Large Scale, Actively Secure Computation from LPN and Free-XOR Garbled Circuits

Aner Ben-Efraim[2], Kelong Cong[1], Eran Omri[2], Emmanuela Orsini[1], Nigel P. Smart[1,3], and Eduardo Soria-Vazquez[4]

[1] imec-COSIC, KU Leuven, Leuven, Belgium.
[2] Dept. Computer Science, Ariel Univeristy, Israel.
[3] Dept. Computer Science, University of Bristol, Bristol, UK.
[4] Dept. Computer Science, Aarhus University, Aarhus, Denmark. `anermosh@post.bgu.ac.il`, `kelong.cong@esat.kuleuven.be`, `omrier@gmail.com`, `emmanuela.orsini@kuleuven.be`, `nigel.smart@kuleuven.be`, `eduardo@cs.au.dk`

Abstract. We present a secure multiparty computation (MPC) protocol based on garbled circuits which is both actively secure and supports the free-XOR technique, and which has communication complexity $O(n)$ per party. This improves on a protocol of Ben-Efraim, Lindell and Omri which only achieved passive security, without support for free-XOR. Our construction is based on a new variant of LPN-based encryption, but has the drawback of requiring a rather expensive garbling phase. To address this issue we present a second protocol that assumes at least n/c of the parties are honest (for an arbitrary fixed value c). This second protocol allows for a significantly lighter preprocessing, at the cost of a small sacrifice in online efficiency. We demonstrate the practicality of our evaluation phase with an implementation.

Table of Contents

1 Introduction

The last decade has seen an enormous amount of progress in the practicality of actively secure multiparty computation (MPC), spanning many new designs and implementations of protocols based on both garbled circuits and secret sharing. Much of the developments have been in the dishonest majority case, where more than half of the parties can arbitrarily deviate from the protocol, trying to compromise privacy and correctness of computation. Despite this, there is still some gap between the complexities one can achieve in theory, and those which can be met by practical protocols in the real world.

Almost all of the most efficient protocols in the dishonest majority setting are designed in the so-called *preprocessing* model, in which parties first produce some input-independent correlated randomness which can be later used to evaluate the function. In secret-sharing-based protocols, the main goal of the preprocessing (or *offline*) phase is to generate secret-shared random multiplication triples, which are consumed during the online computation to evaluate multiplication gates. In garbled-circuit-based protocols, the preprocessing generates a one-time garbled circuit which will be later evaluated on private inputs.

Recent protocols in both of the above paradigms have incredibly fast execution times in their online phases when the number of parties n is relatively small (say less than 10), see for example SPDZ-like protocols [DPSZ12, LOS14, KOS16, KPR18] and SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18, OSV20], for the case of linear secret-sharing based MPC, and BMR-based protocols [HSS17, WRK17b, YWZ19]. However, when we increase the number of parties this practicality drops off.

Secret-sharing based protocols [GMW87, RB89, BGW88, DN07, DPSZ12], which work for both binary and arithmetic circuits, require a small amount of communication between (essentially) all parties for each layer of multiplication gates in the circuit, and hence their round complexity is linear in the depth of the circuit. This means that these protocols require very low bandwidth, and can be very efficient in a LAN (Local-Area-Networks) setting, but the large amount of rounds of communication and high latency make them less suited for the WAN (Wide-Area-Networks) setting, where the parties are usually geographically far apart from each other. If we consider the complexity of the online evaluation, secret-sharing based protocols have $O(n)$ complexity per gate per party[5].

Garbled circuit protocols, introduced by Yao [Yao86] in the two-party setting and later generalized to the multiparty case by Beaver, Micali and Rogaway (BMR) [BMR90], mainly work over binary circuits. In these protocols an "encrypted" version of the circuit is constructed in such a way that its evaluation does not require any communication beyond parties providing their "garbled" inputs. These protocols run in a constant number of rounds and are often slower than secret-sharing based protocols in a LAN setting due to their higher bandwidth requirements. Nevertheless, they are usually much faster in the WAN setting. For practical multiparty garbled-circuit protocols each evaluating party has to perform $O(n^2)$ operations. Thus the scalability of the online phase of secure multiparty computation protocols in a WAN setting, as the number of parties increases, is still an issue.

Theoretically, this is not a problem for multi-party garbled circuits. To achieve a protocol which has complexity $O(n)$ per party, one can take the standard two-party protocol by Yao [Yao86] and then compute the garbling function via an n-party actively secure MPC system. The resulting

[5] The complexity can be reduced to $O(1)$ for all but one of the parties in SPDZ-like protocols by 'opening' being performed in a king-followers fashion: Followers send their shares to the king, who then replies to all followers with the reconstructed value (hence $O(n)$ complexity for the king). For more details, see e.g. [DPSZ12].

garbled circuit will not depend on the number of parties, but the garbling itself will be highly inefficient as the underlying pseudo-random functions (PRFs) used in Yao's construction will need to be evaluated within MPC. Thus, while theoretically interesting, such an approach is unlikely to ever be practical.

The $O(n^2)$ complexity problem for practical BMR-based protocols led Ben-Efraim, Lindell and Omri [BLO17] to present a *passively secure* BMR-based protocol whose evaluation is independent of the number of parties and such that the garbling phase avoids to evaluate PRFs using generic MPC. This was done by utilizing a specific key-homomorphic PRF, for which two instantiations were given in the paper, one based on DDH in prime order groups and one based on Learning-with-Errors. The work of Ben-Efraim et al. provides a large-scale MPC protocol which is *almost practical*: their evaluation phase is concretely faster than previous works for large n, but more research is needed into the offline phase in order to make it practical. The efficiency of online evaluation is demonstrated through an implementation which shows that, roughly, their protocol is more efficient than its $O(n^2)$ counterpart [BLO16] as soon as 100 parties take part in the MPC. However, this large-scale protocol suffers from two major drawbacks: firstly, it only deals with the case of passive adversaries, and secondly their techniques are not compatible with the important free-XOR optimization introduced by Kolesnikov and Schneider [KS08].

Another relevant large-scale, garbled-circuit based protocol is that proposed by Hazay, Orsini, Scholl and Soria-Vazquez [HOSS18b]. Their result, which only deals with passive adversaries, shortens symmetric keys (as the ones for PRFs in the garbled circuit) in order to speed up computation and reduce communication. Security is then retained by relying on the length of the *concatenation* of all honest parties' keys, rather than on each of them individually. Such a protocol allows to evaluate each garble gate with $O(n^2\ell/\kappa)$ operations, compared to $O(n^2)$ of standard approaches, where $\kappa > \ell$ is the security parameter and ℓ is the key length. In subsequent work [HOSS18a], the same authors extended their technique to the active setting, but only for secret-sharing based protocols, leaving actively secure garbled circuits with short keys as an open problem.

1.1 Our Contribution

In this paper we introduce a new n-party garbling technique and present two almost-practical, large-scale BMR-style protocols. Both the size and evaluation complexity of the resulting garbled circuits is $O(1)$, hence resulting in an online phase which has a complexity of $O(n)$ per party[6]. Our protocols are actively secure and employ the free-XOR optimization by Kolesnikov and Schneider [KS08].

Obtaining Free-XOR. Our construction takes inspiration from the work of Ben-Efraim et al. [BLO17], but instead of basing the construction on key-homomorphic PRFs, we use an encryption scheme which is both key-homomorphic and message-homomorphic. In order to enable the free-XOR technique, we further need to restrict ourselves to message and key spaces of characteristic two. This rules out standard Ring-Learning-with-Errors (RLWE) based encryption schemes, for which the secret key and message spaces are modulo distinct primes. Instead, we introduce a new homomorphic encryption scheme based on the Learning-Parity-with-Noise (LPN) problem. We note that LPN-based encryption was also used by Appelbaum [App13] in order to replace the random oracle with standard cryptographic assumptions in two-party, free-XOR garbled circuits.

[6] This increase in complexity is due to parties still needing to reconstruct the circuit and send their masked inputs around.

We would like to stress that the motivation (and also the resulting LPN construction) for our work is different, as we aim to build practical protocols for a large number of parties rather than a purely theoretical result related to cryptographic assumptions. A further overview of our new LPN garbling scheme can be found in the next subsection, and all its details appear in Section 3.

Obtaining Active Security. Our first protocol achieves active security by employing an actively-secure garbling phase which guarantees that the resulting secret-shared garbled circuit is correct. While in standard BMR all of the garbling, except the PRFs evaluations, is computed within an MPC protocol, we instead *entirely* generate the garbled gates in a distributed manner using an actively secure full-threshold MPC system. We will refer to this first protocol as "*authenticated garbling*". This terminology resembles the authenticated-garbling technique by Wang, Ranellucci and Katz [WRK17a, WRK17b] (referred as WRK in the rest of the paper) and more recently by Yang, Wang and Zhang [YWZ19]. However, while their preprocessing phase is explicitly based on TinyOT-like protocols [NNOB12, FKOS15], which rely on Message Authentication Codes (MACs), our preprocessing works with any actively secure protocols.

In our construction each garbled AND gate consists of 4 rather than $4n$ ciphertexts as in previous BMR-style protocols. In the online phase, parties only need to broadcast shares of their inputs and perform a cheap, local computation that requires a single decryption per AND gate. However, this very efficient online evaluation comes at the price of a rather expensive preprocessing. Thus, whilst forming a potential bridge from what is theoretically possible to what is practically realisable, this protocol is only 'almost'-practical.

Bridging the Gap. To further bridge the gap between theory and practice, we also present a second construction with a more efficient preprocessing phase. We achieve this by relaxing some of the requirements in our garbling functionality, which becomes more similar to that described by Hazay, Scholl and Soria-Vazquez (HSS) [HSS17]. In particular, we allow the shares of the garbled circuit to be *unauthenticated*: rather than producing LPN ciphertexts within an actively secure MPC engine, each party will locally produce *additive shares* of these ciphertexts. This effectively allows the adversary to introduce arbitrarily additive errors to corrupted parties' shares. To maintain active security, we need to introduce an extra check in the online evaluation, as we explain in Technical Overview (Section 1.2).

In order to achieve a better performance, this new construction assumes that there are at least n/c honest parties, for an arbitrarily chosen constant $1 < c \leq n$. Since our goal is constructing efficient protocols for a large number of parties (typically more than one hundred), it is very reasonable to assume, in this setting, more than a single honest party.

Experimental Validation. We validate the claim that our protocol is almost-practical by demonstrating that the evaluation phase is indeed more efficient than other truly practical approaches when the number of parties is large. Thus, to turn our almost-practical protocol into a fully practical one, future works only need to concentrate on the garbling phase.

The concrete efficiency of our schemes crucially depends on the LPN parameters and the error correcting codes used to instantiate the two-key LPN based encryption scheme. We set the security of the scheme according to the work of Esser et al. [EKM17] and instantiate the cryptosystem with concatenated codes (see Section 6). We stress that our implementation should be taken more as a proof of feasibility than an optimized implementation of the proposed constructions. Moreover, we believe that using more efficient codes, like LDPC or QC-LDPC, the concrete efficiency of our protocols would improve significantly.

More concretely, in the full-threshold authenticated garbling case, experiments show that our evaluation phase will be more efficient than state of the art protocols such as HSS or WRK when the number of parties exceeds about 100. Notice HSS, WRK and the recent protocol of Yang et al. [YWZ19] have similar online efficiency, therefore, to concretely validate our claim, we compare the results of our experiments in the full-threshold case with the running times reported in [WRK17b]. Setting the statistical security parameter to 40, as in [WRK17b], we report a running time for AES-128 of 1.72 sec (c.f. Table 4 in Section 6), compared to 1.87 sec in a LAN setting and 2.3 sec in a WAN setting reported in WRK [WRK17b] for 128 parties. These numbers from WRK will grow quadratically as the number of parties increases, whereas ours will remain constant.

In the scalable protocol by Ben-Efraim et al. [BLO17] –only passively secure and without free-XOR– the authors also estimate that the cross over point from the $O(n^2)$ to the $O(n)$ protocols comes when n is about 100. Thus we obtain roughly the same cross over point in the case of active security with free-XOR as Ben-Efraim et al. do for passive security with no free-XOR. When comparing our protocol to [BLO17] we see that, assuming a circuit consisting solely of AND gates, our protocol is roughly six times slower than that of [BLO17]. Whilst this penalty for obtaining active security can be considered too much, one needs to consider the effect over typical circuits, as our protocols evaluate XOR gates for free. Thus, in practice, our performance penalty to achieve active security compared to Ben-Efraim et al. is closer to just a 15% of slow down. The details of our implementation can be found in Section 6.

1.2 Technical Overview

We now proceed to discuss our results and techniques in greater detail. They mainly revolve around two key ideas: how to use LPN encryption to allow n-party garbling with free-XOR, and how to achieve active security. We give an overview of these techniques below, more details can be found in the rest of the paper.

Since our constructions assume a circuit-based representation, we fix some conventions and notation we adopt across the paper. We consider binary circuits C_f consisting of $|C_\wedge|$ AND gates, $|C_\oplus|$ XOR gates, each of which has two input wires, u and v, and one output wire w. We use g to indicate the gate index. Let W be the set of all wires, W_{in} and W_{out} be the set of input and output wires, respectively, we assume $|W_{\mathsf{in}}| = n_{\mathsf{in}}$ and $|W_{\mathsf{out}}| = n_{\mathsf{out}}$. We denote by W_{in_i} the set of input wires associated to party P_i, and likewise for output wires W_{out_i}.

Background on BMR. Most of the work in multi-party garbled circuits is based on the BMR protocol by Beaver, Micali and Rogaway [BMR90], which has been recently improved by a sequence of works [BLO16, HSS17, LPSY15, LSS16, WRK17b] both in the case of passive and active security. In this paper we follow the approach described in [BLO16, HSS17].

These protocols consist of two phases: an input-independent preprocessing phase where the garbled circuit is generated, and an online phase where parties locally evaluate the circuit obtaining the output of the computation. While in Yao's two-party protocol only one party, the garbler, creates the garbled circuit, in BMR all parties generate it in a distributed way. This means that, instead of having a single key associated to each wire of the circuit, in multiparty garbling we have n keys for each wire, one for each party.

