state-of-the-art) and a covertly secure TinyOT online phase (our approach). As the offline phase is covertly secure, the overall protocol's security level is the same in both approaches. Communication is reported in kB per party.

Evaluation results. The communication complexity of each party is determined as follows. Let κ be the computational security parameter, λ be the statistical security parameter, t be the cut-and-choose parameter (which results in a deterrence factor $\epsilon = 1 - \frac{1}{t}$), M be the length of the generated MACs, n_1 be the number of authenticated bits required per party, n_2 be the number of authenticated triples, C_{OT} be the communication complexity of one party for performing κ base oblivious transfers with κ-bit strings twice, once as receiver and once as sender, C_{Commit} be the size of a commitment and C_{Open} be the size of an opening to a κ-bit seed. Then, each party needs to send C bits with C equal to

$$(t+1) \cdot C_{\mathsf{Commit}} + t \cdot (C_{\mathsf{OT}} + C_{\mathsf{Open}} + n_2 \cdot (3 + \kappa - 1) + (n_1 + 2 \cdot n_2) \cdot (M - 1))$$

In our approach, M is defined such that $t = 2^M$. In the classical approach with a maliciously secure online phase M is fixed to equal λ. This yields an absolute efficiency gain of G bits with G equal to

$$t \cdot (n_1 + 2 \cdot n_2) \cdot (\lambda - M)$$

In the following, we set $\kappa = 128$, $\lambda = 40$, $C_{\mathsf{OT}} = (2 + \kappa) \cdot 256$ according to [MRR21], $C_{\mathsf{Commit}} = 256$ and $C_{\mathsf{Open}} = 2 \cdot \kappa$ according to a hash-based commitment scheme. Further, we fix $n_1 = 256$. This yields the communication complexity depicted in Table 1. For deterrence factors up to $\frac{7}{8}$, our approach reduces the communication per party by at least 35%. As a reduction of the security of the online phase to the level of the offline phase does not affect the overall protocol's security, as shown in Section 3.2, this efficiency improvement is for free.

Acknowledgments

The first, third, and fourth authors were supported by the German Federal Ministry of Education and Research (BMBF) *iBlockchain project* (grant nr. 16KIS0902), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) *SFB 1119 – 236615297 (CROSSING Project S7)*, and by the BMBF and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*. The second author was supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by ISF grant No. 1316/18.

References

[ABL$^+$18] David W. Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. From keys to databases - real-world applications of secure multi-party computation. *Comput. J.*, 2018.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*, 2007.

[AO12] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *ASIACRYPT*, 2012.

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.

[BCG$^+$20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *FOCS*, 2020.

[BCG$^+$20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO*, 2020.

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using topgear in overdrive: A more efficient zkpok for SPDZ. In *SAC*, 2019.

[BLN$^+$21] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High-performance multi-party computation for binary circuits based on oblivious transfer. *J. Cryptol.*, 34(3):34, 2021.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1), 2000.

[CDE$^+$18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spd$\mathbb{F}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, 2018.

[CKR$^+$20] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In *ASIACRYPT*, 2020.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In *CRYPTO*, 2022.

[DKL$^+$13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, 2013.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO*, 2017.

[DOS20] Ivan Damgård, Claudio Orlandi, and Mark Simkin. Black-box transformations from passive to covert security with public verifiability. In *CRYPTO*, 2020.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, 2013.

[FHKS21] Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. Generic compiler for publicly verifiable covert multi-party computation. In *EUROCRYPT*, 2021.

[FKOS15] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In *ASIACRYPT*, 2015.

[HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, tinykeys for tinyot). In *ASIACRYPT*, 2018.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO*, 2014.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *CCS*, 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, 2018.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *CRYPTO*, 2018.

[KVH$^+$21] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *NeurIPS*, 2021.

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In *CRYPTO*, 2014.

[MPC] MPC Alliance. https://www.mpcalliance.org/. (Accessed on 10/14/2022).

[MRR21] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Batching base oblivious transfers. In *ASIACRYPT*, 2021.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, 2012.

[Ors20] Emmanuela Orsini. Efficient, actively secure MPC with a dishonest majority: A survey. In *WAIFI*, 2020.

[SSS22] Peter Scholl, Mark Simkin, and Luisa Siniscalchi. Multiparty computation with covert security and public verifiability. In *ITC*, 2022.

[VSG$^+$19] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. Conclave: secure multi-party computation on big data. In *EuroSys*, 2019.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, 2017.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *CCS*, 2017.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *CCS*, 2020.

[Zen] ZenGo - crypto wallet app. https://zengo.com/. (Accessed on 10/14/2022).

Supplementary Materials:

Putting the Online Phase on a Diet: Covert Security from Short MACs

A Discussion of Constraints on Online Protocol

In this section, we discuss the constraints on the online protocol used in our theorem. These constraints emerged from technical issues and it is unclear how to prove our deterrence replacement theorem in a more generic setting. Recall that in our proof $\mathcal{S}$ uses the simulator $\mathcal{S}_1$ which exists since π_{on} is covertly secure in the $\mathcal{F}^1_{\mathsf{off}}$-hybrid world.

First, the hybrid functionality $\mathcal{F}_{\mathsf{off}}$ needs to be called directly at the beginning. This enables the simulator $\mathcal{S}$ to react to the adversary's cheating decision in the offline phase, i.e., its input to $\mathcal{F}_{\mathsf{off}}$, right at the start of the simulation. More specifically, $\mathcal{S}$ uses the black-box simulator $\mathcal{S}_1$ in case the adversary does not cheat and simulates on its own in case there is a cheating attempt. If there would be protocol interactions before the call to $\mathcal{F}_{\mathsf{off}}$, $\mathcal{S}$ would have to decide whether it simulates this interactions itself or via $\mathcal{S}_1$. This means that the adversary's input to $\mathcal{F}_{\mathsf{off}}$ could require $\mathcal{S}$ to change its decision, e.g., require $\mathcal{S}$ to simulate the following steps itself while $\mathcal{S}$ initially used $\mathcal{S}_1$ for the earlier steps. This leads to a problem as $\mathcal{S}$ uses $\mathcal{S}_1$ in a black-box way, and hence, can only use it for all or none of the protocol steps. Rewinding does not solve the problem as a change in the simulation of the steps before the call to $\mathcal{F}_{\mathsf{off}}$ can influence the adversary's input to $\mathcal{F}_{\mathsf{off}}$, and hence, $\mathcal{S}$'s decision to simulate the steps afterwards based on $\mathcal{S}_1$ or not.

Second, we require that in case $\mathcal{F}_{\mathsf{off}}$ outputs $\mathsf{corrupted}$, the protocol π_{on} instructs the parties to output $\mathsf{corrupted}$ as well. This is due to some subtle detail in the security proof. As $\mathcal{S}_1$ runs in a world, in which cheating in the offline phase is not possible, $\mathcal{S}_1$ does not know how to deal with undetected cheating. Further, we treat the protocol π_{on} in a black-box way. Due to these facts, the only way for $\mathcal{S}$ to simulate the case of undetected cheating is to follow the actual protocol. To do so in a consistent way, $\mathcal{S}$ has to get the input of the honest parties. Hence, $\mathcal{S}$ has to notify the ideal covert functionality $\mathcal{F}^{\epsilon_{\mathsf{on}}}_{\mathsf{on}}$ about the cheating attempt in the offline phase. In case of detected cheating, $\mathcal{F}^{\epsilon'_{\mathsf{on}}}_{\mathsf{on}}$ sends $\mathsf{corrupted}$ to the honest parties and thus the honest parties output $\mathsf{corrupted}$ in the ideal world. In order to achieve indistinguishability between the ideal world and the real world, π_{on} needs to instruct the honest parties to output $\mathsf{corrupted}$ in the real world, too.

Finally, we emphasize that known offline/online protocols (SPDZ [DPSZ12], TinyOT [NNOB12], authenticated garbling [WRK17a, WRK17b]) either directly fulfill the aforementioned requirements or can easily be adapted to do so.

B Proof of Theorem 1

Here, we provide the full and formal proof of our deterrence replacement theorem which is stated in Theorem 1.

Proof. By assumption, π_{on} is covertly secure in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world and thus there exists a corresponding simulator $\mathcal{S}_1$ in the ideal world with the covert online functionality $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$. We prove Theorem 1 by constructing a simulator $\mathcal{S}$ for π_{on} in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world that utilizes simulator $\mathcal{S}_1$. This means, $\mathcal{S}$ is running in the ideal world with the covert online functionality $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and internally runs the adversary Adv and $\mathcal{S}_1$. Here, Adv is the adversary attacking the protocol π_{on} executed in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world.

In the ideal world with $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$, $\mathcal{S}$ plays the role of both the honest parties and the hybrid functionality $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ towards Adv. In addition, $\mathcal{S}$ needs to provide the inputs of the corrupted parties to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$. Let $\mathcal{I}$ be the set of indices denoting the set of corrupted parties. The full simulator description is given in Section 3.2.