At the beginning of the preprocessing step, each party P_i chooses a global correlation $\varDelta^i \in \mathbb{F}_2^k$ to support free-XOR, and, for each wire w that is not the output wire of a XOR gate, samples a random key $\mathbf{k}^i_{w,0}$, associated to the value 0, and sets $\mathbf{k}^i_{w,1} = \mathbf{k}^i_{w,0} \oplus \varDelta^i$ for the value 1. Moreover, each P_i samples a random wire mask $\lambda^i_w \in \mathbb{F}_2$, for all the input wires $w \in W_{\mathsf{in}_i}$ and output wires of AND gates. Therefore the actual wire mask for such wires is given by $\lambda_w = \oplus_{i\in[n]}\lambda^i_w$.

In this way, XOR gates do not need any additional preprocessed material, as parties simply set $\mathbf{k}^i_{w,0} = \mathbf{k}^i_{u,0} \oplus \mathbf{k}^i_{v,0}$, $\mathbf{k}^i_{w,1} = \mathbf{k}^i_{w,0} \oplus \varDelta^i$ and $\lambda_w = \lambda_u \oplus \lambda_v$ (where u and v are the input wires and w is the output wire).

Let g be denote an AND gate with input wires u, v and output wire w. Given wire masks $\lambda_u, \lambda_v, \lambda_w$ and wire keys $\{\mathbf{k}^i_{u,\alpha}, \mathbf{k}^i_{v,\beta}, \mathbf{k}^i_{w,0}\}_{(\alpha,\beta)\in\{0,1\}^2, i\in[n]}$, parties generate a garbled gate corresponding to the AND truth table. It consists of four rows, indexed by the values $(\alpha, \beta) \in \{0,1\}^2$ on the input wires. Every row contains n ciphertexts, each of which is encrypted under $2n$ keys as follows:

$$\tilde{g}^j_{\alpha,\beta} = \left(\bigoplus_{i=1}^{n} F_{\mathbf{k}^i_{u,\alpha},\mathbf{k}^i_{v,\beta}}(g\|j)\right) \oplus \mathbf{k}^j_{w,0} \oplus \varDelta^j \cdot \big((\lambda_u \oplus \alpha)\cdot(\lambda_v \oplus \beta) \oplus \lambda_w\big), \quad (1)$$

where $j \in [n]$ represents the j-th ciphertext on the (α, β)-row and F is a double-key PRF. Note that, as free-XOR asks for every pair of keys $(\mathbf{k}^j_{w,0}, \mathbf{k}^j_{w,1})$ to be correlated according to $\varDelta^j$, we further need F to be a circular 2-correlation robust PRF [HSS17].

In the online phase, these encrypted truth tables, along with the input and output wire masks, are revealed to all parties so to allow local evaluation of the circuit. More precisely, in the input phase each party P_i broadcasts values $\epsilon_w = \rho_w \oplus \lambda_w$, for each $w \in W_{\mathsf{in}_i}$, where ρ_w is the actual input and λ_w the corresponding wire mask provided to P_i with other preprocessed material. In response, every party P_j broadcasts their key $\mathbf{k}^i_{w,\epsilon_w}$. Upon collecting all the keys and masked inputs, parties can start evaluating the circuit. At this point, this does not require any interaction. Given complete sets of input keys $(\mathbf{k}^1_{u,\epsilon_u}, \dots, \mathbf{k}^n_{u,\epsilon_u})$ and $(\mathbf{k}^1_{v,\epsilon_v}, \dots, \mathbf{k}^n_{v,\epsilon_v})$, it is possible to decrypt a single row of AND garbled gates obtaining $(\mathbf{k}^1_{w,\epsilon_w}, \dots, \mathbf{k}^n_{w,\epsilon_w})$. Note that during evaluation each party decrypts the entire row, requiring n^2 PRF evaluations. Once these output keys are obtained, every party P_i can check that the i-th key corresponds to one of its keys $\mathbf{k}^i_{w,0}, \mathbf{k}^i_{w,1}$ generated in the garbling phase. This check allows: 1) To determine the masked output value, i.e. if $\mathbf{k}^i_{w,\epsilon_w} = \mathbf{k}^i_{w,0}$, P_i sets $\epsilon_w = 0$, and $\epsilon_w = 1$ otherwise; 2) To ensure active security for the online evaluation.

Notice that, while [LPSY15] uses the actively secure SPDZ protocol [DPSZ12] to create an *authenticated* secret-sharing of Equation (1), Hazay et al. [HSS17] show that, in order to obtain an actively secure BMR-style protocol, it is enough to generate an *unauthenticated* additive sharing of the garbled circuit, provided that the values $\varDelta^j \cdot \big((\lambda_u \oplus \alpha)\cdot(\lambda_v \oplus \beta) \oplus \lambda_w\big)$ in Equation (1) are correctly generated.

BMR Garbling with LPN Encryption. We replace the circular 2-correlation robust PRF needed to allow the free-XOR technique in garbled circuit based protocols with a two-key symmetric encryption scheme based on LPN. By applying the key and message homomorphism, each garbled gate contains only a single ciphertext per row instead of n. However to achieve efficiency we need to modify the LPN encryption used in [App13], as we have n rather than two parties, and prove that our system still satisfies the Linear Related-Key and Key-Dependent-Message (LIN-RK-KDM) security needed to support the free-XOR optimization.

On the other hand, we cannnot naively modify the standard single-key LPN-based encryption scheme because of the free-XOR technique. Due to the key-homomorphism of LPN, there would be only two different keys –either $\mathbf{k}_{u,0} + \mathbf{k}_{v,0}$ or $\mathbf{k}_{u,0} + \mathbf{k}_{v,0} + \Delta$– encrypting each four-ciphertext gate entries in every garbled table (more details are in Section 3), essentially allowing the adversary to always decrypt half of them. We define a new scheme that still takes as input two keys but applies a permutation σ to the second one. We prove that the newly defined scheme satisfies a related notion of LIN-RK-KDM security, which we denote by LIN-RK-KDM$^\sigma$, while supporting the use of free-XOR in our garbled circuits.

Using our new scheme, we can replace the $4 \cdot n$ ciphertexts given in Equation (1) with 4 ciphertexts of the form

$$\tilde{g}_{\alpha,\beta} = \mathsf{Enc}\left((\mathbf{k}_{w,\epsilon_{w,\alpha,\beta}}, \epsilon_{w,\alpha,\beta}), (g\|\alpha\|\beta), (\mathbf{k}_{u,\alpha}, \mathbf{k}_{v,\beta})\right), \ (\alpha, \beta) \in \{0,1\}^2, \tag{2}$$

where the values $\epsilon_{w,\alpha,\beta} = (\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \oplus \lambda_w$, $\mathbf{k}_{w,\epsilon_{w,\alpha,\beta}} = \mathbf{k}_{w,0} \oplus \Delta \cdot \epsilon_{w,\alpha,\beta}$ correpond to the output public-value and output key, respectively.

Obtaining Active Security. We use the garbling technique just described to design our actively secure BMR protocols with linear online complexity in the number of parties. At a very high level the approach we follow to obtain active security is the same approach used in HSS, but with some significant differences.

The first one is clearly in the evaluation phase. In HSS, upon receiving all the input-wire keys and reconstructing the garbled circuit, parties evaluate the circuit locally by computing, for every AND gate, n^2 PRF evaluations. By subtracting those PRF outputs (see Equation 1), they obtain the n keys $(\mathbf{k}^1_{w,\epsilon_w}, \ldots, \mathbf{k}^n_{w,\epsilon_w})$ corresponding to the AND gate's output, which can be used to evaluate subsequent gates. Since, during this operation, each party P_i should recover one of its two possible output keys, $(\mathbf{k}^i_{w,0}, \mathbf{k}^i_{w,1})$, checking whether this condition verifies is enough to guarantee active security for the online evaluation. In our case this is no longer true, because upon decryption any party obtains a single unknown output key, $\mathbf{k}_{w,\epsilon_w}$. For security reasons, such a key needs to remain unknown to all parties up to this step, therefore, if we just plug-in our new garbling into HSS, it is no longer possible to check that the keys obtained by evaluating AND gates are correct. We describe two different ways to overcome this issue.

The first method, described in Section 4 and corresponding to the fully authenticated LPN-based garbling, proposes to fully authenticate the entire garbled circuit, and not just the wire mask. This is achieved using any MPC protocol with active security and dishonest majority. In this way the garbled values opened during the circuit evaluation are guaranteed to be correct, leading to a very efficient online phase. However, this comes at the price of a rather expensive preprocessing.

In our second protocol, described in Section 5, we improve the practicality of the preprocessing phase while maintaining almost the same online efficiency. In order to do so, we increase the number of honest parties to n/c, with $c \in \mathbb{R}$ and $1 < c \leq n$. The proposed protocol works for any $1 < c \leq n$: when $c \geq 2$ we are in the dishonest majority setting and when $c = n$ we go back to the full threshold case.

By setting the LPN parameters in the right way, we can design a protocol where each party locally generates "weak" (in term of security) ciphertexts. Since an adversary will be able to see only the sum of these ciphertexts, we show that this is enough to obtain a secure protocol. The balance then has to be drawn to ensure that enough 'noise' is added by each party in creating their own LPN-based ciphertexts in order to ensure privacy, but not too much to still guarantee correctness. The garbling we use in this case is unauthenticated, like in HSS, with only few actively secure MPC operations. Since, as explained before, we cannot rely on the online check used in HSS, we need to introduce a new additional test. In a little more detail, for each output gate g, with input wire u and output wire w, we construct a new garbled gate as

$$\tilde{g}_\alpha = \mathsf{Enc}\left((\xi^1_{w,\alpha}\| \dots \|\xi^n_{w,\alpha}), (g\|\alpha\|0), (\mathbf{k}_{u,\alpha}, \mathbf{0})\right), \ \alpha \in \{0,1\},$$

where each value $\xi^i_{w,\alpha}$ is generated by party P_i and then secret-shared among all parties. In the online phase each P_i decrypts $\tilde{g}_{\epsilon_u}$, where ϵ_u is the public value of g's input wire, and checks if the i-entry in the obtained vector correspond to one of the two values $\xi^i_{w,0}, \xi^i_{w,1}$. This extra check per output gate is sufficient to guarantee active security of our second protocol.

2 Preliminaries

We denote by sec the security parameter. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if, for every positive polynomial $p(\cdot)$ and all sufficiently large sec, it holds that $\mu(\mathsf{sec}) < \frac{1}{p(\mathsf{sec})}$. We assume that all involved algorithms are probabilistic polynomial time Turing machines. We let $x \leftarrow X$ denote the uniformly random assignment to the variable x from the set X, assuming a uniform distribution over X. We also write $x \leftarrow y$ as shorthand for $x \leftarrow \{y\}$. If $\mathcal{D}$ is a probability distribution over a set X, then we let $x \leftarrow \mathcal{D}$ denote sampling from X with respect to the distribution $\mathcal{D}$. If A is a (probabilistic) algorithm then we denote by $a \leftarrow A$ the assignment of the output of A where the probability distribution is over the random tape of A. With Ber_τ we denote the Bernoulli distribution of parameter τ, i.e. $\Pr[x = 1 : x \leftarrow \mathsf{Ber}_\tau] = \tau$.

Security Model. The protocols presented in this work are proved secure in the Universal Composability framework of Canetti [Can01]. We consider security against a static, malicious adversary who corrupts a subset $I \subset \mathcal{P} = \{P_1, \dots, P_n\}$ of parties at the beginning of the protocol.

We assume all parties are connected via authenticated channels as well as secure point-to-point channels and a broadcast channel. The default method of communication is through authenticated channels, unless otherwise specified.

Randomized Functions: To describe our garbling technique we follow the same approach used in [App13] and use the terminology of randomized encodings for garbled circuits [IK00, IK02].

A *randomized function* $f : X \times R \longrightarrow Y$ is a two argument function such that, for every input $x \in X$, we can think of $f(x)$ as a random variable which samples $r \in R$ and then applies $f(x; r)$. When an algorithm A gets oracle access to a randomized function f we assume A only has control on the inputs x. We denote the resulting randomized function by A^f. We say that two randomized functions are *equivalent*, written $f \equiv g$, if for every input, their output is identically distributed.

A set of randomized functions $\{f_\mathbf{s}\}_{\mathbf{s}\in\{0,1\}^*}$, indexed by a key $\mathbf{s}$, is called a *collection of randomized functions* if $f_\mathbf{s}$ is a randomized function for every $\mathbf{s}$. In the following we drop the dependency on $\mathbf{s}$.

We say that two collections $\{f_\mathbf{s}\}$ and $\{g_\mathbf{s}\}$ of randomized functions are computationally indistinguishable, written $\{f_\mathbf{s}\} \overset{c}{\equiv} \{g_\mathbf{s}\}$, if the probability that an efficient adversary can distinguish between them, given oracle access to a function in $\{f_\mathbf{s}\}$ and a function in $\{g_\mathbf{s}\}$, is negligible.

Let $\{f_\mathbf{s}\}, \{g_\mathbf{s}\}, \{h_\mathbf{s}\}$ be collections of randomized functions, we have the following standard facts [Mau02]:

– if $\{f_{\mathbf{s}}\} \overset{c}{\equiv} \{g_{\mathbf{s}}\}$ and A is an efficient function then $\{A^{f_{\mathbf{s}}}\} \overset{c}{\equiv} \{A^{g_{\mathbf{s}}}\}$;
– if $\{f_{\mathbf{s}}\} \overset{c}{\equiv} \{g_{\mathbf{s}}\}$ and $\{g_{\mathbf{s}}\} \overset{c}{\equiv} \{h_{\mathbf{s}}\}$ then $\{f_{\mathbf{s}}\} \overset{c}{\equiv} \{h_{\mathbf{s}}\}$.

2.1 LIN-RK-KDM Security

We briefly recall the notion of (Linear) Related-Key and Key-Dependent-Message security [App13, AHI11, BK03, BRS03, CL01] that we need in our constructions: Given a symmetric encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ over the plaintext space $\mathcal{M} = \mathbb{F}_2^*$ and key space $\mathcal{K} = \mathbb{F}_2^{\mathsf{sec}}$, we define two families of key-derivation and key-dependent message functions:

$$\Phi_{\mathsf{RKA}} = \{\phi : \mathcal{K} \to \mathcal{K}\} \quad \text{and} \quad \Psi_{\mathsf{KDM}} = \{\psi : \mathcal{K} \to \mathcal{M}\},$$

such that Related-Key and Key-Dependent-Message (RK-KDM) security can be defined through two oracles $\mathsf{Real}_{\mathbf{s}}$ and $\mathsf{Fake}_{\mathbf{s}}$, indexed by a key $\mathbf{s} \in \mathcal{K}$, as follows: for each query $(\phi, \psi) \in \Phi_{\mathsf{RKA}} \times \Psi_{\mathsf{KDM}}$, Real_s returns a sample from the distribution $\mathsf{Enc}(\psi(\mathbf{s}); \phi(\mathbf{s}))$ and $\mathsf{Fake}_{\mathbf{s}}$ a sample from the distribution $\mathsf{Enc}(0^{|\psi(\mathbf{s})|}; \phi(\mathbf{s}))$.

Definition 1 (RK-KDM secure encryption, [App13]). *We say that a symmetric encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ is semantically-secure under* RK-KDM *attacks with respect to Φ_{RKA} and Ψ_{KDM} if $\mathsf{Real}_{\mathbf{s}} \overset{c}{\equiv} \mathsf{Fake}_{\mathbf{s}}$, where $\mathbf{s} \leftarrow \mathcal{K}$.*

If both ϕ and ψ are linear functions over $\mathbb{F}_2$, we refer to this notion as Linear Key-Related and Key-Dependent-Message (LIN-RK-KDM) security. In this case we can rewrite the oracles in a compact way:

$$\begin{aligned} \mathsf{Real}_{\mathbf{s}} : (\delta, \mathbf{m}, b) &\longmapsto \mathsf{Enc}(\ \mathbf{m} \oplus b \cdot \mathbf{s},\ \ \delta \oplus \mathbf{s}\ ) \\ \mathsf{Fake}_{\mathbf{s}} : (\delta, \mathbf{m}, b) &\longmapsto \mathsf{Enc}(\ 0^{|\mathbf{m}|},\ \ \delta \oplus \mathbf{s}\ ), \end{aligned}$$

where $\mathbf{m} \in \mathcal{M}$ is a message, $\mathbf{s} \in \mathcal{K}$ a key, $b \in \mathbb{F}_2$ a bit and $\delta \in \mathbb{F}_2^{\mathsf{sec}}$ a key-shift. Notice in computing $\mathbf{m} \oplus b \cdot \mathbf{s}$ we multiply $\mathbf{s}$ by b bitwise, and then pad the result with $|\mathbf{m}| - k$ zeros to left before xor-ing with $\mathbf{m}$.

2.2 Error Correcting Codes

An $[\ell, \mathfrak{m}, d]$ binary linear code L is a subspace of dimension $\mathfrak{m}$ of $\mathbb{F}_2^{\ell}$, where ℓ is the length of the code, $\mathfrak{m}$ its dimension and d its distance, i.e. the minimum (Hamming) distance between any distinct codewords in L. We denote by G a *generator matrix* of L, that is any matrix in $\mathbb{F}_2^{\mathfrak{m} \times \ell}$ whose rows form a basis for L. If G has the form $[I_{\mathfrak{m}} | P]$, where $I_{\mathfrak{m}}$ is the $\mathfrak{m} \times \mathfrak{m}$ identity matrix, G is said to be in *standard form*. A *parity-check matrix* for L is a matrix in $\mathbb{F}_2^{(\ell - \mathfrak{m}) \times \ell}$ such that $GH^T = 0$. A linear code can be uniquely specified either by its generator matrix or its parity-check matrix.

Given an $[\ell, \mathfrak{m}, d]$ binary linear code L, we can define a pair of algorithms $(\mathsf{Encode}, \mathsf{Decode})$, where $\mathsf{Encode} \colon \mathbb{F}_2^{\mathfrak{m}} \to \mathbb{F}_2^{\ell}$ (resp. $\mathsf{Decode} \colon \mathbb{F}_2^{\ell} \to \mathbb{F}_2^{\mathfrak{m}}$) is an encoding (resp. decoding) algorithm, such that:

1. **Linearity:** For every pair of messages $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}_2^{\mathfrak{m}}$ we have $\mathsf{Encode}(\mathbf{x}_1) \oplus \mathsf{Encode}(\mathbf{x}_2) = \mathsf{Encode}(\mathbf{x}_1 \oplus \mathbf{x}_2)$.

2. $\lfloor(d-1)/2\rfloor$**-Correction:** The decoding algorithm can correct any error of Hamming weight up to $\lfloor(d-1)/2\rfloor$, i.e., for every message $\mathbf{x} \in \mathbb{F}_2^{\mathfrak{m}}$ and every error vector $\mathbf{e} \in \mathbb{F}_2^{\ell}$ with at most $\lfloor(d-1)/2\rfloor$ non-zero elements, it always holds that $\mathsf{Decode}(\mathsf{Encode}(\mathbf{x}) \oplus \mathbf{e}) = \mathsf{Decode}(\mathsf{Encode}(\mathbf{x})) = \mathbf{x}$.

We will also need the following more general property.

Definition 2 ((ℓ, τ)-Correction:). *Let* Ber_τ *be the Bernoulli distribution with parameter* τ. *Given an* $[\ell, \mathfrak{m}, d]$ *binary linear code* L *and a pair of* efficient *encoding and decoding algorithms,* $(\mathsf{Encode}, \mathsf{Decode})$, *we say that* L *is* (ℓ, τ)*-correcting if, for any message* $\mathbf{x} \in \mathbb{F}_2^{\mathfrak{m}}$, *the decoding algorithm* Decode *will, with overwhelming probability, satisfy* $\mathsf{Decode}(\mathsf{Encode}(\mathbf{x}) \oplus \mathbf{e}) = \mathsf{Decode}(\mathsf{Encode}(\mathbf{x})) = \mathbf{x}$, *where* $\mathbf{e} \leftarrow \mathsf{Ber}_\tau^\ell$ *is a noise vector, and* Ber_τ^ℓ *is the distribution over* $\mathbb{F}_2^\ell$ *obtained by drawing each entry of the vector* $\mathbf{e}$ *independently according to* Ber_τ.

2.3 LPN-based Encryption

The Learning Parity with Noise (LPN) problem [GKL90, BFKL94] is a well-studied problem in learning and coding theory, and has recently found many applications in cryptography. In this section we introduce the decisional version of the LPN problem together with some variants of the standard LPN-based encryption scheme that we need in our garbling construction.

Definition 3 (Decisional LPN). *Let* $\ell, k \in \mathbb{N}$ *and* $\tau \in (0, 1/2)$, *the* $\mathsf{DLPN}_{\ell,k,\tau}$ *problem is to distinguish between the distributions given by*

$$\left\{(C, \mathbf{c}) : C \leftarrow \mathbb{F}_2^{\ell \times k},\ \mathbf{s} \leftarrow \mathbb{F}_2^k,\ \mathbf{e} \leftarrow \mathsf{Ber}_\tau^\ell,\ \mathbf{c} \leftarrow C \cdot \mathbf{s} \ \oplus \ \mathbf{e}\right\}$$

and

$$\left\{(C, \mathbf{c}) : C \leftarrow \mathbb{F}_2^{\ell \times k},\ \mathbf{c} \leftarrow \mathbb{F}_2^\ell\right\}.$$

The decisional and search variants of the LPN problem are polynomially equivalent, they have been extensively studied and are widely believed to be hard for any τ. The DLPN assumption has been used to build various cryptographic primitives and, in particular, symmetric encryption schemes.

Definition 4 (Standard LPN Encryption). *Let* $\mathfrak{m}, k, \ell = \mathsf{poly}(\mathsf{sec})$ *be three integers. Let* $\mathcal{K} = \mathbb{F}_2^k$ *be the key space,* $\mathcal{C} = \mathbb{F}_2^{\ell \times k} \times \mathbb{F}_2^\ell$ *the ciphertext space and* $\mathcal{M} = \mathbb{F}_2^{\mathfrak{m}}$ *the message space. Let* $\tau \in (0, 1/2)$ *be a parameter defining the Bernoulli distribution* Ber_τ^ℓ. *Finally, let* $G \in \mathbb{F}_2^{\ell \times \mathfrak{m}}$ *be a generator matrix for an* $[\ell, \mathfrak{m}, d]$ *binary linear code* L *which is* (ℓ, τ)*-correcting. The (standard) LPN symmetric encryption scheme consists of the three following algorithms:*

- $\mathsf{KeyGen}_\tau^1(1^{\mathsf{sec}})$*: Given as input the security parameter* sec, *sample uniformly at random a secret key,* $\mathbf{s} \leftarrow \mathcal{K}$.
- $\mathsf{Enc}_\tau^1(\mathbf{m}, \mathbf{s})$*: Given a message* $\mathbf{m} \in \mathcal{M}$ *and the secret key* $\mathbf{s} \in \mathcal{K}$, *sample a matrix* $C \leftarrow \mathbb{F}_2^{\ell \times k}$, *noise* $\mathbf{e} \leftarrow \mathsf{Ber}_\tau^\ell$ *and output*
$$\mathbf{c} \leftarrow C \cdot \mathbf{s} \ \oplus \ \mathbf{e} \ \oplus \ G \cdot \mathbf{m}.$$
- $\mathsf{Dec}_\tau^1((C, \mathbf{c}), \mathbf{s})$*: Given a ciphertext* $(C, \mathbf{c})$ *and the secret key* $\mathbf{s}$, *compute* $\mathbf{c} \ \oplus \ C \cdot \mathbf{s}$ *and apply a decoding algorithm to recover* $\mathbf{m}$.

In [App13], Appelbaum proved that (an extension of) the above encryption scheme is LIN-RK-KDM secure.

Theorem 1. *Assuming* $\mathsf{DLPN}_{\ell,k,\tau}$ *is hard, the encryption scheme* $(\mathsf{KeyGen}^1_\tau, \mathsf{Enc}^1_\tau, \mathsf{Dec}^1_\tau)$ *is* LIN-RK-KDM secure *according to the above definition of LIN-RK-KDM security.*

Assuming the DLPN-problem is hard, it is easy to show that also the following nonce-based symmetric encryption scheme is IND-CPA, where it is required that a specific nonce is used only once for each key $\mathbf{s}$.

An eXtendable Output Function (XOF). A XOF is a way to model a random oracle that can produce outputs of any length. Implementations of such functions can be created from SHA-3 in a standardized manner [NIS16, BDP+18].

Definition 5 (XOF-Based LPN Encryption). *Let* $\mathfrak{m}, k, \ell = \mathsf{poly}(\mathsf{sec})$ *be three integers and* $\mathcal{K}, \mathcal{C}, \mathcal{M}$ *as in* Definition 4. *Let* $\tau \in (0, 1/2)$ *and* $G \in \mathbb{F}_2^{\ell\times\mathfrak{m}}$ *be chosen in the same way as there too. Let a XOF* $H : \{0,1\}^* \longrightarrow \mathbb{F}_2^{\ell\times k}$ *be modelled as a random oracle. The XOF-Based LPN symmetric encryption scheme consists of the three following algorithms:*

- $\mathsf{KeyGen}^{\mathsf{XOF}}_\tau(1^{\mathsf{sec}})$*: Sample uniformly at random a secret key,* $\mathbf{s} \leftarrow \mathcal{K}$.
- $\mathsf{Enc}^{\mathsf{XOF}}_\tau((\mathbf{m}, \mathsf{nonce}), \mathbf{s})$*: Given a message* $\mathbf{m} \in \mathcal{M}$*, a key* $\mathbf{s} \in \mathcal{K}$ *and a string* nonce*, sample noise* $\mathbf{e} \leftarrow \mathsf{Ber}^\ell_\tau$ *and compute*

$$C \leftarrow H(\mathsf{nonce}) \text{ and } \mathbf{c} \leftarrow C \cdot \mathbf{s} \oplus \mathbf{e} \oplus G \cdot \mathbf{m}.$$

- $\mathsf{Dec}^{\mathsf{XOF}}_\tau((C, \mathbf{c}), \mathbf{s})$*: Given a ciphertext* $(C, \mathbf{c})$*, compute* $\mathbf{c} \oplus C \cdot \mathbf{s}$ *and then apply error correction to recover* $\mathbf{m}$

The above LPN encryption scheme is trivially additively homomorphic in the message space, and is also key homomorphic if two encryptions with the same nonce value are added together. To reduce bandwidth and storage requirements, it is possible to define the ciphertext to be $(\mathsf{nonce}, \mathbf{c})$ instead of $(C, \mathbf{c})$.

Looking ahead, we will choose the parameters for our LPN-based encryption scheme based on recent analysis on the security of the LPN assumption by Esser et al. [EKM17], which implies that the parameter k in the scheme should be selected to be

$$k \geq \frac{\mathsf{sec}}{\log_2\left(\frac{1}{1-\tau}\right)}, \tag{3}$$

where sec is the (symmetric-key equivalent) security parameter and τ defines the noise rate. In what follows one should think of sec as being equal to 128 or 256.

2.4 Functionalities for Secret-shared MPC

Our protocols make use of the functionality $\mathcal{F}_{\mathsf{MPC}}$ for MPC over binary circuits described in Figure 1. The functionality is independent of how the values are stored and represented. In particular, we will need two different implementations of $\mathcal{F}_{\mathsf{MPC}}$, one achieving only passive security and the second achieving active security. Note that any generic MPC protocol can be used to practically

Functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{flag}}$

The functionality runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$.
It is parametrized by $\mathsf{flag} \in \{\mathsf{Auth}, \mathsf{UnAuth}\}$. Given a set ID of valid identifiers, all values are stored in the form $(varid, x)$, where $varid \in ID$.

Initialize: On input $(Init)$ from all parties. The adversary is assumed to have corrupted a subset I of the parties.