Next, we show indistinguishability between the ideal world execution and the hybrid world execution. In the hybrid world, the adversary Adv interacts with the honest parties, where the honest parties act as specified by π_{on}, and all parties have access to the hybrid functionality $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$. The ideal world involves ideal functionality $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$, the honest parties, which just forwards their inputs to the ideal functionality, and simulator $\mathcal{S}$ which internally executes Adv. For both worlds, we consider the joint distribution of the honest parties' outputs and the view of Adv. We show indistinguishability between these two distributions in three steps. First, we prove that the adversary tries to cheat in the offline hybrid functionality with the same probability in the hybrid and in the ideal world. Second, we show that the joint output distributions are indistinguishable if the adversary refrains from cheating in the offline hybrid functionality. Third, we argue about the indistinguishability if the adversary tries to cheat in the offline hybrid functionality. At the end, since the cases happen with the same probability in both worlds and the joint output distributions are indistinguishable in each case, it follows that the entire ideal world execution is indistinguishable from the hybrid world execution.

Claim. The probability that Adv sends $(\mathsf{cheat}_i, \cdot)$ for $i \in \mathcal{I}$ to the offline hybrid functionality is identical in the hybrid and ideal world.

Recall that the call to the hybrid functionality $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ is the first message sent by the adversary Adv as stated as the first requirement on the online protocol above. Thus, the input to $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ only depends on the random tape and the code of Adv. This is true for the hybrid world as well as for the ideal world.

In the ideal world, simulator $\mathcal{S}$ ask $\mathcal{S}_1$ for the random tape of the adversary. We make the standard assumption that $\mathcal{S}_1$ fixes a uniformly sampled random tape for the adversary at the beginning and uses this randomness throughout the simulation. As Adv receives a uniformly sampled random tape in both worlds and its decision depends only on the random tape and its own code, it follows that the cheating probability $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world is identical to the cheating probability in the ideal world with ideal online functionality $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$.

Claim. The ideal world execution is indistinguishable from the hybrid world execution if the adversary *does not* send $(\mathsf{cheat}_i, \cdot)$ for $i \in \mathcal{I}$ to $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$.

In case (a) of the simulator, i.e., if Adv sends $\perp$ or abort as additional input to $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$, $\mathcal{S}$ uses simulator $\mathcal{S}_1$ for the remaining simulation. $\mathcal{S}_1$ exists since π_{on} is covertly secure in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world. To use $\mathcal{S}_1$, every message from Adv is forwarded to $\mathcal{S}_1$ and every response is sent back to Adv. The same procedure is done with messages from $\mathcal{S}_1$ to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and vice versa.

We show Claim 2 via a reduction to the assumption that π_{on} is covertly secure in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world. To this end, we start with the opposite, i.e., we assume there exists an adversary Adv that does not send $(\mathsf{cheat}_i, \cdot)$ with its first message with non-negligible probability and the simulation in the $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ ideal world conditioned on the event that Adv does not send $(\mathsf{cheat}_i, \cdot)$ is distinguishable from the protocol execution in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world. Then, we construct an adversary Adv' that uses Adv in a black-box way to make the simulation involving $\mathcal{S}_1$ in the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world distinguishable from the protocol execution in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid world. Since this leads to a contradiction, our initial assumption that the hybrid world execution is distinguishable from the ideal world execution is false and thus Claim 2 holds.

Adv' is constructed as follows. Upon receiving the random tape from $\mathcal{S}_1$ it executes Adv on the same random tape. If Adv does send $(\mathsf{cheat}_i, \cdot)$, Adv' forwards abort_i and aborts. Once, Adv does not send $(\mathsf{cheat}_i, \cdot)$, Adv' forwards the message to $\mathcal{S}_1$ (in the ideal world) resp. $\mathcal{F}_{\mathsf{off}}^{1}$ (in the hybrid world) and continues by forwarding all messages between Adv and $\mathcal{S}_1$ (in the ideal world) resp. the honest parties (in the hybrid world). In the end Adv' outputs whatever Adv outputs.

We observe two important aspects. First, conditioned on the event that Adv does not send $(\mathsf{cheat}_i, \cdot)$ with its first message, each simulation produced by $\mathcal{S}_1$ when interacting with Adv' in the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world is equivalent to the simulation $\mathcal{S}$ produces when interacting with Adv in the $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ ideal world. Second, given the same condition, each hybrid world protocol execution produced by Adv' in the $\mathcal{F}_{\mathsf{off}}^{\epsilon_{\mathsf{off}}}$-hybrid world is equivalent to the hybrid world protocol execution Adv produces in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world. Recall that by our initial assumption, conditioned on the event that Adv does not send $(\mathsf{cheat}_i, \cdot)$, Adv produces views that can be used to distinguish between the $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$-ideal world with $\mathcal{S}$ and the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world. As Adv', conditioned on the same event, produces views that are equivalent to the ones produces by Adv, these views can be used to distinguish between $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world with simulator $\mathcal{S}_1$ and $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid. This contradicts our assumption that π_{on} is covertly secure in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world.

What remains is to show that our two observations hold. Observation one follows from the following facts: In any two concrete experiments with equivalent inputs in the $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ ideal world and the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world, in which our condition holds, it is true that (a) $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ is executed identical to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$, (b) $\mathcal{S}$'s execution is identical to the one of $\mathcal{S}_1$ and (c) Adv's execution is identical to the one of Adv'. The only difference between $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ is that $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ can only be called with special input $(\mathsf{cheat}_i, \epsilon_i)$ if $\epsilon_i \geq \epsilon'_{\mathsf{on}}$ while $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ is called with $\epsilon_i \geq \epsilon_{\mathsf{on}}$. $\mathcal{S}_1$ being the correct simulator for the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world will call the ideal functionality always with $\epsilon_i \geq \epsilon_{\mathsf{on}}$. $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$, if only called with $\epsilon_i \geq \epsilon_{\mathsf{on}}$, behaves exactly as $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$, which shows fact (a). If our condition holds, $\mathcal{S}$ determines all of its behavior via $\mathcal{S}_1$, and hence, the two are executed identical (b). Equivalently, if the condition holds, Adv' determines all of its behavior via Adv, and hence, the two are executed identical (c). Note that in both cases even the randomness given to Adv is determined by $\mathcal{S}_1$. From facts (a), (b), (c), it follows that the outputs of the honest parties and the simulator in any two concrete experiments with equivalent inputs in the $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ ideal world and the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world, in which our condition holds, are identical. Hence, observation one is true, i.e., each simulation produced by Adv' when interacting with $\mathcal{S}_1$ in the $\mathcal{F}_{\mathsf{on}}^{\epsilon_{\mathsf{on}}}$ ideal world is identical to the simulation Adv would produce when interacting with $\mathcal{S}$ in the $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ ideal world. The second observation is straight forward. As the protocol, based on our condition, is the same in the $\mathcal{F}_{\mathsf{off}}^{\epsilon_{\mathsf{off}}}$-hybrid world and the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world, the honest parties behave the same. Further, Adv' behaves exactly as Adv without telling Adv in which hybrid world it is. Hence, the behavior of Adv' in the $\mathcal{F}_{\mathsf{off}}^{\epsilon_{\mathsf{off}}}$-hybrid world is the same as the one of Adv in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world. It follows that the output of the honest parties and Adv' in the $\mathcal{F}_{\mathsf{off}}^{\epsilon_{\mathsf{off}}}$-hybrid world is identical to the output of the honest parties and Adv in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world execution. Subsequently, observation two holds, i.e., given our condition, each hybrid world protocol execution produced by Adv' in the $\mathcal{F}_{\mathsf{off}}^{\epsilon_{\mathsf{off}}}$-hybrid world is equivalent to the hybrid world protocol execution Adv would produce in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world.

Claim. The ideal world execution is indistinguishable from the hybrid world execution if the adversary sends $(\mathsf{cheat}_i, \cdot)$ for $i \in \mathcal{I}$ to $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$.

In case (b) of the simulator, the adversary tries to cheat by sending $m = (\mathsf{cheat}_i, \epsilon_i)$ for $i \in \mathcal{I}$ and $\epsilon_i \geq \epsilon'_{\mathsf{off}}$. In the hybrid world, the hybrid functionality $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ signals detected cheating with probability ϵ_i. In the ideal world, simulator $\mathcal{S}$ simulates the hybrid functionality by first asking the online functionality $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ whether or not the cheating is detected. To this end, $\mathcal{S}$ samples dummy inputs $\{\hat{x}_i^{\mathsf{on}}\}_{i\in\mathcal{I}}$ and sends $\{\hat{x}_i^{\mathsf{on}}\}_{i\in\mathcal{I}}$ together with $m = (\mathsf{cheat}_i, \epsilon_i)$ to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$. $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ accepts message m as $\epsilon_i \geq \epsilon'_{\mathsf{on}}$. Note that sampling dummy values for the inputs is no problem, since only the output values $\{\hat{y}_i^{\mathsf{on}}\}$ returned from $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ depend on these values and these output values are not used later on. $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ signals that cheating is detected with probability ϵ_i. It follows that the detection probability is ϵ_i in both worlds.