Input: On input $(\mathsf{Input}, P_i, varid, x)$ from P_i, with $x \in \mathbb{F}_2$, and $(\mathsf{Input}, P_i, varid, ?)$ from all other parties, with $varid$ a fresh identifier.

Add: On command $(\mathsf{Add}, varid_1, varid_2, varid_3)$ from all parties:

1. The functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \oplus y)$.

Multiply: On input $(\mathsf{Multiply}, varid_1, varid_2, varid_3)$ from all parties:

1. The functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y)$.

Output/Open: On input $(\mathsf{Output/Open}, varid, i)$ from all honest parties the functionality retrieves $(varid, y)$, sends y to the adversary, and waits for a reply. If $\mathcal{A}$ answers with $\mathsf{Deliver}$, then do one of the following:

- If $\mathsf{flag} = \mathsf{Auth}$: output y to either all parties (if $i = 0$) or P_i (if $i \neq 0$).
- If $\mathsf{flag} = \mathsf{UnAuth}$: $\mathcal{A}$ further specifies an additive error $e \in \mathbb{F}_2$. The functionality outputs $y + e$ to either all parties (if $i = 0$) or P_i (if $i \neq 0$).

In both cases, if $\mathcal{A}$ does not answer with $\mathsf{Deliver}$, output abort.

Figure 1. The ideal functionality for MPC over $\mathbb{F}_2$

instantiate $\mathcal{F}_{\mathsf{MPC}}$ in our constructions. However, since TinyOT-like protocols, that rely on message authentication codes (MACs) to achieve active security, are currently the most efficient protocols on binary circuits and are used in previous works like HSS and WRK, we abuse notation and use $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ and $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{UnAuth}}$ to distinguish between an active and a passive implementation of $\mathcal{F}_{\mathsf{MPC}}$. Also notice that each value in $\mathcal{F}_{\mathsf{MPC}}$ is uniquely identified by an identifier $varid \in ID$, where ID is a set of identifiers.

After an **Initialize** step, the functionality allows the parties to provide their inputs, which can be added and multiplied using **Add** and **Multiply**, respectively. The functionality also provides an **Output/Open** command that allows values to be revealed either publicly or privately to a single party. Note we maintain the double notation **Output/Open** only to distinguish between output values and intermediate values that are opened during the execution of the protocol.

Unauthenticated values: We denote $\langle x \rangle$ an additive sharing of x over $\mathbb{F}_2$ generated by $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{UnAuth}}$, where $x = \oplus_{i \in [n]} x^i$ with party P_i holding the share $x^i \in \mathbb{F}_2$.
Looking ahead, using such a sharing we can perform arbitrary linear operations, however, upon opening values, an adversary is able to introduce an arbitrary additive error and reveal incorrect values. For this reason when we use unauthenticated values to instantiate our LPN-based protocol, we need to add an new mechanism to prevent these additive errors introducing a security weakness in the protocol.

Authenticated values: We denote $[x]$ an actively secure additive sharing of x, for example using a fixed MAC scheme. Addition and multiplication of such elements will be represented by $[x] + [y]$ and $[x] \cdot [y]$.

To simplify notation we will use the following shorthands for inputing and outputting values to/from a party/all parties:

$$[x] \leftarrow \mathsf{Input}(P_i), \qquad x \leftarrow \mathsf{Output}([x], P_i), \qquad x \leftarrow \mathsf{Open}([x]),$$

$$\langle x\rangle \leftarrow \mathsf{Input}(P_i), \qquad x \leftarrow \mathsf{Output}(\langle x\rangle, P_i), \qquad x \leftarrow \mathsf{Open}(\langle x\rangle),$$

respectively in $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ and $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{UnAuth}}$. If the type (authenticated/unauthenticated) of operation is not obvious from the context we will write $\mathsf{Input}^P, \mathsf{Output}^P, \mathsf{Open}^P$ for the unauthenticated variant, with no superscript added for the authenticated variant.

Trivially, from a $[x]$ sharing we can obtain (immediately and with no computation or communication) a $\langle x\rangle$ sharing of the same value. We denote this operation by $\langle x\rangle \leftarrow \mathsf{Convert}([x])$. Extension of this notation to act on elements $\mathbf{x} \in \mathbb{F}_2^k$, for various values of k, will be by using $[\mathbf{x}]$ and $\langle \mathbf{x}\rangle$ in the obvious way.

We can extend the $\mathcal{F}_{\mathsf{MPC}}$ functionality by a command, which we denote by $[x] \leftarrow \mathsf{GenBit}()$ which produces a shared random bit within the MPC engine. This command can be derived from the base commands by performing:

1. All parties call $[x^i] \leftarrow \mathsf{Input}(P_i)$, $x^i \in \mathbb{F}_2$.
2. Parties compute $[x] \leftarrow \oplus_i [x^i]$.

3 Free-XOR Garbling using LPN

We now discuss how to garble a single AND gate using LPN-based encryption while maintaining the free-XOR invariant. Later on, in Sections 4 and 5, we will show how this technique can be used in order to build our actively secure garbled-circuit based MPC protocols.

Our garbling method is similar to the one given in Equation (1), with two main differences. Firstly and most importantly, we have a single ciphertext per row, rather than n of them; secondly, we replace the circular 2-correlation robust PRF F with a nonce-based, two-key symmetric encryption scheme based on LPN. Thus we obtain the garbling method given in Equation (2).

To achieve this modification one could naively think of just adapting standard LPN encryption (c.f. Definition 4) to use two keys, where $\Delta = \bigoplus_{i=1}^n \Delta^i$, and, for $t \in \{u, v, w\}$, $\mathbf{k}_{t,0} = \bigoplus_{i=1}^n \mathbf{k}_{t,0}^i$ and $\mathbf{k}_{t,1} = \mathbf{k}_{t,0} \oplus \Delta$. Each garbled row $(\epsilon_u, \epsilon_v) \in \{0,1\}^2$ could then be set as:

$$\tilde{g}_{\epsilon_u,\epsilon_v} = (C, c), \quad C \leftarrow \mathbb{F}_2^{\ell\times k}, \quad c \leftarrow C \cdot (\mathbf{k}_{u,\epsilon_u} \oplus \mathbf{k}_{v,\epsilon_v}) \oplus \mathbf{e} \oplus G \cdot \mathbf{k}_{w,\epsilon_w} \tag{4}$$

This naive solution does not result in a secure garbling method. To see this denote $\mathbf{s}_{\epsilon_u,\epsilon_v} = \mathbf{k}_{u,\epsilon_u} \oplus \mathbf{k}_{v,\epsilon_v}$, then due to free-XOR we would have that $\mathbf{s}_{\epsilon_u,\epsilon_v} = \mathbf{k}_{u,0} \oplus \mathbf{k}_{v,0} \oplus (\epsilon_u \oplus \epsilon_v) \cdot \Delta$, and hence $s_{0,0} = s_{1,1}$ as well as $s_{1,0} = s_{0,1}$. This would trivially allow corrupted parties to always decrypt half of the entries of every garbled gate, breaking completely the security of the scheme. A possible fix to this problem would be to sample two different matrices $C_u, C_v \leftarrow \mathbb{F}_2^{\ell\times k}$ and compute $c \leftarrow C_u \cdot \mathbf{k}_{u,\epsilon_u} \oplus C_v \cdot \mathbf{k}_{v,\epsilon_v} \oplus \mathbf{e} \oplus G \cdot \mathbf{k}_{w,\epsilon_w}$, but this would incur in increased computational costs due to the sampling of the matrices and the cost of calculating the matrix-vector products.

In order to avoid these issues in our garbling, while still maintaining security, we introduce a modification to the previously provided nonce-based version of LPN encryption. In particular, our scheme will take as input two keys in $\mathbb{F}_2^k$, but this time a permutation $\sigma \in S_k$ (where S_k is the set of permutations on k elements) will be applied to the second one.

Definition 6 (XOF-Based Two-Key LPN Encryption). *Let $\mathfrak{m}, k, \ell = \mathsf{poly}(\mathsf{sec})$ be three integers. Let $\mathcal{K} = \mathbb{F}_2^k \times \mathbb{F}_2^k$ be the key space, $\mathcal{C} = \mathbb{F}_2^{\ell \times k} \times \mathbb{F}_2^\ell$ the ciphertext space and $\mathcal{M} = \mathbb{F}_2^{\mathfrak{m}}$ the message space. Let $\tau \in (0, 1/2)$ be a parameter defining a Bernoulli distribution and σ a permutation in S_k. Finally, let $G \in \mathbb{F}_2^{\ell \times \mathfrak{m}}$ be a generator matrix for an $[\ell, \mathfrak{m}, d]$ binary linear code L which is (ℓ, τ)-correcting (c.f. Definition 2). Let $H : \{0,1\}^* \longrightarrow \mathbb{F}_2^{\ell \times k}$ be a XOF. A XOF-based, two-key symmetric LPN encryption scheme $\mathcal{E}_\tau^{\mathsf{XOF}}$ is defined by the following algorithms:*

- $\mathsf{KeyGen}(1^{\mathsf{sec}})$*: Samples $(\mathbf{k}_u, \mathbf{k}_v) \leftarrow \mathbb{F}_2^{2 \times k}$ at random.*
- $\mathsf{Enc}_\tau((\mathbf{m}, \mathsf{nonce}), (\mathbf{k}_u, \mathbf{k}_v))$*: On input of a message $\mathbf{m} \in \mathcal{M}$, a pair of keys $(\mathbf{k}_u, \mathbf{k}_v)$ and a string* nonce*, compute*

$$\begin{aligned} &C \leftarrow H(\mathsf{nonce}), \\ &\mathbf{c} \leftarrow C \cdot (\ \mathbf{k}_u \ \oplus \ \sigma(\mathbf{k}_v)\ ) \ \oplus \ \mathbf{e} \ \oplus \ G \cdot \mathbf{m}, \ \mathbf{e} \leftarrow \mathsf{Ber}_\tau^\ell. \end{aligned}$$

- $\mathsf{Dec}((C, \mathbf{c}), (\mathbf{k}_u, \mathbf{k}_v))$*: Compute $\mathbf{c} \ \oplus \ C \cdot (\ \mathbf{k}_u \ \oplus \ \sigma(\mathbf{k}_v)\ )$ and then apply error correction to recover $\mathbf{m}$.*

Note that this scheme is message homomorphic, and it only requires to store nonce rather than C. In addition, when the same nonce is used, it is also key homomorphic.

Returning to our garbling proposal from the beginning of this section, now the key used to garble entry (ϵ_u, ϵ_v) of a given gate g is $\mathbf{s}_{\epsilon_u, \epsilon_v} = \mathbf{k}_{u,\epsilon_u} \oplus \sigma(\mathbf{k}_{v,\epsilon_v})$. By substituting the free-XOR correlation, we see that security now relies on the secrecy of

$$\mathbf{s}_{\epsilon_u, \epsilon_v} = \mathbf{k}_{u,0} \ \oplus \ \sigma(\mathbf{k}_{v,0}) \ \oplus \ \epsilon_u \cdot \varDelta \ \oplus \ \epsilon_v \cdot \sigma(\varDelta), \tag{5}$$

and hence on four possible (distinct) values of $\mathbf{s}_{\epsilon_u, \epsilon_v}$. Nevertheless, the security analysis requires additional care. As it is always the case when using the free-XOR optimization, we have the problem that we are encrypting key-dependent messages (where the dependence is the free-XOR correlation $\varDelta$), as well as we are using related keys when encrypting the inactive rows of a garbled gate. Explicitly, given the active row $\mathbf{s}_{\epsilon_u, \epsilon_v}$, for $(\alpha, \beta) \in \{0,1\}^2$ these inactive rows are:

$$\mathbf{s}_{\epsilon_u \oplus \alpha, \epsilon_v \oplus \beta} = \mathbf{s}_{\epsilon_u, \epsilon_v} \ \oplus \ \alpha \cdot \varDelta \ \oplus \ \beta \cdot \sigma(\varDelta).$$

Hence, once the parties learn any $\mathbf{s}_{\epsilon_u, \epsilon_v}$ by evaluating the garbled circuit, security for each of the three remaining rows is relying, respectively, on the secret values $\varDelta, \sigma(\varDelta)$ and $\varDelta \oplus \sigma(\varDelta)$. To define an appropriate way of dealing with this RK-KDM problem, we will first define the following variant of LPN.

Definition 7 (DLPN$^\sigma$ Problem). *Let $\sigma \in S_k$ be the set of permutations of k elements and $\ell, k, \tau \in \mathbb{N}$. The $\mathsf{DLPN}^\sigma_{\ell,k,\tau}$ problem is to distinguish between the two distributions given by*

$$\left\{(C, \mathbf{c}, \sigma) : C \leftarrow \mathbb{F}_2^{\ell \times k}, \ \mathbf{s} \leftarrow \mathbb{F}_2^k, \ \mathbf{e} \leftarrow \mathsf{Ber}_\tau^\ell, \ \mathbf{c} \leftarrow C \cdot (\mathbf{s} \ \oplus \ \sigma(\mathbf{s})) \ \oplus \ \mathbf{e}\right\}$$

and

$$\left\{(C, \mathbf{c}, \sigma) : C \leftarrow \mathbb{F}_2^{\ell \times k}, \ \mathbf{c} \leftarrow \mathbb{F}_2^\ell\right\},$$

where Ber_τ^ℓ is the Bernoulli distribution with parameter τ.

Recalling that any permutationon of a finite set can be uniquely expressed as the product of disjoint cycles, we now show how the DLPN and DLPN^{σ} problems are related to each other by the following Lemma.

Lemma 1. *Let $\sigma \in S_k$ be a permutation consisting of exactly $\tilde{k}$ disjoint cycles, the $\mathsf{DLPN}_{\ell,k-\tilde{k},\tau}$ problem reduces to $\mathsf{DLPN}^{\sigma}_{\ell,k,\tau}$ problem.*

Proof. Our goal is, given a challenge $(C, \mathbf{c})$ for the standard $\mathsf{DLPN}_{\ell,k-\tilde{k},\tau}$, to come up with an appropriate challenge $(\hat{C}, \hat{\mathbf{c}}, \sigma)$ for the $\mathsf{DLPN}^{\sigma}_{\ell,k,\tau}$ problem. Then, we can apply the distinguisher for the $\mathsf{DLPN}^{\sigma}_{\ell,k,\tau}$ problem on the resulting challenge. We then argue that this forms a good distinguisher for the $\mathsf{DLPN}_{\ell,k-\tilde{k},\tau}$ problem.