In case cheating is detected in the hybrid world, the hybrid functionality returns $\mathsf{corrupted}_i$ to the honest parties and sends the outputs of the corrupted parties of the offline function to the adversary. These outputs are computed by evaluating f_{off} on the inputs of the corrupted parties and some freshly and uniformly sampled randomness but without any input from the honest parties. By assumption (the second restriction) the protocol instructs the honest parties to output $\mathsf{corrupted}_i$ upon receiving $\mathsf{corrupted}_i$. In the ideal world, upon receiving $\mathsf{corrupted}_i$ from $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$, $\mathcal{S}$ computes the outputs $\{\hat{y}_i^{\mathsf{off}}\}_{i\in\mathcal{I}}$ of the probabilistic function f_{off} itself using the inputs of the corrupted parties $\{x_i^{\mathsf{off}}\}_{i\in\mathcal{I}}$ received by Adv. As $\mathcal{S}$ has access to Adv's inputs and can sample the randomness provided to f_{off} itself, the computed output of $\mathcal{S}$ is indistinguishable from the output computed by $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$ in the hybrid world. Next, $\mathcal{S}$ sends $(\mathsf{corrupted}_i, \{\hat{y}_i^{\mathsf{off}}\}_{i\in\mathcal{I}})$ to Adv and returns whatever Adv returns. At the same time, the ideal online functionality sends $\mathsf{corrupted}_i$ to the honest parties which output $\mathsf{corrupted}_i$. As the outputs of the honest parties are identical in the hybrid and the ideal world in this case and the view of the adversary is indistinguishable, the resulting output distributions are indistinguishable as well.

In case cheating is undetected, the hybrid functionality in the hybrid world sends $\mathsf{undetected}$ to the adversary. Since the offline hybrid functionality takes no inputs from the honest parties, no inputs are sent to the adversary. Next, the adversary can either return abort, $\mathsf{corrupted}_i$ or a set of output values for the honest parties $\{y_j^{\mathsf{off}}\}_{j\notin\mathcal{I}}$. In the ideal world, the simulator $\mathcal{S}$ sends $\mathsf{undetected}$ to Adv and waits for its reply exactly as in the hybrid world. In case the adversary sends abort or $\mathsf{corrupted}_i$ in the hybrid world, the hybrid offline functionality forwards the message to the honest parties. By assumption (second restriction) the honest parties terminate the protocol by outputting the same message. Similarly, in the ideal world, $\mathcal{S}$ forwards the message to the ideal online functionality which sends the message to the honest parties and instructs them to terminate by outputting the same message. Then, $\mathcal{S}$ terminates the execution by outputting whatever Adv outputs. It follows, that the joint output distributions are indistinguishable in the case that Adv sends abort or $\mathsf{corrupted}_i$ after successful cheating. In case the adversary sends output values for the honest parties $\{y_j^{\mathsf{off}}\}_{j\notin\mathcal{I}}$ in the hybrid world, the hybrid functionality forwards these values to the honest parties. Then, the honest parties continue the execution of the online protocol with the adversary by using their inputs $\{x_j^{\mathsf{on}}\}_{j\notin\mathcal{I}}$ and the outputs of the offline hybrid functionality $\{y_j^{\mathsf{off}}\}_{j\notin\mathcal{I}}$. As a result of the execution, the honest parties may either obtain output values $\{y_j^{\mathsf{on}}\}_{j\notin\mathcal{I}}$ or output abort or $\mathsf{corrupted}_i$ for $i \in \mathcal{I}$. In the ideal world, if $\mathcal{S}$ receives $\{y_j^{\mathsf{off}}\}_{j\notin\mathcal{I}}$ from the adversary, the simulator simulates the rest of the protocol as follows. For every honest party P_j for $j \notin \mathcal{I}$, $\mathcal{S}$ takes party P_j's input obtained from $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ and the value y_j^{off} obtained from Adv to act like the honest party P_j. Since $\mathcal{S}$ knows all inputs and the outputs of the offline hybrid functionality for all honest parties, $\mathcal{S}$ can act exactly like all the honest parties. It follows that the view of the adversary is indistinguishable from the hybrid world execution. Finally, upon obtaining the outputs $\{y_j^{\mathsf{on}}\}_{j \notin \mathcal{I}}$ for the honest parties at the end of the simulation, $\mathcal{S}$ sends these messages to $\mathcal{F}_{\mathsf{on}}^{\epsilon'_{\mathsf{on}}}$ which forwards them to the honest parties. Since the outputs of the honest parties are indistinguishable too, the final joint output distribution is indistinguishable as well. This finishes the argumentation about the indistinguishability of the hybrid world execution and the ideal world execution.

We analyze the deterrence factor ϵ'_{on} for both cases. In case (a), the adversary does not send cheat to the offline functionality and thus cheating happens at most during the remaining steps of the online protocol. In these steps, we can detect cheating with probability at least ϵ_{on} which is given by the deterrence factor of the online protocol in the $\mathcal{F}_{\mathsf{off}}^{1}$-hybrid model. In case (b), the adversary sends cheat_i to the offline functionality in which case cheating is detected with probability at least ϵ'_{off}, since π_{on} is executed in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid model. It follows that the overall deterrence factor of π_{on} in the $\mathcal{F}_{\mathsf{off}}^{\epsilon'_{\mathsf{off}}}$-hybrid world is $\min(\epsilon_{\mathsf{on}}, \epsilon'_{\mathsf{off}})$.

Finally, it is easy to see that in case (b) the simulator runs in polynomial time, since $\mathcal{S}$ behaves like an honest party which also runs in polynomial time. In case (a), $\mathcal{S}$ internally executes $\mathcal{S}_1$. Since $\mathcal{S}_1$ runs in polynomial time per definition, $\mathcal{S}$ runs in polynomial time as well. $\square$

C Comparison of Theorem 1 with [AL07]

Aumann and Lindell [AL07] presented a sequential composition theorem for the (strong) explicit cheat formulation. The theorem shows that a protocol π that is covertly secure in an $(\mathcal{F}_1^{\epsilon_1}, \ldots, \mathcal{F}_{p(n)}^{\epsilon_{p(n)}})$-hybrid world with deterrence factor ϵ_π, i.e., parties have access to a polynomial number of functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_{p(n)}$ with deterrence factor $\epsilon_1, \ldots, \epsilon_{p(n)}$, respectively, is also covertly secure with deterrence ϵ_π if functionality $\mathcal{F}_i$ is replaced by a protocol π_i that realizes $\mathcal{F}_i$ with deterrence factor ϵ_i for $i \in \{1, \ldots, p(n)\}$. This theorem allows to analyze the security of a protocol in a hybrid model and replace the hybrid functionalities with subprotocols afterwards. Aumann and Lindell already noted that the computation of the deterrence factor ϵ_π needs to take all the deterrence factors of the subprotocols into account. However, the theorem does not make any statement about how the individual deterrence factors influence the deterrence factor of the overall protocol and neither analyzes the effect of changing some of the deterrence factors ϵ_i.

Out theorem takes on step further and addresses the aforementioned drawbacks. In particular, it allows to analyze the security of a protocol in a *simple* hybrid world, in which the hybrid functionality is associated with deterrence factor 1. As there is no successful cheating in the hybrid functionality, a proof in this hybrid world is expected to be much simpler. The same holds for the calculation of the overall deterrence factor. Once having proven a protocol to be secure in the simple hybrid world, our theorem allows to derive the security and the deterrence factor of the same protocol in the hybrid world, in which the offline phase is associated with some smaller deterrence factor, $\epsilon' \in [0, 1]$.

D Covert Offline Phase

In this section, we present the covertly secure offline protocol implementing $\mathcal{F}^{\epsilon}_{f_{\mathsf{off}}}$ stated in Section 4 which we base our evaluation on. Following the black-box approach to covert security (resp. publicly verifiable covert security) introduced by prior work [DOS20, FHKS21, SSS22], we start with a semi-honest protocol and transform it into a covert protocol using the cut-and-choose approach.

We use a substring routine $\mathsf{Sub}(a, \ell)$, that takes as input a bit-string a and an index ℓ and outputs the substring b composed of the first ℓ elements of a.

Oblivious transfer. We use an ideal functionality to compute random correlated OTs of dynamic length. The functionality generates n_1 random correlated OT tuples. The first n_2 of them have messages of length l_2 while the remaining $n_1 - n_2$ have messages of length l_1. The functionality is defined as follows:

Functionality $\mathcal{F}_{\mathsf{OT}}$: Random Correlated Dynamic Length String OT

The functionality involves a sender $\mathcal{S}$ and a receiver $\mathcal{R}$.
Upon receiving $(\mathsf{ok}, n_1, l_1, n_2, l_2)$ with $n_1 \geq n_2$ and $l_2 \geq l_1$ from both $\mathcal{S}$ and $\mathcal{R}$, the functionality

- defines $\mathcal{S} := [n_1] \setminus [n_2]$ and samples $R \in \{0,1\}^{l_2}$, $b_i \in \{0,1\}$ for $i \in [n_1]$, $m_i^2 \in \{0,1\}^{l_2}$ for $i \in [n_2]$ and $m_i^1 \in \{0,1\}^{l_1}$ for $i \in \mathcal{S}$,
- computes $o_i^2 := m_i^2 \oplus b_i \cdot R$ for $i \in [n_2]$ and $o_i^1 := m_i^1 \oplus b_i \cdot \mathsf{Sub}(R, l_1)$ for $i \in \mathcal{S}$,
- outputs $(R, \{m_i^1\}_{i \in \mathcal{S}}, \{m_i^2\}_{i \in [n_2]})$ to $\mathcal{S}$ and $(\{b_i\}_{i \in [n_1]} \{o_i^1\}_{i \in \mathcal{S}}, \{o_i^2\}_{i \in [n_2]})$ to $\mathcal{R}$.

A malicious sender may pick R and m_*^* itself. A malicious receiver may pick b_* itself.

This functionality can be efficiently instantiated by adapting the IKNP OT extension protocol [IKNP03] in a straightforward way. In the original protocol, the OT receiver sends correction terms correcting all message pairs of the base OT to share the same difference x which represents the choice bits of the receiver. In the dynamic length version, the receiver does not correct all message pairs completely. Instead, it corrects the first l_1 message pairs completely (all n_1 bits) and corrects the remaining pairs just up to length n_2. This way, the OT receiver only sends l_1 correction terms of length n_1 bits and $(l_2 - l_1)$ correction terms of length n_2 bits. This yields a communication complexity of γ bits per γ-bit OT for $\gamma \in \{l_1, l_2\}$ plus the communication complexity of the base OTs. As we are only interested in random choice bits (represented by x), we can define x to be the difference of the first message pair, and hence, fix the first correction term to be the 0-string of length n_1. Thus, we can reduce the communication complexity to $(\gamma - 1)$ per γ-bit OT plus the communication complexity of the base OTs. Obviously, this approach is only beneficial if $l_2 \leq \kappa$ and $l_1 < \kappa$. Otherwise, we could generate κ-bit OTs and extend them to longer OTs using a pseudo random generator seeded by the κ-bit message.

Commitments. The covert protocol uses an extractable commitment scheme $(\mathsf{Commit}, \mathsf{Open})$ that is *computationally binding and hiding.* A party commits to a message m by computing $(c, d) \leftarrow \mathsf{Commit}(m)$, where c is the commitment value and d denotes the decommitment or opening value. Similarly, a party opens a commitment c with decommitment d by computing $m' \leftarrow \mathsf{Open}(c, d)$. It holds with overwhelming probability that either $m' = m$ or $m' = \bot$. The extractability property allows the simulator to extract the committed message m and the opening value d from the commitment c by using some trapdoor information.

Such a scheme can be implemented in the random oracle model by defining $\mathsf{Commit}(x; r) = H(i, x, r)$ where i is the identity of the committer, $H : \{0,1\}^* \rightarrow \{0,1\}^{2\kappa}$ is a random oracle and $r \in_R \{0,1\}^\kappa$.

Semi-honest instances. The semi-honest protocol which is executed several times by the covert protocol via the cut-and-choose approach is defined as follows:

Protocol $\Pi_{\mathsf{sh\text{-}off}}$: Semi-honest triple generation

The protocol is executed between parties $\mathcal{A}$ and $\mathcal{B}$. Both parties do not have any input (the adversary is allowed to pick its own random tape). It makes use of an ideal functionality $\mathcal{F}_{\mathsf{OT}}$ for random correlated oblivious transfer for strings of dynamic length, a hash function H. When denoting a particular party with P, we denote the respective other party with $\bar{P}$.

Public parameters: The computational security parameter κ, the desired MAC length l, the number of one-sided authenticated bits for each party n_1, and the number of two-sided authenticated triples n_2.

Generation of authenticated bits:

1. The parties compute $n_3 = 3 \cdot n_2 + n_1$ and invoke $\mathcal{F}_{\mathsf{OT}}$ with input $(\mathsf{ok}, n_3, l, n_2, \kappa)$ twice, once with $\mathcal{A}$ as sender and once with $\mathcal{B}$ as sender. As a result, each party $P \in \{\mathcal{A}, \mathcal{B}\}$ receives $(R^P, \{m_i^{(1,P)}\}_{i \in \mathcal{S}}, \{m_i^{(2,P)}\}_{i \in [n_2]})$ in the execution, in which it acts as sender, and $(\{b_i^P\}_{i \in [n_3]}, \{o_i^{(1,P)}\}_{i \in \mathcal{S}}, \{o_i^{(2,P)}\}_{i \in [n_2]})$ in the execution, in which it acts as receiver, for $\mathcal{S} := [n_3] \setminus [n_2]$.
2. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$ defines $m_i^{(1,P)} := \mathsf{Sub}(m_i^{(2,P)}, l)$ and $o_i^{(1,P)} := \mathsf{Sub}(o_i^{(2,P)}, l)$ for $i \notin \mathcal{S}$. This way $m_i^{(1,P)}$ and $o_i^{(1,P)}$ are defined for $i \in [n_3]$.
3. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$ defines $\Delta_P := \mathsf{Sub}(R^P, l)$, $x_i^P := b_j^P$ (for $i \in [n_3]$), $M[x_i^P] := o_i^{(1,P)}$ (for $i \in [n_3]$), $K[x_i^{\bar{P}}] := m_i^{(1,P)}$ (for $i \in [n_3]$), $\Delta'_P := R^P$, $M'[x_i^P] := o_i^{(2,P)}$ (for $i \in [n_2]$) and $K'[x_i^{\bar{P}}] := m_i^{(2,P)}$ (for $i \in [n_2]$). *Note that $\Delta'_P, M'[\cdot]$ and $K'[\cdot]$ are extensions of the short MACs resp. keys $\Delta_P, M[\cdot]$ and $K[\cdot]$, i.e., for each v for which $M'[v]$ is defined it holds that* $\mathsf{Sub}(\Delta'_P, l) = \Delta_P$, $\mathsf{Sub}(M'[v], l) = M[v]$ *and* $\mathsf{Sub}(K'[v], l) = K[v]$.
4. For each $i \in [n_3]$, the parties jointly define authenticated bits $\langle r_i \rangle^{\mathcal{A}} := (x_j^{\mathcal{A}}, K[x_j^{\mathcal{A}}], M[x_j^{\mathcal{A}}])$ and $\langle s_i \rangle^{\mathcal{B}} := (x_j^{\mathcal{B}}, K[x_j^{\mathcal{B}}], M[x_j^{\mathcal{B}}])$ (each party defines its share of the authenticated bit).

Turn bits into triples: The parties execute the following for $i \in [n_2]$:

4. Define $j_1 := i$, $j_2 := n_2 + i$, $j_3 := 2 \cdot n_2 + i$, $\langle a_{\mathcal{A}} \rangle^{\mathcal{A}} := \langle r_{j_1} \rangle^{\mathcal{A}}$, $\langle a_{\mathcal{B}} \rangle^{\mathcal{B}} := \langle s_{j_1} \rangle^{\mathcal{B}}$, $\langle b_{\mathcal{A}} \rangle^{\mathcal{A}} := \langle r_{j_2} \rangle^{\mathcal{A}}$, $\langle b_{\mathcal{B}} \rangle^{\mathcal{B}} := \langle s_{j_2} \rangle^{\mathcal{B}}$, $\langle y_{\mathcal{A}} \rangle^{\mathcal{A}} := \langle r_{j_3} \rangle^{\mathcal{A}}$, and $\langle y_{\mathcal{B}} \rangle^{\mathcal{B}} := \langle s_{j_3} \rangle^{\mathcal{B}}$.
5. The parties execute a semi-honest bit oblivious transfer with $\mathcal{A}$ as sender and $\mathcal{B}$ as receiver. $\mathcal{A}$ samples a random bit $z_1^{\mathcal{A}}$ and inserts $z_1^{\mathcal{A}}$ and $z_1^{\mathcal{A}} \oplus b_{\mathcal{A}}$. $\mathcal{B}$ inserts choice bits $a_{\mathcal{B}}$ and receives $z_2^{\mathcal{B}} := z_1^{\mathcal{A}} \oplus a_{\mathcal{B}} \cdot b_{\mathcal{A}}$. The oblivious transfer is efficiently instantiated as follows:

(a) $\mathcal{A}$ sends $H_0 := \mathsf{Sub}(H(K'[a_{\mathcal{B}}]), 1) \oplus z_1^{\mathcal{A}}$ and $H_1 := \mathsf{Sub}(H(K'[a_{\mathcal{B}}] \oplus \Delta'_{\mathcal{A}}), 1) \oplus z_1^{\mathcal{A}} \oplus b_{\mathcal{A}}$. $\mathcal{B}$ calculates $z_2^{\mathcal{B}} := H_{x_{\mathcal{B}}} \oplus \mathsf{Sub}(H(M'[x_{\mathcal{B}}]), 1)$.