Given a challenge $(C, \mathbf{c})$, where $C \in \{0,1\}^{\ell \times k-\tilde{k}}$ and $\mathbf{c} \in \mathbb{F}_2^{\ell}$, construct a challenge $(\hat{C}, \hat{\mathbf{c}}, \sigma)$ (where σ remains the above fixed permutation) as follows. First, simply set $\hat{\mathbf{c}} = \mathbf{c}$. Second, to construct $\hat{C}$ from C, the basic idea is to add an extra column for every cycle in σ and adjust other columns that are attributed with this cycle to agree with the added column and with $\mathbf{c}$.

More precisely, for every $1 \le j \le \tilde{k}$, denote by $\sigma_j = i^j_1, \ldots, i^j_{\ell_j}$ the j'th cycle in the representation of σ, where $i^j_1, \ldots, i^j_{\ell_j} \in [k]$ and $\sigma_j(i^j_t) = i^j_{t+1}$, for $1 \le t < \ell_j$, and $\sigma_j(i^j_{\ell_j}) = i^j_1$. Assume for simplicity of presentation and without loss of generality that all cycles in σ are of consecutive indices and in an increasing ordering, i.e., $i^j_{t+1} = i^j_t + 1$, for $1 \le t < \ell_j$ (this could be imposed by a reordering of the columns in C).

To construct $\hat{C}$ from C, for every cycle σ_j on indices $i_1, \ldots, i_{\ell_j} \in [k]$ (omitting the superscript j for the sake of clarity), set the columns corresponding these indices as follows. For the last index $i_{\ell_j} = i^j_{\ell_j}$, set column $\hat{C}_{i_{\ell_j}} \leftarrow \{0,1\}^{\ell}$. For every $1 \le t < \ell_j$, set the column $\hat{C}_t = C_{t-j+1} \oplus \hat{C}_{i_{\ell_j}}$. Note that we indeed defined a matrix C of ℓ rows and k columns.

To argue that a good distinguisher for the $\mathsf{DLPN}^{\sigma}_{\ell,k,\tau}$ problem makes the above a good distinguisher for the $\mathsf{DLPN}_{\ell,k-\tilde{k},\tau}$ problem, we show that if $(C, \mathbf{c})$ is sampled according to the first (resp. second) random variable of Definition 3, then the resulting $(\hat{C}, \hat{\mathbf{c}}, \sigma)$ is sampled according to the first (resp. second) random variable of Definition 7. To see this, first note that $\hat{\mathbf{c}} = \mathbf{c}$ and that if C is uniformly random, then so is $\hat{C}$.

Now, given $\mathbf{s} \in \mathbb{F}_2^{k-\hat{k}}$, define $\hat{\mathbf{s}} = \hat{s}_1, \ldots, \hat{s}_k$ as follows. Using the same notation as before, for every σ_j, let $\hat{\mathbf{s}}_{i_{\ell_j}} \leftarrow \mathbb{F}_2^{k}$, and proceeding in a decreasing order, fix $\hat{\mathbf{s}}_{i_t} = \hat{\mathbf{s}}_{it+1} + \mathbf{s}_{i_{t-j+1}}$ (recall that the $j-1$ long shift in the coordinate of $\mathbf{s}$ is needed because we added this number of new elements for the leftmost $j-1$ cycles). Note that if $\mathbf{s}$ is uniformly random, then so is $\hat{\mathbf{s}}$.

It remains to show that $\hat{C} \cdot (\hat{\mathbf{s}} \oplus \sigma(\hat{\mathbf{s}})) = C \cdot \mathbf{s}$. Indeed, it suffices to show that for every row $m \in [\ell]$ and for every cycle σ_j it holds that

$$
\begin{aligned}
\sum_{t=1}^{\ell_j} \hat{C}_{m,i_t} \cdot (\hat{\mathbf{s}}_{i_t} + \sigma(\hat{\mathbf{s}}_{i_t})) &= \sum_{t=1}^{\ell_j - 1} \hat{C}_{m,i_t} \cdot (\hat{\mathbf{s}}_{i_{t+1}} + \mathbf{s}_{i_{t-j+1}} + \hat{\mathbf{s}}_{i_{t+1}}) r \\
&\qquad + \hat{C}_{m,i_t} \cdot (\hat{\mathbf{s}}_{i_{\ell_j}} + \sigma(\hat{\mathbf{s}}_{i_{\ell_j}})) \\
&= \sum_{t=1}^{\ell_j - 1} \hat{C}_{m,i_t} \cdot \mathbf{s}_{i_{t-j+1}} + \hat{C}_{m,i_{\ell_j}} \cdot (\hat{\mathbf{s}}_{i_{\ell_j}} + \hat{\mathbf{s}}_{i_1}) \\
&= \sum_{t=1}^{\ell_j - 1} (C_{m,t-j+1} + \hat{C}_{m,i_{\ell_j}}) \cdot \mathbf{s}_{i_{t-j+1}}
\end{aligned}
$$

$$+ \ \hat{C}_{m,i_{\ell_j}} \cdot (\hat{\mathbf{s}}_{i_{\ell_j}} + \hat{\mathbf{s}}_{i_1})$$

$$= \sum_{t=1}^{\ell_j - 1} C_{m,t-j+1} \cdot \mathbf{s}_{i_{t-j+1}}$$

The first equality follows by the choice of $\hat{\mathbf{s}}_{i_t}$ and the assumption on σ_j. The second equality is true since our computations are done over $\mathbb{F}_2$. The third equality holds by the choice of $\hat{C}$. Finally, the last equality follows since $\hat{\mathbf{s}}_{i_1} = \hat{\mathbf{s}}_{i_{\ell_j}} + \sum_{t=1}^{\ell_j-1} \mathbf{s}_{i_{t-j+1}}$, and hence, $\hat{C}_{m,i_{\ell_j}} \cdot (\hat{\mathbf{s}}_{i_{\ell_j}} + \hat{\mathbf{s}}_{i_1}) = \hat{C}_{m,i_{\ell_j}} \cdot \sum_{t=1}^{\ell_j-1} \mathbf{s}_{i_{t-j+1}}$. □

In our construction, the permutation σ will be chosen to map $(\delta_0, \dots, \delta_{k-1}) \in \mathbb{F}_2^k$ to $(\delta'_0, \dots, \delta'_{k-1})$, where $\delta'_j = \delta_{j-1 \pmod{k}}$. Note that this σ consists of a single cycle of length k and, hence, the security of DLPN^σ is the same as that of DLPN with keys which are one bit shorter.

We are now just one step away from defining the right RK-KDM notion for our scheme. A detail that was overlooked in Equation (4) is that the key space $\mathcal{K} = \mathbb{F}_2^k$ and the message space $\mathcal{M} = \mathbb{F}_2^{\mathfrak{m}}$ are different, so we cannot write $G \cdot \mathbf{k}_{w,\epsilon_w}$. Furthermore, as in our protocols nobody will know neither $\mathbf{k}_{w,0}$ nor $\mathbf{k}_{w,1}$ (a problem which does not come up in previous works, because each P_i has its own pair of keys $\mathbf{k}^i_{w,0}, \mathbf{k}^i_{w,1}$), we need the garbled gate to also encrypt explicitly the external value ϵ_w.

We thus define an injection of the space $\mathcal{K} \times \mathbb{F}_2$ into the message space $\mathcal{M}$, which requires that $\mathfrak{m} \geq k+1$, via the following linear map:

$$\Psi : \begin{cases} \mathcal{K} \times \mathbb{F}_2 \longrightarrow \quad \mathcal{M} \\ \quad (\mathbf{k}, b) \ \longmapsto A \cdot (\mathbf{k}, b)^\intercal \end{cases}$$

for some matrix $A \in \mathbb{F}_2^{\mathfrak{m} \times (k+1)}$. In order to make the image of Ψ easily recognizable, so that we can efficiently recover its preimage when decrypting a garbled row, we pick the matrix A in the map Ψ such that we obtain:

$$\Psi : (\mathbf{k}, b) \longmapsto (0^{\mathfrak{m}-k-1} \| \mathbf{k} \| b) = \begin{pmatrix} \mathbf{0}_{(\mathfrak{m}-k-1)\times(k+1)} \\ I_k \| \mathbf{0}_{k \times 1} \\ \mathbf{0}_{1 \times k} \| 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{k}^\intercal \\ b \end{pmatrix}.$$

This choice of matrix A also simplifies somewhat the proof of Theorem 2 below.

We can now finally define the relevant notion of RK-KDM security for our scheme defined in Definition 6 (LIN-RK-KDM$^\sigma$ security), and show how we will use it to garble gates in our protocols. For security reasons, which will become apparent in the proofs, we need to make the assumption that the free-XOR correlation $\varDelta \in \mathbb{F}_2^k$ is of the form $(1, \varDelta', 0)$.

Let $\varDelta = (1, \varDelta', 0)$ with $\varDelta' \leftarrow \mathbb{F}_2^{k-2}$ be a secret value. Let H the XOF associated with the scheme ($\mathsf{KeyGen}^{\mathsf{XOF}}$, $\mathsf{Enc}_\tau^{\mathsf{XOF}}$, $\mathsf{Dec}^{\mathsf{XOF}}$) of Definition 5. In the following we think of the encryption scheme as being defined with respect to three possible keys $\varDelta$, $\sigma(\varDelta)$, and $\varDelta \oplus \sigma(\varDelta)$ chosen by (α, β). The variable $\mathbf{k}$ is defining a linearly homomorphic relation with respect to one of the keys and b is defining the linearly homomorphic key-dependent offset $\Psi(b \cdot \varDelta, b)$. With this understanding we define the following oracles:

$$\mathsf{Real}^\sigma_\varDelta : (\mathbf{k}, \alpha, \beta, \mathbf{m}, b, \mathsf{nonce}) \longmapsto$$
$$\mathsf{Enc}_\tau^{\mathsf{XOF}}(\ (\mathbf{m} \oplus \Psi(b \cdot \varDelta, b),\ \mathsf{nonce}),\ \mathbf{k} \oplus \alpha \cdot \varDelta \oplus \beta \cdot \sigma(\varDelta)\ )$$

The security game GarbleANDSec

This is a game between a challenger and an adversary. The challenger has access to the oracles $\mathsf{Fake}^{\sigma}_{\Delta}$ or $\mathsf{Real}^{\sigma}_{\Delta}$, which we denote by $\mathcal{O}$,

1. The challenger picks three bits $\epsilon_u, \epsilon_v, \epsilon_w \in \{0,1\}$, three keys $\mathbf{k}_{u,\epsilon_u}, \mathbf{k}_{v,\epsilon_v}, \mathbf{k}_{w,\epsilon_w} \in \mathbb{F}_2^k$, a nonce g and $b_u, b_v \in \{0,1\}$.
2. The challenger sets $b_w \leftarrow b_u \cdot b_v$ and $\lambda_t \leftarrow b_t \oplus \epsilon_t, t \in \{u,v,w\}$.
3. The challenger sets $\mathbf{k} \leftarrow \mathbf{k}_{u,\epsilon_u} \oplus \sigma(\mathbf{k}_{v,\epsilon_v})$.
4. The challenger computes the ciphertext
$$\mathfrak{ct}_{\epsilon_u,\epsilon_v} \leftarrow \mathsf{Enc}_{\tau}(\ (\Psi(\mathbf{k}_{w,\epsilon_w}, \epsilon_w),\ (g\|\epsilon_u\|\epsilon_v)),\ (\mathbf{k}_{u,\epsilon_u}, \mathbf{k}_{v,\epsilon_v})\ )$$
5. For $\alpha, \beta \in \{0,1\}, (\alpha,\beta) \neq (\epsilon_u, \epsilon_v)$ set
$$\ell_{\alpha,\beta} = (\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \oplus b_w.$$
6. The challenger computes, for $(\alpha,\beta) \neq (\epsilon_u,\epsilon_v)$ the three remaining ciphertexts:
$$\mathfrak{ct}_{\alpha,\beta} \leftarrow \mathcal{O}(\ \mathbf{k},\ \epsilon_u \oplus \alpha,\ \epsilon_v \oplus \beta,\ \Psi(\mathbf{k}_{w,\epsilon_w},\ \epsilon_w),\ \ell_{\alpha,\beta},\ (g\|\alpha\|\beta)\ )$$
7. The ciphertexts $(\mathfrak{ct}_{0,0}, \mathfrak{ct}_{1,0}, \mathfrak{ct}_{0,1}, \mathfrak{ct}_{1,1})$ along with the keys values, $(\mathbf{k}_{u,\epsilon_u}, \epsilon_u)$ and $(\mathbf{k}_{v,\epsilon_v}, \epsilon_v)$, are returned to the adversary.
8. The adversary goal is to determine which oracle the challenger is using.