6. The parties execute another semi-honest bit oblivious transfer with $\mathcal{B}$ as sender and $\mathcal{A}$ as receiver. $\mathcal{B}$ samples a random bit $z_1^{\mathcal{B}}$ and inserts $z_1^{\mathcal{B}}$ and $z_1^{\mathcal{B}} \oplus b_{\mathcal{B}}$. $\mathcal{A}$ inserts $a_{\mathcal{A}}$ and receives $z_2^{\mathcal{A}} := z_1^{\mathcal{B}} \oplus a_{\mathcal{A}} \cdot b_{\mathcal{B}}$. The oblibious transfer is instantiated as above.
7. $\mathcal{A}$ defines $c_{\mathcal{A}} := z_1^{\mathcal{A}} \oplus z_2^{\mathcal{A}} \oplus a_{\mathcal{A}} \cdot b_{\mathcal{A}}$. $\mathcal{B}$ defines $c_{\mathcal{B}} := z_1^{\mathcal{B}} \oplus z_2^{\mathcal{B}} \oplus a_{\mathcal{B}} \cdot b_{\mathcal{B}}$. Hence, $c_{\mathcal{A}} \oplus c_{\mathcal{B}} = z_1^{\mathcal{A}} \oplus z_1^{\mathcal{B}} \oplus a_{\mathcal{A}} \cdot b_{\mathcal{B}} \oplus a_{\mathcal{A}} \cdot b_{\mathcal{A}} \oplus z_1^{\mathcal{B}} \oplus z_1^{\mathcal{A}} \oplus a_{\mathcal{B}} \cdot b_{\mathcal{A}} \oplus a_{\mathcal{B}} \cdot b_{\mathcal{B}} = a_{\mathcal{A}} \cdot b_{\mathcal{A}} \oplus a_{\mathcal{A}} \cdot b_{\mathcal{B}} \oplus a_{\mathcal{B}} \cdot b_{\mathcal{A}} \oplus a_{\mathcal{B}} \cdot b_{\mathcal{B}} = (a_{\mathcal{A}} \oplus a_{\mathcal{B}}) \cdot (b_{\mathcal{A}} \oplus b_{\mathcal{B}})$.
8. The parties authenticate values $c_{\mathcal{A}}$ and $c_{\mathcal{B}}$ as follows. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$ sends $d_P := c_P \oplus y_P$. Then the parties non-interactively (see Section 4) authenticate constant d_P to $\langle d_P \rangle^P$ and calculate $\langle c_P \rangle^P := \langle y_P \rangle^P \oplus \langle d_P \rangle^P$.
9. The parties define $\langle \alpha_i \rangle := \langle a_{\mathcal{A}} | a_{\mathcal{B}} \rangle$, $\langle \beta_i \rangle := \langle b_{\mathcal{A}} | b_{\mathcal{B}} \rangle$ and $\langle \gamma_i \rangle := \langle c_{\mathcal{A}} | c_{\mathcal{B}} \rangle$.

Output:

11. The parties output their respective shares of $\{(\langle r_i \rangle^{\mathcal{A}}, \langle s_i \rangle^{\mathcal{B}})\}_{i \in [n_1]}$ and $\{(\langle \alpha_i \rangle, \langle \beta_i \rangle, \langle \gamma_i \rangle_{i \in [n_2]})\}$.

The covert protocol. The semi-honest protocol is transformed into a covert protocol as follows:

Protocol $\mathit{\Pi}_{\mathsf{cov\text{-}off}}$: Covert triple generation

The protocol is executed between parties $\mathcal{A}$ and $\mathcal{B}$. Both parties do not have any input (the adversary is allowed to pick its own random tape). It makes use of a commitment scheme $(\mathsf{Commit}, \mathsf{Open})$ and the semi-honest protocol $\mathit{\Pi}_{\mathsf{sh\text{-}off}}$. When denoting a particular party with P, we denote the respective other party with $\bar{P}$.
Public parameters: The computational security parameter κ, the cut-and-choose parameter t, the mac length l, the number of one-sided authenticated bits of each party n_1 and the number of two-sided authenticated triples n_2. The deterrence factor of the offline phase is $\epsilon_{\mathsf{off}} = 1 - \frac{1}{t}$.
The protocol:

1. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$ samples t random seeds $\mathsf{seed}_i^P \in \{0,1\}^\kappa$, samples a random coin $\mathsf{coin}^P \in [t]$, computes commitments $\{(c_i^P, d_i^P) \leftarrow \mathsf{Commit}(\mathsf{seed}_i^P)\}_{i \in [t]}$ and $c^P \leftarrow \mathsf{Commit}(\mathsf{coin}^P)$, and sends $(\{c_i^P\}_{i \in [t]}, c^P)$.
2. The parties execute t instances of the protocol $\mathit{\Pi}_{\mathsf{sh\text{-}off}}$. In the i-th instance, party P derives all randomness from seed seed_i^P and receives transcript $\mathsf{transcript}_i$ and output out_i^P.
3. In case a party aborts before this step, i.e., does not send a valid and timely message, the other party outputs abort and aborts.
4. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$ sends (d^P) and receives $(d^{\bar{P}})$, Then, P computes $\mathsf{coin}^{\bar{P}} \leftarrow \mathsf{Open}(c^{\bar{P}}, d^{\bar{P}})$, and $\mathsf{coin} := \mathsf{coin}^P \oplus \mathsf{coin}^{\bar{P}} \mod t$. If $\mathsf{coin}^{\bar{P}} := \perp$ or $\bar{P}$ aborted, P outputs $\mathsf{corrupted}_{\bar{P}}$ and aborts.
5. Each party $P \in \{\mathcal{A}, \mathcal{B}\}$ sends $\{d_i^P\}_{i \in [t] \setminus \{\mathsf{coin}\}}$. Upon receiving $\{d_i^{\bar{P}}\}_{i \in [t] \setminus \{\mathsf{coin}\}}$, P calculates $\mathsf{seed}_i^{\bar{P}} \leftarrow \mathsf{Open}(c_i^{\bar{P}}, d_i^{\bar{P}})$ for each $i \in [t] \setminus \{\mathsf{coin}\}$. If any $\mathsf{seed}_i^{\bar{P}} = \perp$ or $\bar{P}$ aborted, P outputs $\mathsf{corrupted}_{\bar{P}}$ and aborts. Otherwise, for each $i \in [t] \setminus \{\mathsf{coin}\}$, P locally emulates the semi-honest protocol $\mathit{\Pi}_{\mathsf{sh\text{-}off}}$ with randomness seed_i^P and $\mathsf{seed}_i^{\bar{P}}$ and receives transcripts $\widetilde{\mathsf{transcript}}_i$. If any $\widetilde{\mathsf{transcript}}_i \neq \mathsf{transcript}_i$ P outputs $\mathsf{corrupted}_{\bar{P}}$ and aborts. Otherwise, P outputs $\mathsf{out}_{\mathsf{coin}}^P$.

E Proof of Theorem 2

This section provides the proof of Theorem 2 in Section 4.

Proof. We prove covert security by defining an ideal-world simulator $\mathcal{S}$ using a real-world adversary Adv in a black-box way as subroutine and playing the role of the corrupted parties when interacting with the ideal covert-functionality $\mathcal{F}_{\mathsf{Cov}}$. The proof is given in the $(\mathcal{F}_{\mathsf{off}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid world. Without loss of generality, we assume that party $\mathcal{A}$ is corrupted. The simulator $\mathcal{S}$ is fully specified as follows.

1. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{off}}$ as specified but stores both the output for $\mathcal{A}$ and the one for $\mathcal{B}$. If Adv sends special input $\mathsf{abort}_{\mathcal{A}}$ or $(\mathsf{cheat}_{\mathcal{A}}, \cdot)$, $\mathcal{S}$ sends a dummy input $(\bot)$ and special input $\mathsf{abort}_{\mathcal{A}}$ or $(\mathsf{cheat}_{\mathcal{A}}, 1)$ to $\mathcal{F}_{\mathsf{Cov}}$, terminates the experiment and outputs whatever Adv outputs.
2. $\mathcal{S}$ picks an empty dummy input 0^{n_1} for the honest party $\mathcal{B}$ and executes Step 2 as specified by the protocol. Further, when receiving $d_{(i,\mathcal{A})}$ from Adv, $\mathcal{S}$ extracts the actually input provided for $\mathcal{A}$, $\{\tilde{x}_{(i,\mathcal{A})} := d_{(i,\mathcal{A})} \oplus r_{(i,\mathcal{A})}\}_{i\in[n_1]}$ (for $i \in [n_1]$). Using its knowledge of the precomputation, $\mathcal{S}$ computes all components of $\langle w_i \rangle$ $(w_i^{\mathcal{A}}, w_i^{\mathcal{B}}, K[w_i^{\mathcal{A}}], K[w_i^{\mathcal{B}}], M[w_i^{\mathcal{A}}], M[w_i^{\mathcal{B}}])$ for $i \in \mathcal{I}_{\mathcal{A}}^{\mathsf{in}} \cup \mathcal{I}_{\mathcal{B}}^{\mathsf{in}}$. Additionally, $\mathcal{S}$ initializes an empty set $\mathcal{E}$ to keep track of the flipped bits sent by Adv during the circuit evaluation.
3. $\mathcal{S}$ executes Step 3 as an honest party would do but monitors the computation performed by Adv using its knowledge of the precomputation and the already computed wire values. In particular, $\mathcal{S}$ computes all components of $\langle w_j \rangle$. However, in case of an AND-Gate, the resulting $\langle w_j \rangle$ is not computed based on the $e^{\mathcal{A}}$, $d^{\mathcal{A}}$ values that Adv is supposed to send but on the values $\tilde{e}^{\mathcal{A}}$, $\tilde{d}^{\mathcal{A}}$ that Adv actually sends. This means that $\mathcal{S}$ does not make use of its knowledge of Adv's secrets to influence the computation. During the computation of the i-th AND-Gate, if Adv sends incorrect values for $\tilde{e}^{\mathcal{A}} \neq e^{\mathcal{A}}$ resp. $\tilde{d}^{\mathcal{A}} \neq d^{\mathcal{A}}$, $\mathcal{S}$ adds $(i, \text{`}e\text{'})$ resp. $(i, \text{`}d\text{'})$ to $\mathcal{E}$. *In the following, we will denote the e-value resp. the d-value that* Adv *is supposed to send in the evaluation of the i-th AND-gate by $e_i^{\mathcal{A}}$ resp. $d_i^{\mathcal{A}}$ and the correct MAC on these values by $M[e_i^{\mathcal{A}}]$ resp. $M[d_i^{\mathcal{A}}]$.*
4. $\mathcal{S}$ computes $\mathcal{M}^1_{(\mathcal{B},\mathcal{B})}$ and $\mathcal{M}^1_{(\mathcal{B},\mathcal{A})}$ as specified by the protocol and sends $(\mathcal{M}^1_{(\mathcal{B},\mathcal{B})})$. Upon receiving $(\mathcal{M}^1_{(\mathcal{A},\mathcal{A})})$, $\mathcal{S}$ backtracks the preimage of $\mathcal{M}^1_{(\mathcal{A},\mathcal{A})}$, $\{(\widetilde{M}[e_i^{\mathcal{A}}], \widetilde{M}[d_i^{\mathcal{A}}])\}_{i\in[n_2]}$ and differentiates the following cases:
   - (i) *(Failure):* In case there are several such preimages, $\mathcal{S}$ returns (fail).
   - (ii) *(Early abort):* If Adv aborted before this step, $\mathcal{S}$ sends (abort) to $\mathcal{F}_{\mathsf{Cov}}$, terminates the experiment and outputs whatever Adv outputs.
   - (iii) *(Blatant cheat):* Informally, a blatant cheating attempt, which is always detected, happens if Adv (a) modifies the MACs of at least one bit that has not been flipped during the evaluation, or (b) applies different global keys to the MACs of two flipped bits. In this case, $\mathcal{S}$ sends $(\mathsf{cheat}_{\mathcal{A}}, 1, \bot)$ to $\mathcal{F}_{\mathsf{Cov}}$, terminates the experiment and outputs whatever Adv outputs. Formally, the cases are defined as follows:
     - (a) If $\exists i \in [n_2]$ s.th. $(i, \cdot) \notin \mathcal{E}$ and either $\widetilde{M}[e_i^{\mathcal{A}}] \neq M[e_i^{\mathcal{A}}]$ or $\widetilde{M}[d_i^{\mathcal{A}}] \neq M[d_i^{\mathcal{A}}]$.
     - (b) If $\exists i, j \in [n_2]$ s.th. $i \neq j$, $(i, v) \in \mathcal{E}$, $(j, u) \in \mathcal{E}$ and $\widetilde{M}[v_i^{\mathcal{A}}] \oplus M[v_i^{\mathcal{A}}] \neq \widetilde{M}[u_j^{\mathcal{A}}] \oplus M[u_j^{\mathcal{A}}]$
   - (iv) *(Consistent cheat):* If $\mathcal{E} \neq \emptyset$ and none of the above has occurred, $\mathcal{S}$ picks the first element (i, v) from $\mathcal{E}$ to compute the guessed global key of $\mathcal{B}$, $\widetilde{\Delta} = \widetilde{M}[v_i^{\mathcal{A}}] \oplus$

$M[v_i^{\mathcal{A}}]$. Then, $\mathcal{S}$ sends input $(\{\tilde{x}_{(i,\mathcal{A})}\}_{i\in[n_1]})$ and special input $(\mathsf{cheat}_{\mathcal{A}}, \epsilon)$ to $\mathcal{F}_{\mathsf{Cov}}$. If $\mathcal{F}_{\mathsf{Cov}}$ replies with $(\mathsf{corrupted}_{\mathcal{A}}, \cdot)$, $\mathcal{S}$ terminates the experiment and outputs whatever Adv outputs. Otherwise, $\mathcal{S}$ continues as following:

(a) When receiving $(\mathsf{undetected}, \{x_{(i,\mathcal{B})}\}_{i\in[n_1]})$ from $\mathcal{F}_{\mathsf{Cov}}$, $\mathcal{S}$ emulates the interactive circuit evaluation *in his head.* The emulation is performed based on input bits $(\{\tilde{x}_{(i,\mathcal{A})}\}_{i\in[n_1]})$ and $(\{x_{(i,\mathcal{B})}\}_{i\in[n_1]})$. In the emulation $\mathcal{S}$ fixes all values received by Adv in the previous execution including the precomputation phase. In particular, $\mathcal{S}$ fixes $\mathcal{A}$'s output of the precomputation, the bits Adv receives during the input phase, $(\{d_{(i,\mathcal{B})}\}_{i\in[n_1]})$, and the bits Adv receives during the circuit evaluation phase, the values $(e^{\mathcal{B}}, d^{\mathcal{B}})$ for each AND-gate. To do so while still ensuring correctness of $\mathcal{B}$'s input and the multiplication triples, $\mathcal{S}$ adapts $\mathcal{B}$'s share of the precomputation accordingly in an ad-hoc way. In addition, each $e_{\mathcal{A}}$ or $d_{\mathcal{A}}$ value, that has been flipped by Adv during the evaluation of the AND-gates, is flipped in the emulation as well. This way, $\mathcal{S}$ obtains the output bits $(\{z_{(i,\mathcal{A})}\}_{i\in[n_1]}, \{z_{(i,\mathcal{B})}\}_{i\in[n_1]})$.

(b) $\mathcal{S}$ sends $(\mathsf{Committed}, \mathcal{B})$ to Adv. When receiving $(\mathsf{Commit}, (\{\tilde{w}_i^{\mathcal{A}}\}, \mathcal{M}^2_{(\mathcal{A},\mathcal{A})}))$ from Adv, $\mathcal{S}$ backtracks the preimage of $\mathcal{M}^2_{(\mathcal{A},\mathcal{A})}$, $\{\widetilde{M}[w_i^{\mathcal{A}}]\}_{i\in\mathcal{I}_{\mathcal{B}}^{\mathsf{out}}}$, and defines $\{u_j\}_{j\in[n_1]} := \{\tilde{w}^{\mathcal{A}}_{\mathcal{I}_{\mathcal{B}}^{\mathsf{out}}[j]} \oplus w^{\mathcal{A}}_{\mathcal{I}_{\mathcal{B}}^{\mathsf{out}}[j]}\}_{j\in[n_1]}$. In case there are several such preimages, $\mathcal{S}$ returns (fail). Then, $\mathcal{S}$ checks for consistent cheating. In particular, if there exists $i \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}$ such that $\tilde{w}_i^{\mathcal{A}} = w_i^{\mathcal{A}}$ and $\widetilde{M}[w_i^{\mathcal{A}}] \neq M[w_i^{\mathcal{A}}]$ or $\tilde{w}_i^{\mathcal{A}} \neq w_i^{\mathcal{A}}$ and $\widetilde{M}[w_i^{\mathcal{A}}] \neq M[w_i^{\mathcal{A}}] \oplus \widetilde{\Delta}$, $\mathcal{S}$ defines $\mathsf{consistent} := \mathsf{false}$. Otherwise, $\mathcal{S}$ defines $\mathsf{consistent} := \mathsf{true}$. *Note that $M[w_i^{\mathcal{A}}]$ denotes the correct MAC on the unflipped bit $w_i^{\mathcal{A}}$.*

(c) $\mathcal{S}$ forces $\mathcal{A}$'s output to match $\mathcal{S}$'s emulation by defining for each $j \in [n_1]$, $\tilde{w}^{\mathcal{B}}_{\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}[j]} := z_{(j,\mathcal{A})} \oplus w^{\mathcal{A}}_{\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}[j]}$ and $\widetilde{M}[w^{\mathcal{B}}_{\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}[j]}] := K[w^{\mathcal{B}}_{\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}[j]}] \oplus \tilde{w}^{\mathcal{B}}_{\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}[j]} \cdot \Delta_{\mathcal{A}}$ and sending $(\mathsf{Opened}, \mathcal{B}, (\{\tilde{w}_i^{\mathcal{B}}\}_{i\in\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}}, H(\{\widetilde{M}[w_i^{\mathcal{B}}]\}_{i\in\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}})))$ to Adv.