Figure 2. The security game GarbleANDSec

$$\mathsf{Fake}^{\sigma}_{\Delta} : (\mathbf{k}, \alpha, \beta, \mathbf{m}, b, \mathsf{nonce}) \longmapsto (H(\mathsf{nonce}),\ \mathbf{c}), \qquad \mathbf{c} \leftarrow \mathcal{C},$$

where $\mathcal{C}$ is the ciphertext space, and forbid the following kind of queries: Let $\{(\mathbf{k}_i, \alpha_i, \beta_i, \mathbf{m}_i, b_i, \mathsf{nonce})\}_{i=1}^{q}$ be a sequence of queries under the same nonce. Such a sequence is not allowed if and only if there exist coefficients $c_1, \ldots, c_q \in \mathbb{F}_2$, not all zero, such that $\sum_{i=1}^{q} c_i \cdot (\alpha_i, \beta_i) = (0,0)$. We can now define our notion of LIN-RK-KDM$^{\sigma}$ security:

Definition 8 (LIN-RK-KDM$^{\sigma}$ secure encryption). *The encryption scheme* ($\mathsf{KeyGen}^{\mathsf{XOF}}$, $\mathsf{Enc}_{\tau}^{\mathsf{XOF}}$, $\mathsf{Dec}^{\mathsf{XOF}}$) *is said to be LIN-RK-KDM$^{\sigma}$ secure if the two oracles* $\mathsf{Real}^{\sigma}_{\Delta}$ *and* $\mathsf{Fake}^{\sigma}_{\Delta}$ *are computationally indistinguishable, when we forbid the above queries.*

The reason for the forbidden queries is in order to stop the distinguisher $\mathcal{D}$ from mounting a trivial attack. Take for example the simplest forbidden query, where $\mathcal{D}$ simply asks once for $(\mathbf{k}, 0, 0, \mathbf{m}, b, \mathsf{nonce})$. As none of the three possible secret keys depending on Δ has been applied, then $\mathcal{D}$ can just decrypt using $\mathbf{k}$ and see whether the oracle was implementing Real or Fake. For longer sequences, the idea is essentially the same, as the key-homomorphism of LPN would otherwise allow $\mathcal{D}$ to mount the same kind of attack simply by computing the linear combination defined by the c_i values.

Theorem 2. *Let $\Delta = (1, \Delta', 0)$ with $\Delta' \leftarrow \mathbb{F}_2^{k-2}$ be a secret value, then, assuming that DLPN is hard, the XOF-Based Two-Key LPN Encryption scheme (c.f. Definition 6) is LIN-RK-KDM$^{\sigma}$ secure, i.e.* $\mathsf{Real}^{\sigma}_{\Delta} \overset{c}{\equiv} \mathsf{Fake}^{\sigma}_{\Delta}$.

Proof. We let $\mathcal{R}_{\Delta} : \mathcal{M} \times \{0,1\}^* \to \mathbb{F}_2^{\ell \times k} \times \mathbb{F}_2^{\ell}$ denote the randomized function which ignores the key Δ and is given by $\mathcal{R}_{\Delta}(\mathbf{m}, \mathsf{nonce}) = (H(\mathsf{nonce}), \mathbf{c})$, where $\mathbf{c} \in \mathbb{F}_2^{\ell}$ is sampled uniformly at random. The hardness of the XOF variant of the DLPN problem implies that $\{\mathsf{Enc}_{\tau}^{1}((\cdot,\cdot),\Delta)\} \overset{c}{\equiv} \{\mathcal{R}_{\Delta}(\cdot,\cdot)\}$

and that $\{\mathsf{Enc}^1_\tau((\cdot,\cdot),\sigma(\Delta))\} \overset{c}{\equiv} \{\mathcal{R}_{\sigma(\Delta)}(\cdot,\cdot)\} \overset{c}{\equiv} \{\mathcal{R}_\Delta(\cdot,\cdot)\}$. Since DLPN$^\sigma$ is also hard (c.f. Lemma 1), we additionally have that $\{\mathsf{Enc}^1_\tau((\cdot,\cdot),\Delta\oplus\sigma(\Delta))\} \overset{c}{\equiv} \{\mathcal{R}_{\Delta\oplus\sigma(\Delta)}(\cdot,\cdot)\} \overset{c}{\equiv} \{\mathcal{R}_\Delta(\cdot,\cdot)\}$. Hence, we have that

$$\mathsf{Enc}^1_\tau(\ (\cdot,\cdot)\ ,\ \alpha\cdot\Delta\ \oplus\ \beta\cdot\sigma(\Delta)\ ) \overset{c}{\equiv} \{\mathcal{R}_{\Delta\oplus\sigma(\Delta)}(\cdot,\cdot)\} \overset{c}{\equiv} \{\mathcal{R}_\Delta(\cdot,\cdot)\} \tag{6}$$

for all $(\alpha,\beta)\in\{0,1\}^2\setminus(0,0)$. In order to prove the following Lemma, we need to define matrices $H_{\alpha,\beta,b}\in\mathbb{F}_2^{\mathfrak{m}\times k}$ for $\alpha,\beta,b\in\{0,1\}$ with $(\alpha,\beta)\neq(0,0)$ such that

$$H_{\alpha,\beta,b}\cdot(\ \alpha\cdot\Delta\ \oplus\ \beta\cdot\sigma(\Delta)\ )^{\mathsf{T}} = b\cdot\Psi(\Delta,b) \tag{7}$$

It is to enable this that we selected Δ such that the first bit of Δ is always equal to one and the last bit is equal to zero. This means we also can trivially recover Δ from $\Delta\oplus\sigma(\Delta)$ via a linear map, and the first element in $\Delta\oplus\sigma(\Delta)$ is always equal to one.

For the choice of σ given earlier, namely the permutation which rotates the bits Δ to the right by one, and the earlier embedding Ψ, we define the matrices $H_{\alpha,\beta,b}$ as follows; where we let P_σ denote the $k\times k$ permutation matrix corresponding to the permutation σ.

$$H_{1,0,b} = \begin{pmatrix} \mathbf{0}_{(\mathfrak{m}-k-1)\times k} \\ b\cdot I_k \\ b\ \|\ \mathbf{0}_{1\times(k-1)} \end{pmatrix},$$
$$H_{0,1,b} = \begin{pmatrix} \mathbf{0}_{(\mathfrak{m}-k-1)\times k} \\ b\cdot P_\sigma^{-1} \\ b\ \|\ \mathbf{0}_{1\times(k-1)} \end{pmatrix},$$
$$H_{1,1,b} = \begin{pmatrix} \mathbf{0}_{(\mathfrak{m}-k-1)\times (k+1)} \\ b\cdot Q \\ b\ \|\ \mathbf{0}_{1\times(k-1)} \end{pmatrix}.$$

where and Q is the matrix $k\times k$ matrix

$$\begin{pmatrix} 1\,0\,0\,0\,0\ldots0\,0 \\ 1\,1\,0\,0\,0\ldots0\,0 \\ 1\,1\,1\,0\,0\ldots0\,0 \\ \vdots \qquad\qquad \vdots \\ 1\,1\,1\,1\,1\ldots1\,1 \end{pmatrix}$$

Note, these matrices are correct according to equation (7) as we have, for $\Delta' = (\delta_1,\ldots,\delta_{k-2})$,

$$\begin{aligned} b\cdot I_k\cdot\Delta &= b\cdot\Delta \\ b\cdot P_\sigma^{-1}\cdot\sigma(\Delta) &= b\cdot P_\sigma^{-1}\cdot P_\sigma\cdot\Delta = b\cdot\Delta, \\ b\cdot Q\cdot(\ \Delta\ \oplus\ \sigma(\Delta)\ ) &= b\cdot Q\cdot(\ 1\ ,\ 1\oplus\delta_1\ ,\ \delta_1\oplus\delta_2\ ,\ \ldots\ ,\ \delta_{k-2}\ )^{\mathsf{T}} = b\cdot\Delta. \end{aligned}$$

It is to construct these explicit matrices that we selected the specific permutation σ and the specific embedding Ψ. These matrices allow us to prove the following Lemma.

Lemma 2. *Let $\mathcal{O}_\Delta$ denote the oracle machine given by*

$$\mathcal{O}_\Delta(\alpha,\beta,\mathbf{m},\mathsf{nonce}') = \mathsf{Enc}^1_\tau(\mathbf{m},\ \alpha\cdot\Delta\ \oplus\ \beta\cdot\sigma(\Delta)\ )$$

Note this oracle crucially uses standard LPN encryption, not the XOF variant, thus the nonce is ignored by the oracle. Then, there exists an efficient oracle machine

$$F^{(\cdot)} : (\mathbf{k}, \alpha, \beta, \mathbf{m}, b, \mathsf{nonce}') \longrightarrow (C, \mathbf{c})$$

such that for all $\Delta \in \mathbb{F}_2^k$, in the random oracle model, we have

$$\mathsf{Real}_\Delta^\sigma \overset{c}{\equiv} F^{\mathcal{O}_\Delta} \quad \textit{and} \quad \mathcal{R}_\Delta \equiv F^{\mathcal{R}_\Delta}.$$

Proof. We define F as follows: Given a query $(\mathbf{k}, \alpha, \beta, \mathbf{m}, b, \mathsf{nonce}')$ the machine F calls its oracle with inputs $(\mathbf{k}, \alpha, \beta, \mathbf{m}, \mathsf{nonce}')$, which returns a pair $(C', \mathbf{c}')$. The machine F now sets $C = C' \oplus G \cdot H_{\alpha,\beta,b}$ and $\mathbf{c} = \mathbf{c}' \oplus C \cdot \mathbf{k}$, and F then outputs $(C, \mathbf{c})$.

For a fixed key Δ, we aim to first show that $F^{\mathcal{O}_\Delta}$ is distributed identically to $\mathsf{Real}_\Delta^\sigma$. For a given query to F's oracle we obtain $(C', \mathbf{c}')$ where $C' \leftarrow \mathbb{F}_2^{\ell \times k}$ as this oracle uses standard LPN encryption (see Definition 4). It is clear that C is uniformly random since C' is uniformly random, so we program the random oracle used in $\mathsf{Real}_\Delta^\sigma$ to be $H(\mathsf{nonce}' \| \alpha \| \beta) = C = C' \oplus G \cdot H_{\alpha,\beta,b}$. Thus the matrix C output by F is exactly the same one obtained by calling the RO in $\mathsf{Real}_\Delta^\sigma$. This shift is indistinguishable by the semantic security of the basic LPN scheme.

We also notice that the value $\mathbf{c}$ output by F is equal to, where $\mathbf{e} \leftarrow \mathsf{Ber}_\tau^\ell$,

$$\begin{aligned}
\mathbf{c} &= \mathbf{c}' \oplus C \cdot \mathbf{k} \\
&= (\, C' \cdot (\, \alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta)\,) \oplus \mathbf{e} \oplus G \cdot \mathbf{m}\,) \oplus C \cdot \mathbf{k} \\
&= (\, C \oplus G \cdot H_{\alpha,\beta,b}\,) \cdot (\, \alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta)\,) \oplus \mathbf{e} \oplus G \cdot \mathbf{m} \oplus C \cdot \mathbf{k} \\
&= C \cdot (\, \alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta) \oplus \mathbf{k}\,) \oplus \mathbf{e} \\
&\qquad\qquad \oplus G \cdot \mathbf{m} \oplus G \cdot H_{\alpha,\beta,b} \cdot (\, \alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta)\,) \\
&= C \cdot (\, \alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta) \oplus \mathbf{k}\,) \oplus \mathbf{e} \oplus G \cdot (\, \mathbf{m} \oplus b \cdot \Psi(\Delta, b)\,) \\
&= \mathsf{Enc}_\tau^{\mathsf{XOF}}(\, (\mathbf{m} \oplus b \cdot \Psi(\Delta, b),\ \mathsf{nonce}' \| \alpha \| \beta),\ \alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta) \oplus \mathbf{k}\,).
\end{aligned}$$

Thus we have indeed that $\mathsf{Real}_\Delta^\sigma \overset{c}{\equiv} F^{\mathcal{O}_\Delta}$.

The proof that $\mathcal{R}_\Delta \overset{c}{\equiv} F^{\mathcal{R}_\Delta}$ is (essentially) immediate as the transformation applied by the machine F is invertible. However, one will notice that the machine F has an oracle which takes four input values, whilst when writing $F^{\mathcal{R}_\Delta}$ we are supplying an oracle which takes two input values only; which it does by ignoring any key dependent value which is given by $\alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta)$. Thus by a hybrid argument, and using the fact that $\{\mathcal{R}_\Delta\} \overset{c}{\equiv} \{\mathcal{R}_{\alpha \cdot \Delta \oplus \beta \cdot \sigma(\Delta)}\}$ for all $(\alpha, \beta) \neq (0, 0)$ the result follows. □

The proof of Theorem 2 now follows from the above Lemma, since we have:

$$\{\mathsf{Real}_\Delta^\sigma\} \overset{c}{\equiv} \{F^{\mathcal{O}_\Delta}\} \overset{c}{\equiv} \{F^{\mathcal{R}_\Delta}\} \overset{c}{\equiv} \{\mathcal{R}_\Delta\},$$

where the middle equality follows from Equation 6 □

We end this section by showing, *intuitively*, why the garbling method using our (XOF-Based) Two-Key LPN Encryption is secure. Consider the garbling game in Figure 2, which models an adversary that is trying to learn something about a garbled AND gate, given only the pair of keys and external values for the active path. From our previous discussion, if the LIN-RK-KDM$^\sigma$ problem is hard then the adversary is clearly unable to win this game. We remark that this game just provides the intuition around the security of our garbling protocols, which will not explicitly use it in their respective proofs.

Protocol Π_{Garble}

Let $\mathcal{E}_\tau^{\mathsf{XOF}} = \{\mathsf{KeyGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau\}$ be a XOF-based two-key LPN encryption scheme, where τ is a parameter of the scheme. Let $\mathcal{K} = \mathbb{F}_2^k$.

Garbling:

```
1. Each $P_i$ samples $\Delta^i \leftarrow \mathbb{F}_2^{k-2}$ and calls $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ to compute $[\Delta^i] \leftarrow \mathsf{Input}(P_i)$.
2. Set $[\Delta] \leftarrow (1, \mathbf{0}) \oplus \bigoplus_{i\in[n]}(0, [\Delta^i], 0)$.
3. For every input wire $w \in W_{\mathsf{in}}$ and output wire of an AND gate, parties do:
   – Call $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ obtaining a shared random bit $[\lambda_w] \leftarrow \mathsf{GenBit}()$.
   – Each $P_i$ samples $\mathbf{k}^i_{w,0} \leftarrow \mathcal{K}$ and call $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ on $[\mathbf{k}^i_{w,0}] \leftarrow \mathsf{Input}(P_i)$.
   – Set $[\mathbf{k}_{w,0}] \leftarrow \bigoplus_{i\in[n]}[\mathbf{k}^i_{w,0}]$ and $[\mathbf{k}_{w,1}] \leftarrow [\mathbf{k}_{w,0}] \oplus [\Delta]$.
4. For every wire $w$ in the circuit which is the output of a XOR gate:
   – Parties compute the mask on the output wire $[\lambda_w] \leftarrow [\lambda_u] \oplus [\lambda_v]$.
   – Parties compute $[\mathbf{k}_{w,0}] \leftarrow [\mathbf{k}_{v,0}] \oplus [\mathbf{k}_{v,0}]$ and set $[\mathbf{k}_{w,1}] \leftarrow [\mathbf{k}_{w,0}] \oplus [\Delta]$
5. For every wire $w$ in the circuit which is the output of an AND gate and for $\alpha, \beta \in \{0,1\}$, parties call
   $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ to compute
   (a) $[\epsilon_{w,\alpha,\beta}] \leftarrow ([\lambda_u] \oplus \alpha) \cdot ([\lambda_v] \oplus \beta) \oplus [\lambda_w]$.
   (b) $[\mathbf{k}_{w,\alpha,\beta}] \leftarrow [\mathbf{k}_{w,0}] \oplus ([\Delta] \cdot [\epsilon_{w,\alpha,\beta}])$.
   (c) The encryption $(C^{w,\alpha,\beta}, [\mathbf{c}^{w,\alpha,\beta}])$, given by
       $$\mathsf{Enc}_\tau\Big( \ (\ \Psi([\mathbf{k}_{w,\alpha,\beta}], [\epsilon_{w,\alpha,\beta}]),\ (g\|\alpha\|\beta)\ ),\ ([\mathbf{k}_{u,\alpha}], [\mathbf{k}_{v,\beta}])\ \Big),$$
       where $g$ is a unique gate identifier.
   (d) Parties call $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ to open the values $\lambda_w \leftarrow \mathsf{Output}([\lambda_w], P_i)$ corresponding to party $P_i$'s output
       values.
```

Open Garbling:

```
1. Parties call $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ to open $\mathbf{c}^{w,\alpha,\beta} \leftarrow \mathsf{Open}([\mathbf{c}^{w,\alpha,\beta}]), \alpha, \beta \in \{0,1\}$.
2. Set the garbled gates to be $\tilde{g}_{w,\alpha,\beta} = (C^{w,\alpha,\beta}, \mathbf{c}^{w,\alpha,\beta})$ for $\alpha, \beta \in \{0,1\}$.
```

Figure 3. The protocol for authenticated garbling Π_{Garble}

4 MPC from Fully Authenticated LPN-Garbling

We use the garbling technique introduced in the previous section to describe our first protocol. As we said before, we evaluate the entire garbled circuit using a generic, actively secure MPC protocol.

In particular, given a secret shared key $[\mathbf{k}]$, message $[\mathbf{m}]$, and noise vector $[\mathbf{e}]$ (obtained by calling $\mathsf{GenBit}()$ and Mult in $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$), the parties can compute a secret shared ciphertext $(C, [\mathbf{c}])$, where C is in the clear, using a double-key encryption scheme $\mathcal{E}_\tau^{\mathsf{XOF}}$ as described in Definition 6. Since both the generation and opening of the garbled circuit are done using an active secure MPC system, the reconstructed garbled circuit is guaranteed to be correct and thus there is no need for any consistency checks during the evaluation phase. The downside of this simple approach is that the amount of multiplications required to produce noise vectors $[\mathbf{e}]$ with the right distribution could be prohibitively high in some scenarios.

4.1 Garbling

Our garble protocol Π_{Garble}, is described in Figure 3. First, the parties produce, in an actively-secure way, shares of the global key $[\Delta]$, the wire labels $[\mathbf{k}^i_{0,w}], [\mathbf{k}^i_{1,w}]$ and the wire masks $[\lambda_w]$ for the garbled circuit using $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$. Then, for each AND gate g with input wires u, v and output wire

Functionality $\mathcal{F}_{\text{Preprocessing}}$

Let $\mathcal{E}_\tau^{\mathsf{XOF}} = \{\mathsf{KeyGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau\}$ be a XOF-based two-key LPN encryption scheme, where τ is a parameter of the scheme. The functionality runs with parties $P_1, \ldots, P_n$ and a full-threshold adversary.

Garbling: On input $(\texttt{Garbling}, C_f)$ from all parties, the functionality does as follows:

- Sample a global difference $\Delta = (1, \Delta', 0)$ where $\Delta' \leftarrow \{0,1\}^{k-2}$.
- Passing topologically through all the wires $w \in W$ of the circuit:
  - If w is an input wire for some P_i:
    1. Sample $\lambda_w \leftarrow \{0,1\}$. If P_i is corrupt, instead receive λ_w from the adversary.
    2. Sample a key $\mathbf{k}_{w,0} \leftarrow \{0,1\}^k$. Define $\mathbf{k}_{w,1} = \mathbf{k}_{w,0} \oplus \Delta$.
  - If w is the output of an AND gate:
    1. Sample $\lambda_w \leftarrow \{0,1\}$.
    2. Sample a key $\mathbf{k}_{w,0} \leftarrow \{0,1\}^k$. Set $\mathbf{k}_{w,1} = \mathbf{k}_{w,0} \oplus \Delta$.
  - If w is the output of a XOR gate, and u and v its input wires:
    1. Compute and store $\lambda_w = \lambda_u \oplus \lambda_v$.
    2. Set $\mathbf{k}_{w,0} = \mathbf{k}_{u,0} \oplus \mathbf{k}_{v,0}$ and $\mathbf{k}_{w,1} = \mathbf{k}_{w,0} \oplus \Delta$.
- For every AND gate $g \in C_\wedge$, the functionality computes $e_{w,\alpha,\beta} = (\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \oplus \lambda_w$ and stores the four entries $\tilde{g}_{\alpha,\beta}$ of the garbled version of g as, for $(\alpha, \beta) \in \{0,1\}^2$:

$$\mathsf{Enc}_\tau\left(\ (\Psi(\ \mathbf{k}_{w,0} \oplus e_{w,\alpha,\beta} \cdot \Delta,\ e_{w,\alpha,\beta}\ ),\ (g||\alpha||\beta)\ ) \,; (\mathbf{k}_{u,\alpha}, \mathbf{k}_{v,\beta})\ \right).$$

- Wait for an input from the adversary. If it inputs $\texttt{OK}$ then output λ_w to P_i for each of its corresponding circuit-output wires $w \in W_{\mathsf{out}_i}$. Otherwise, output $\perp$ and terminate.

Open Garbling: On receiving $(\texttt{OpenGarbling})$ from all parties, when the **Garbling** command has already run successfully, the functionality sends to the adversary the values $\tilde{g}_{\alpha,\beta}$ for all $g \in C_\wedge$ and waits for a reply.

- If the adversary returns $\perp$ then the functionality aborts. Otherwise, if it receives an $\texttt{OK}$ message, it sends to all parties the garbled circuit $\tilde{g}_{\alpha,\beta}$ for all $g \in C_\wedge$ and $\alpha, \beta \in \{0,1\}$.

Figure 4. The Authenticated Preprocessing Functionality $\mathcal{F}_{\text{Preprocessing}}$.

w, and for each $\alpha, \beta \in \{0,1\}$, the parties compute authenticated additive sharing of the values

$$[\epsilon_{w,\alpha,\beta}] \leftarrow ([\lambda_u] \oplus \alpha) \cdot ([\lambda_v] \oplus \beta) \oplus [\lambda_w].$$

Thus the garbled gate for each AND gate is obtained by calling $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ to evaluate the following encryptions

$$(C^{w,\alpha,\beta}, [\mathbf{c}^{w,\alpha,\beta}]) = \mathsf{Enc}_\tau^{\mathsf{XOF}}\Big(\ (\ \Psi([\mathbf{k}_{w,\alpha,\beta}], [\epsilon_{w,\alpha,\beta}]),\ (g\|\alpha\|\beta)\ ),\ ([\mathbf{k}_{u,\alpha}], [\mathbf{k}_{v,\beta}])\ \Big)$$

where $\alpha, \beta \in \{0,1\}$, g is a unique gate identifier and $\mathbf{k}_{w,\alpha,\beta} = k_{w,0} \oplus \epsilon_{w,\alpha,\beta} \cdot \Delta$. Finally, parties open the masks for all the output wires of the circuit, so that they will be able to recover the output at the end of the evaluation phase.

When the garbled circuit is opened, using $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$, the parties reconstruct the four values $(C^{w,\alpha,\beta}, \mathbf{c}^{w,\alpha,\beta})$, $\alpha, \beta \in \{0,1\}$, and set these to be the garbled gates $\tilde{g}_{\alpha,\beta}$. Note that the first component $C^{w,\alpha,\beta}$ of the ciphertexts in the garbled gates does not need to be stored, as it can be generated on the fly by applying the XOF to the relevant $\mathsf{nonce} = (g\|\alpha\|\beta)$.

In order to see how the garbling is correct, note that the output of the AND gate is exactly the value $(\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta)$. Hence, assuming $\lambda_w = 0$, we have two cases: if $(\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) = 0$, then $\epsilon_{w,\alpha,\beta} = 0$ and $k_{w,\alpha,\beta} = k_{w,0}$; otherwise $\epsilon_{w,\alpha,\beta} = 1$ and $k_{w,\alpha,\beta} = k_{w,0} \oplus \Delta$. The result is reversed if $\lambda_w = 1$.

In more formality, we state the following theorem. It has a relatively standard proof, which follows the pattern of previous works on n-party garbling.

Protocol Π_{Evaluate}

Inputs: A circuit C_f computing the function f, which consists of XOR and AND gates. Let W be the set of all wires in C_f, W_{in_i} be the set of input wires for party P_i, and W_{out} be the set of output wires. Each party P_i has input $\{\rho_w\}_{w \in W_{\mathsf{in}_i}}$. The parties execute the following commands in sequence.

Preprocessing: This sub-task is performed as follows.

- Call **Garbling** on $\mathcal{F}_{\text{Preprocessing}}$ with input C_f.
- Each party P_i obtains the λ_w wire masks for every wire $w \in W_{\mathsf{out}}$.

Online Computation: This sub-task is performed as follows.

1. For all input wires w from party P_i with input ρ_w on this wire, P_i calls $\mathcal{F}_{\mathsf{MPC}}$ on $[\rho_w] \leftarrow \mathsf{Input}(P_i)$. If this step returns abort then the protocol aborts.
2. The parties call $\mathcal{F}_{\mathsf{MPC}}$ to compute $\epsilon_w \leftarrow \mathsf{Open}([\rho_w] \oplus [\lambda_w])$ and $\mathbf{k}_{w,\epsilon_w} \leftarrow \mathsf{Open}([\mathbf{k}_{w,\epsilon_w}])$ for all input wires w. If any of these steps returns abort then the protocol aborts.
3. Parties call **Open Garbling** on $\mathcal{F}_{\text{Preprocessing}}$
4. The gates are now executed as follows in topological order:
   (a) For all XOR gates parties locally compute the public values via $\epsilon_w \leftarrow \epsilon_u \oplus \epsilon_v$ and the key values via $\mathbf{k}_{w,\epsilon_w} \leftarrow \mathbf{k}_{u,\epsilon_u} \oplus \mathbf{k}_{v,\epsilon_v}$.
   (b) For AND gates parties perform the following steps:
      - Decrypt the ciphertext $(C^{w,\epsilon_u,\epsilon_v}, \mathbf{c}^{w,\epsilon_u,\epsilon_v})$ using the key $(\mathbf{k}_{u,\epsilon_u}, \mathbf{k}_{v,\epsilon_v})$ to obtain a message $\mathbf{m}$.
      - Invert Ψ on $\mathbf{m}$ to obtain $(\mathbf{k}_{w,\epsilon_w}, \epsilon_w)$.
5. Each party will obtain a public value ϵ_w for every output wire w. From this, the actual output can be obtained by party P_i by computing $y_w \leftarrow \epsilon_w \oplus \lambda_w$.

Figure 5. The protocol for evaluation in the authenticated garbling case Π_{Evaluate}

Theorem 3. *Let $\mathcal{E}_\tau^{\mathsf{XOF}}$ be a XOF-based two-key LPN encryption scheme with parameter τ. The protocol Π_{Garble}, given in Figure 3, UC-securely computes the functionality $\mathcal{F}_{\text{Preprocessing}}$ (see Figure 4) in the presence of a static, active adversary corrupting up to $n-1$ parties in the $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$-hybrid model.*

Proof. Let $\mathcal{A}$ denote a PPT adversary corrupting a subset $I \subsetneq [n]$ of parties. We will construct a simulator $\mathcal{S}$ that plays the roles of the honest parties on arbitrary inputs and interacts with $\mathcal{A}$. Since most of the protocol consists of interactions with $\mathcal{F}_{\mathsf{MPC}}$ based on value identifiers, it is straightforward. The only values sent in the protocol are openings through $\mathcal{F}_{\mathsf{MPC}}$.

Description of the simulator $\mathcal{S}$.

- $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPC}}$ on input the corrupt shares provided by $\mathcal{A}$ for all parties $\{P_i\}_{i \in I}$. For honest parties, it inputs random values. In this way obtain $[\Delta]$ and $[\mathbf{k}_{w,0}]$ for each input wire and output wire of AND gates.
- Local computation requires no simulation.
- For each AND gates, emulates $\mathcal{F}_{\mathsf{MPC}}$ obtaining $[\epsilon_{w,\alpha,\beta}]$, for $\alpha, \beta \in \{0,1\}$, and the corresponding garbled gates $\tilde{g}_{w,\alpha,\beta}$.
- Every time a value is opened, emulates $\mathcal{F}_{\mathsf{MPC}}$. If the result is abort, then forwards abort to the functionality and terminates.

Indistinguishability. It is straightforward to see that the transcript of both ideal and real executions only consist of random values and hence they are indistinguishable. ☐

4.2 Evaluation

The protocol Π_{Evaluate}, in Figure 5, describes how parties evaluate the garbled circuit. This protocol is very similar to that of HSS, where everyone evaluates the garbled circuit obtained in the preprocessing phase by broadcasting their inputs XORed with the corresponding wire mask. The main difference with HSS is that, as there is a single output key $\mathbf{k}_{w,\epsilon_w}$ for every wire, rather than one such key per party, parties need to explicitly obtain the masked wire value ϵ_w when decrypting $\tilde{g}_{\epsilon_u,\epsilon_v}$. Once the whole circuit has been evaluated, making use of the output wire masks they obtained at the preprocessing stage, parties can unmask their corresponding outputs and learn their intended result.