(d) Upon receiving (Open) from Adv, $\mathcal{S}$ sends $(\{z_{(i,\mathcal{B})} \oplus u_i\}_{i\in[n_1]}$ to $\mathcal{F}_{\mathsf{Cov}}$, if $\mathsf{consistent} = \mathsf{true}$, and $\mathsf{corrupted}_{\mathcal{A}}$, otherwise. Either way, $\mathcal{S}$ outputs whatever Adv outputs.

(e) If Adv does not send the expected message (timely) in any step before $\mathcal{S}$ received $(\mathsf{Commit}, \cdot)$ from Adv, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{Cov}}$. If the same happens after $\mathcal{S}$ received $(\mathsf{Commit}, \cdot)$ from Adv, $\mathcal{S}$ sends $\mathsf{corrupted}_{\mathcal{A}}$ to $\mathcal{F}_{\mathsf{Cov}}$. In both cases, $\mathcal{S}$ terminates the experiment and outputs whatever Adv outputs.

(v) *(Tentatively honest behavior):* If $\mathcal{E} = \emptyset$ and $\mathcal{M}^1_{(\mathcal{A},\mathcal{A})} = \mathcal{M}^1_{(\mathcal{B},\mathcal{A})}$, $\mathcal{S}$ sends $(\mathsf{Committed}, \mathcal{B})$ to Adv. Then, $\mathcal{S}$ waits to receive $(\mathsf{Commit}, (\{\tilde{w}_i^{\mathcal{A}}\}_{i\in\mathcal{I}_{\mathcal{P}}^{\mathsf{out}}}, \mathcal{M}^2_{(\mathcal{A},\mathcal{A})}))$ from Adv, backtracks the primage of $\mathcal{M}^2_{(\mathcal{A},\mathcal{A})}$, $\{\widetilde{M}[w_i^{\mathcal{A}}]\}_{i\in\mathcal{I}_{\mathcal{B}}^{\mathsf{out}}}$, defines $\mathsf{tmp} = 0$ and differentiates the following cases:

(a) In case there are several such preimages, $\mathcal{S}$ returns (fail).

(b) If Adv aborted before sending $(\mathsf{Commit}, \cdot)$, $\mathcal{S}$ sends (abort) to $\mathcal{F}_{\mathsf{Cov}}$, terminates the experiment and outputs whatever Adv outputs.

(c) If for each $i \in \mathcal{I}_{\mathcal{B}}$ it holds that $\tilde{w}_i^{\mathcal{A}} = w_i^{\mathcal{A}}$ and $\widetilde{M}[w_i^{\mathcal{A}}] = M[w_i^{\mathcal{A}}]$, $\mathcal{S}$ sends $(\{\tilde{x}_{(i,\mathcal{A})}\}_{i\in[n_1]})$ to $\mathcal{F}_{\mathsf{Cov}}$, waits to receive $(\{z_{(i,\mathcal{A})}\}_{i\in[n_1]})$ from $\mathcal{F}_{\mathsf{Cov}}$, sets $\mathsf{tmp} = 1$ and continues with (f).

(d) If $\exists(i,j) \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}$ such that $i \neq j$, $\tilde{w}_i^{\mathcal{A}} \neq w_i^{\mathcal{A}}$, $\tilde{w}_j^{\mathcal{A}} \neq w_j^{\mathcal{A}}$, and $\widetilde{M}[w_i^{\mathcal{A}}] \oplus M[w_i^{\mathcal{A}}] \neq \widetilde{M}[w_i^{\mathcal{A}}] \oplus M[w_j^{\mathcal{A}}]$, $\mathcal{S}$ sends input $(\{\tilde{x}_i\}_{i\in[n_1]})$ and special input $(\mathsf{cheat}_{\mathcal{A}}, 1)$ to $\mathcal{F}_{\mathsf{Cov}}$, waits to receive $(\{z_{(i,\mathcal{A})}\}_{i\in[n_1]})$ from $\mathcal{F}_{\mathsf{Cov}}$, sets $\mathsf{tmp} = 2$, and continues with (f).

(e) Otherwise, $\mathcal{S}$ defines $\{u_j\}_{j\in[n_1]} := \{\tilde{w}^{\mathcal{A}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{B}}[j]} \oplus w^{\mathcal{A}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{B}}[j]}\}_{j\in[n_1]}$, sends input $(\{\tilde{x}_i\}_{i\in[n_1]})$ and special input $(\mathsf{cheat}_{\mathcal{A}}, \epsilon)$ to $\mathcal{F}_{\mathsf{Cov}}$, and reacts to the possible replies as follows:
   - If $\mathcal{F}_{\mathsf{Cov}}$ replies with $(\mathsf{corrupted}_{\mathcal{A}}, \{z_{(i,\mathcal{A})}\}_{i\in[n_1]})$, $\mathcal{S}$ sets $\mathsf{tmp} = 3$ and continues with (f).
   - If $\mathcal{F}_{\mathsf{Cov}}$ replies with $(\mathsf{undetected}, \{x_{(i,\mathcal{B})}\}_{i\in[n_1]})$, $\mathcal{S}$ computes $(\{z_{(i,\mathcal{A})}\}_{i\in[n_1]}, \{z_{(i,\mathcal{B})}\}_{i\in[n_1]}) = f(\{\tilde{x}_{(i,\mathcal{A})}\}_{i\in[n_1]}, \{x_{(i,\mathcal{B})}\}_{i\in[n_1]})$, $\{\tilde{z}_{(i,\mathcal{B})} := z_{(i,\mathcal{B})} \oplus u_i\}_{i\in[n_1]}$, sets $\mathsf{tmp} = 4$, and continues with (f).

(f) $\mathcal{S}$ defines $\{\tilde{w}^{\mathcal{B}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}[j]} := w^{\mathcal{A}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}[j]} \oplus z_{(j,\mathcal{A})}\}_{j\in[n_1]}$ and $\{\widetilde{M}[w^{\mathcal{B}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}[j]}] := K[w^{\mathcal{B}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}[j]}] \oplus \tilde{w}^{\mathcal{B}}_{\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}[j]} \cdot \Delta_{\mathcal{A}}\}_{j\in[n_1]}$. Then, $\mathcal{S}$ sends $(\mathsf{Opened}, \mathcal{B}, (\{\tilde{w}^{\mathcal{B}}_i\}_{i\in\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}}, H(\{\widetilde{M}[w^{\mathcal{B}}_i]\}_{i\in\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}})))$ to Adv. If Adv aborts and $\mathsf{tmp} \notin \{2,3\}$, $\mathcal{S}$ sends $(\mathsf{corrupted}_{\mathcal{A}})$ to $\mathcal{F}_{\mathsf{Cov}}$. If Adv sends $(\mathsf{Open}, \mathcal{A})$ and $\mathsf{tmp} = 1$, $\mathcal{S}$ sends (ok) to $\mathcal{F}_{\mathsf{Cov}}$. If Adv sends $(\mathsf{Open}, \mathcal{A})$ and $\mathsf{tmp} = 4$, $\mathcal{S}$ sends $(\{\tilde{z}_{(i,\mathcal{B})}\}_{i\in[n_1]})$ to $\mathcal{F}_{\mathsf{Cov}}$. Either way, $\mathcal{S}$ outputs whatever Adv outputs.

In order to finish the proof, we show that the joint distribution of the output of the adversary Adv and the honest party $\mathcal{B}$ in the ideal world is computationally indistinguishable from the output of Adv and $\mathcal{B}$ in the real world. We prove this via a sequence of hybrid experiments and sketch the indistinguishability between the outputs for each two subsequent hybrids.

Hybrid-0: $\mathcal{S}$ impersonates the ideal functionality $\mathcal{F}_{\mathsf{Cov}}$ and the honest party $\mathcal{B}$. $\mathcal{S}$ does not instruct $\mathcal{B}$ via $\mathcal{F}_{\mathsf{Cov}}$ but does so directly (just tells $\mathcal{B}$ what to output). As all involved parties still behave the same, the output of this hybrid and the ideal world is identically distributed.
Hybrid-1: In Step 4)-(iv), $\mathcal{S}$ (impersonating $\mathcal{F}_{\mathsf{Cov}}$) decides on successful cheating iff $\Delta_{\mathcal{B}} = \widetilde{\Delta}$. The hybrids are identically distributed.
Hybrid-2: In Step 2), $\mathcal{S}$ does not use a dummy input 0^{n_1} for the honest party $\mathcal{B}$ but uses $\mathcal{B}$'s input $\{x_{(i,\mathcal{B})}\}$. The hybrids are identically distributed.
Hybrid-3: We merge the case of blatant cheating (Step 4)-(iii)) and detection during a consistent cheating attempt (part of Step 4)-(iv)). $\mathcal{S}$ redefines Step 4)-(iii) as follows. If $\exists i$ such that $\widetilde{M}[e^{\mathcal{A}}_i] \neq K[e^{\mathcal{A}}_i] \oplus \tilde{e}^{\mathcal{A}}_i \cdot \Delta_{\mathcal{B}}$ or $\widetilde{M}[d^{\mathcal{A}}_i] \neq K[d^{\mathcal{A}}_i] \oplus \tilde{d}^{\mathcal{A}}_i \cdot \Delta_{\mathcal{B}}$, $\mathcal{S}$ instructs $\mathcal{B}$ to output $(\mathsf{corrupted}_{\mathcal{A}})$. Further $\mathcal{S}$ redefines Step 4)-(iv) as follows. If $\mathcal{E} \neq \emptyset$ and none of the above cases occurred, continue with Step 4)-(iv)-(a). The hybrids are identically distributed. In particular, blatant cheating and detected cheating ($\widetilde{\Delta} \neq \Delta_{\mathcal{B}}$) implies that there exists an i satisfying the above condition.
Hybrid-4: Following up on the previous hybrid, we do not check the individual MACs during the check inserted in the previous hybrid, but the hash of the MACs that are supposed to be sent, $\mathcal{M}^1_{(\mathcal{A},\mathcal{A})}$. The hybrids are computationally indistinguishable due to the collision resistance of the random oracle.
Hybrid-5: In Step 4)-(iv)-a), $\mathcal{S}$ uses the output wires of the real protocol execution to determine the outputs of both parties, i.e., $\mathcal{S}$ defines $(\{z_{(i,\mathcal{A})}\}_{i\in[n_1]}, \{z_{(i,\mathcal{B})}\}_{i\in[n_1]}) = (\{w^{\mathcal{A}}_i \oplus w^{\mathcal{B}}_i\}_{i\in\mathcal{I}^{\mathsf{out}}_{\mathcal{A}}}, \{w^{\mathcal{A}}_i \oplus w^{\mathcal{B}}_i\}_{i\in\mathcal{I}^{\mathsf{out}}_{\mathcal{B}}})$. The hybrids are identically distributed.

Hybrid-6: In Step 4)-(iv)-(b), $\mathcal{S}$ decides for inconsistent cheating if $\mathcal{M}^2_{(\mathcal{A},\mathcal{A})} \neq \mathcal{M}^2_{(\mathcal{B},\mathcal{A})}$, where $\mathcal{M}^2_{(\mathcal{B},\mathcal{A})}$ is computed as an honest party $\mathcal{B}$ would compute it. The hybrids are computationally indistinguishable due to the collision resistance of the random oracle.

Hybrid-7: In Step 4)-(iv)-(c), $\mathcal{S}$ sends the real output shares instead of the forged ones, i.e., $\mathcal{S}$ sends $(\mathsf{Opened}, \mathcal{B}, (\{w_i^{\mathcal{B}}\}_{i \in \mathcal{I}_{\mathcal{A}}^{\mathsf{out}}}, H(\{M[w_i^{\mathcal{B}}]\}_{i \in \mathcal{I}_{\mathcal{A}}^{\mathsf{out}}})))$. The hybrids are identically distributed.

Hybrid-8: In Step 4)-(iv)-(d), $\mathcal{S}$ does not instruct the honest party to output $(\{z_{(i,\mathcal{B}) \oplus u_i}\}_{i \in [n_1]})$ but $(\{\tilde{w}_i^{\mathcal{A}} \oplus w_i^{\mathcal{B}}\}_{i \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}})$. The hybrids are identically distributed. Further, we do some syntactic changes: $\mathcal{S}$ no longer computes u_*, $z_{(*,*)}$ and $\widetilde{\Delta}$, and the consistency check is moved from Step 4)-(iv)-(b) to Step 4)-(iv)-(d).

Hybrid-9: In Step 4)-(v), $\mathcal{S}$ defines $\mathcal{A}$'s resp $\mathcal{B}$'s supposed output to be $\{z_{(i,\mathcal{A})} := w_{\mathcal{I}_{\mathcal{A}}^{\mathsf{out}}[i]}\}_{i \in [n_2]}$ resp. $\{z_{(i,\mathcal{B})} := w_{\mathcal{I}_{\mathcal{B}}^{\mathsf{out}}[i]}\}_{i \in [n_2]}$. Instead of recomputing those values via $f(\cdot,\cdot)$ (within or outside of $\mathcal{F}_{\mathsf{Cov}}$) below Step 4)-(v), $\mathcal{S}$ uses the predefined values. The hybrids are identically distributed.

Hybrid-10: In Step 4)-(v)-(f), $\mathcal{S}$ sends the real output shares instead of the forged ones, i.e., $\mathcal{S}$ sends $(\mathsf{Opened}, \mathcal{B}, (\{w_i^{\mathcal{B}}\}_{i \in \mathcal{I}_{\mathcal{A}}^{\mathsf{out}}}, H(\{M[w_i^{\mathcal{B}}]\}_{i \in \mathcal{I}_{\mathcal{A}}^{\mathsf{out}}})))$. The hybrids are identically distributed.

Hybrid-11: In Step (4)-(v)-(f), $\mathcal{S}$ instructs the honest party to output $\{\tilde{w}_i^{\mathcal{A}} \oplus w_i^{\mathcal{B}}\}_{i \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}}$ if $\mathsf{tmp} \in \{1,4\}$. The hybrids are identically distributed.

Hybrid-12: In Step 4)-(v)-(e), $\mathcal{S}$ (impersonating $\mathcal{F}_{\mathsf{Cov}}$) decides on successful cheating if $\nexists i \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}$ such that $\widetilde{M}[w_i^{\mathcal{A}}] \neq K[w_i^{\mathcal{A}}] \oplus \tilde{w}_i^{\mathcal{A}} \cdot \Delta_{\mathcal{B}}$. The hybrids are identically distributed.

Hybrid-13: In Step 4)-(v), we merge the cases of detected cheating and blatant cheating as well as the cases of successful cheating and honest behavior. In particular, we remove Steps 4)-(v)-(c), 4)-(v)-(d) and 4)-(v)-(e), and re-define Step 4)-(v)-(f) as follows. $\mathcal{S}$ sends $(\mathsf{Opened}, \mathcal{B}, \cdot)$ to Adv. If Adv aborts, $\mathcal{S}$ instructs the honest party to output $\mathsf{corrupted}_{\mathcal{A}}$. Otherwise, $\mathcal{S}$ checks correctness of the MACs ($\exists i \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}$ such that $\widetilde{M}[w_i^{\mathcal{A}}] \neq K[w_i^{\mathcal{A}}] \oplus \tilde{w}_i^{\mathcal{A}} \cdot \Delta_{\mathcal{B}}$). If there is such an i, $\mathcal{S}$ instructs the honest party to output $\mathsf{corrupted}_{\mathcal{A}}$. Otherwise, $\mathcal{S}$ instruct the corrupted party to output $\{\tilde{w}_i^{\mathcal{A}} \oplus w_i^{\mathcal{B}}\}_{i \in \mathcal{I}_{\mathcal{B}}^{\mathsf{out}}}$.

The hybrids are identically distributed.

Hybrid-14: In this experiment, in Step 4)-(v), $\mathcal{S}$ decides for detected cheating if $\mathcal{M}^2_{(\mathcal{A},\mathcal{A})} \neq \mathcal{M}^2_{(\mathcal{B},\mathcal{A})}$, where $\mathcal{M}^2_{(\mathcal{B},\mathcal{A})}$ is computed as an honest party would compute it. The hybrids are computationally indistinguishable due to the collision resistance of the random oracle. Further, we do some syntactic changes: $\mathcal{S}$ no longer defines tmp or computes u_*.

Hybrid-15: We remove all failure events. In particular, we no longer backtrack preimages computed with H and remove the check for failure events. The hybrids are computationally indistinguishable due to the collision resistance of the random oracle.

Hybrid-16: We merge the cases of consistent cheating (Step 4)-(iv)) and tentatively honest behavior (Step 4)-(v)). In particular, $\mathcal{S}$ executes Step 4)-(iv) if there has not been an abort and $\mathcal{M}^1_{(\mathcal{B},\mathcal{A})} = \mathcal{M}^1_{(\mathcal{A},\mathcal{A})}$. The hybrids are distributed identically. As set $\mathcal{E}$ is no longer used, we no longer initialize or populate it.

Hybrid-17: $\mathcal{S}$ no longer tracks the computation of Adv, i.e., $\mathcal{S}$ does not store Adv's output of the precomputation, $\mathcal{S}$ does not extract Adv's actually input $\{\tilde{x}_{(i,\mathcal{A})}\}_{i\in[n_1]}$, and $\mathcal{S}$ does not monitor the circuit evaluation. The hybrids are identically distributed.

The final hybrid experiment, *Hybrid-17*:, equals the real world protocol execution, as $\mathcal{S}$ executes the honest party as specified by the protocol and the ideal functionalities as they are defined. As there is a constant number of hybrids and each two subsequent hybrids are at least computationally indistinguishable, it follows that the real-world execution is computationally indistinguishable from the ideal world execution which concludes the proof. □