It is important to note that, unlike in HSS and due to the active security of the base MPC system, all among the garbled circuit, input keys $\mathbf{k}_{w,\epsilon_w}$ and masked inputs ϵ_w are guaranteed to be correct. Since the rest of this phase is purely local computation, this essentially ensures the output is correct. The security of the protocol, provided by the following theorem, follows from adapting the proof of our more complex unauthenticated garbling protocol in Section 5. In other words, the proof of Theorem 4 is just a specialised version of the proof of Theorem 6.

Theorem 4. *Let f be an n-party functionality and $\mathcal{E}_\tau^{\mathsf{XOF}}$ a XOF-based two-key LPN encryption scheme with parameter τ. The protocol Π_{Evaluate} UC-securely computes f in the presence of a static, active adversary corrupting up to $n-1$ parties in the $\{\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\text{Preprocessing}}\}$-hybrid model.*

4.3 Overall Complexity Fully Authenticated Variant

We let r_m denote the number of rounds needed to perform a secure multiplication operation, r_i the number of rounds needed for inputting a value into the MPC engine, and r_o the number of rounds needed to output/open a value. We discuss the complexity, in terms of calls to the underlying base MPC functionality. As before, we let $|C_\wedge|$ denote the number of AND gates in the circuit, $|C_i|$ the number of input gates and $|C_o|$ the number of output gates.

- Step 1 requires each party to execute $k-2$ calls to Input.
- Recall each call to GenBit requires each party to execute one call to Input. Thus the computation in step 3 require each party to execute $(|C_\wedge| + |C_i|) \cdot (1+k)$ call to Input. Note the inputs in steps 1 and 3 can all be performed in parallel, thus this requires r_i rounds.
- Step 5a requires $|C_\wedge|$ calls to Mult (in order to compute $[\lambda_u] \cdot [\lambda_v]$ for every AND gate).
- Step 5b requires $4 \cdot k \cdot |C_\wedge|$ calls to Mult. The multiplications in steps 5a (resp. 5b) can be performed in parallel, although the two steps need to be performed sequentially. Thus these two steps require $2 \cdot r_m$ rounds of communication.
- Step 5c requires $3 \cdot \ell$ calls to GenBit and thus each party needs to execute $12 \cdot \ell \cdot |C_\wedge|$ calls to Input and $8 \cdot \ell \cdot |C_\wedge|$ calls to Mult. The entire step requiring $r_i + 2 \cdot r_m$ rounds of communication, but these can be performed with the previous inputs and multiplications.
- Step 1 requires $4 \cdot \ell \cdot |C_\wedge|$ calls to Open and requires r_o rounds.
- Finally, step 5d requires $|C_o|$ call to Output, but the rounds cost can be amortized with the previous outputs.

Note the evaluation procedure can be performed in $r_i + 2 \cdot r_o$ rounds and requires $|C_i|$ calls to Input and $k \cdot |C_i|$ calls to Open. Thus in total cost per party of this garbling and the following evaluation procedure is given in the following table

	Preprocessing	Evaluation
Input	$(k-2)+(\lvert C_\wedge\rvert+\lvert C_i\rvert)\cdot(1+k)+12\cdot\ell\cdot\lvert C_\wedge\rvert$	$\lvert C_i\rvert$
Open	$4\cdot\ell\cdot\lvert C_\wedge\rvert$	$(k+1)\cdot\lvert C_i\rvert$
Output	$\lvert C_o\rvert$	$-$
Mult	$(4\cdot k+8\cdot\ell+1)\cdot\lvert C_\wedge\rvert$	$-$
Rounds	$r_i+r_o+2\cdot r_m$	$r_i+2\cdot r_o$

4.4 Optimizations

The following optimization for circuit evaluation might be useful in the case where the circuit has many input wires, and relatively few AND gates. It trades the authentication of the input keys in lines 1-3, with the authentication of the obtained wire labels ϵ_w in line 4b.

Thus we replace lines $1-3$ in Figure 5 by the parties executing

1. For all input wires w associated to party P_i, parties call $\lambda_w \leftarrow \mathsf{Output}([\lambda_w], P_i)$.
2. For all input wires w associated to party P_i with input ρ_w on this wire, party P_i executes broadcasts $\epsilon_w = \rho_w \oplus \lambda_w$.
3. The parties call $\mathbf{k}_{w,\epsilon_w} \leftarrow \mathsf{Open}(\mathsf{Convert}([\mathbf{k}_{w,\epsilon_w}]))$.

Thus we have replaced $|C_i|$ calls to Input and $k \cdot |C_i|$ calls to Open, with $|C_i|$ calls to Output and $(k-1)\cdot|C_i|$ calls to Open^P and Convert. This *may* be more efficient in some situations.

We still, however, need to authenticate the output for this optimization. To do this we execute the following check after the garbled circuit has been evaluated. Notice, that parties learn the signal bits ϵ_u, ϵ_v on the input to every AND gate, and they also learn the output signal bit ϵ_w. The output signal bit was also computed in the pre-processing, and hence it is authenticated, so the parties can now execute the underlying authentication of the opening of $[\epsilon_w]$ to ϵ_w, as if it had been done in the MPC engine. Thus, if the base MPC protocol was based on TinyOT-like protocol [KOS15, HSS17, WRK17b], then the MAC values would be checked on all signal bits of the outputs of an AND gate. If the adversary introduced errors in the garbled circuit, then either they were "unlucky" and do not change the active path through the circuit, or at least one ϵ_w is changed, which is then caught with the authentication.

5 MPC from Unauthenticated LPN-Garbling

Whilst the protocol described in the previous section is intuitive and achieves our goals for the evaluation phase, the usage of an authenticated garbling functionality incurs a larger number of oblivious operations in the preprocessing phase. In this section, we turn to use an unauthenticated preprocessing functionality, in the style of HSS, in order to improve the efficiency of this phase. Our unauthenticated garbling protocol makes clever use of the homomorphic properties of the LPN encryption scheme. This turns out to be especially efficient when a large proportion of parties are assumed to be honest.

Our protocols and functionalities in this section are parametrised by a value $c \in \mathbb{R}$ that represents the proportion $1/c$ of parties that are assumed honest. In other words, our protocols will have n/c honest parties, with $1 < c \leq n$. Note that when $2 \leq c$, we obtain a protocol which is secure against a dishonest majority, and by setting $c = n$ we would go back to the case of a full-threshold adversary. As expected, the value of c greatly affects the performances of our construction. We remark that allowing the possibility of having more than a single honest party is a highly reasonable assumption in a large scale setting.

Functionality $\mathcal{F}^{n/c}_{\text{Preprocessing}}$

Let $\mathcal{E}^{\mathsf{XOF}}_\tau = \{\mathsf{KeyGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau\}$ be a XOF-based two-key LPN encryption scheme, where τ is a parameter of the scheme. The functionality runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$ corrupting up to $(c-1) \cdot n/c$ parties.

Garbling: On input ($\texttt{Garbling}, C_f$) from all parties, where C_f is a boolean circuit, denote by W its set of wires, $W_{\mathsf{in}} = \{W_{\mathsf{in}_i}\}_{i\in[n]}$ its set of input wires (where W_{in_i} corresponds to party P_i), $W_{\mathsf{out}} = \{W_{\mathsf{out}_i}\}_{i\in[n]}$ its set of output wires (where W_{out_i} corresponds to party P_i) and $C_\wedge, C_{\mathsf{out}}$ its set of AND gates and output gates, respectively. Then the functionality does as follows:

- Sample a global difference $\Delta = (1, \Delta', 0)$ where $\Delta' \leftarrow \{0,1\}^{k-2}$.
- Passing topologically through all the wires $w \in W$ of the circuit:
  - If w is an input wire for some P_i:
    1. Sample $\lambda_w \leftarrow \{0,1\}$. If P_i is corrupt, instead receive λ_w from $\mathcal{A}$.
    2. Sample a key $\mathbf{k}_{w,0} \leftarrow \{0,1\}^k$. Define $\mathbf{k}_{w,1} = \mathbf{k}_{w,0} \oplus \Delta$.
  - If w is the output of an AND gate:
    1. Sample $\lambda_w \leftarrow \{0,1\}$.
    2. Sample a key $\mathbf{k}_{w,0} \leftarrow \{0,1\}^k$. Set $\mathbf{k}_{w,1} = \mathbf{k}_{w,0} \oplus \Delta$.
  - If w is the output of a XOR gate, and u and v its input wires:
    1. Compute and store $\lambda_w = \lambda_u \oplus \lambda_v$.
    2. Set $\mathbf{k}_{w,0} = \mathbf{k}_{u,0} \oplus \mathbf{k}_{v,0}$ and $\mathbf{k}_{w,1} = \mathbf{k}_{w,0} \oplus \Delta$.
- For every AND gate $g \in C_\wedge$, the functionality computes $e_{w,\alpha,\beta} = (\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \oplus \lambda_w$ and stores the four entries of the garbled version of g as, for $(\alpha, \beta) \in \{0,1\}^2$

$$\tilde{g}_{\alpha,\beta} = \mathsf{Enc}_\tau \,(\ (\Psi(\ \mathbf{k}_{w,0} \oplus e_{w,\alpha,\beta} \cdot \Delta,\ e_{w,\alpha,\beta}\ ),\ (g||\alpha||\beta)\ )\ ;\ (\mathbf{k}_{u,\alpha}, \mathbf{k}_{v,\beta})\ )\,.$$

- For every output gate g associated to a set of parties $\hat{P} = \{P_{i_1}, \ldots, P_{i_{|\hat{\mathcal{P}}|}}\}$, with input wire u and output wire w, store its entries as, for $\alpha \in \{0,1\}$:

$$\tilde{g}_\alpha = \mathsf{Enc}_{\tau_d}\Big(\ (\ (\xi^{i_1}_{w,\alpha} \| \ldots \| \xi^{i_{|\mathcal{P}|}}_{w,\alpha}),\ (g\|\alpha\|0)\ ),\ (\mathbf{k}_{u,\alpha}, \mathbf{0})\ \Big)$$

- Wait for an input from $\mathcal{A}$. Terminate if it output $\perp$. Otherwise, if it inputs $\texttt{OK}$ then output λ_w to P_i for each of its corresponding circuit-input and circuit-output wires $w \in W_{\mathsf{in}_i} \cup W_{\mathsf{out}_i}$. Additionally, for $w \in W_{\mathsf{in}}, \alpha \in \{0,1\}$, secret share $\mathbf{k}_{w,\alpha} = \bigoplus_{i\in[n]} \mathbf{k}^i_{w,\alpha}$ and send $\mathbf{k}^i_{w,\alpha}$ to P_i for $i \in [n]$. Finally, for $i \in [n]$ and $w \in W_{\mathsf{out}_i}$, send $\xi^i_{w,0}$ and $\xi^i_{w,1}$ to P_i.

Open Garbling: On receiving ($\texttt{OpenGarbling}$) from all parties, when the **Garbling** command has already run successfully, the functionality sends to $\mathcal{A}$ the values $\tilde{g}_{\alpha,\beta}$ for all $g \in C_\wedge$ and waits for a reply.

- If $\mathcal{A}$ returns $\perp$ then the functionality aborts. Otherwise, it receives $\texttt{OK}$ and additive errors $\mathbf{e} = \{\{e^{\alpha,\beta}_g\}_{\alpha,\beta\in\{0,1\},g\in C_\wedge}, \{e^\alpha_g\}_{\alpha\in\{0,1\},g\in C_{\mathsf{out}}}\}$ chosen by $\mathcal{A}$. After receiving these, it sends to all parties the garbled circuit as $\tilde{g}_{\alpha,\beta} \oplus e^{\alpha,\beta}_g$ for each $g \in C_\wedge$ and $\alpha, \beta \in \{0,1\}$ and $\tilde{g}_\alpha \oplus e^\alpha_g$ for each $g \in C_{\mathsf{out}}$ and $\alpha \in \{0,1\}$.

Figure 6. The Unauthenticated Preprocessing Functionality $\mathcal{F}^{n/c}_{\text{Preprocessing}}$

Protocol $\Pi^{n/c}_{\mathsf{Garble}}$

Let $\mathcal{E}^{\mathsf{XOF}}_{\tau} = \{\mathsf{KeyGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau\}$ be a XOF-based two-key LPN encryption scheme, where τ is a parameter of the scheme. Let $\mathcal{K} = \mathbb{F}_2^k$. Let $[x]$ and $\langle x \rangle$ denote respectively an authenticated and unauthenticated additive sharing of x.

Garbling:

1. Each P_i generates a random value $\Delta^i \in \mathbb{F}_2^{k-2}$ and call $\langle \Delta^i \rangle \leftarrow \mathsf{Input}^P(P_i)$ of $\mathcal{F}_{\mathsf{MPC}}$.
2. Set $\langle \Delta \rangle \leftarrow (1, \mathbf{0}) \oplus_i (0, \langle \Delta^i \rangle, 0)$.[a]
3. For every wire w in the circuit which is either an input wire or the output of an AND gate, parties do as follows:
   - Create a secret random bit $[\lambda_w] \leftarrow \mathsf{GenBit}()$.
   - Each P_i generates a random $\mathbf{k}^i_{w,0} \in \mathcal{K}$ and calls $\langle \mathbf{k}^i_{w,0} \rangle \leftarrow \mathsf{Input}^P(P_i)$.
   - Set $\langle \mathbf{k}_{w,0} \rangle \leftarrow \oplus_i \langle \mathbf{k}^i_{w,0} \rangle$ and $\langle \mathbf{k}_{w,1} \rangle \leftarrow \langle \mathbf{k}_{w,0} \rangle \oplus \langle \Delta \rangle$.
4. For every wire w in the circuit which is the output of a XOR gate (with input wires u and v) parties locally set:
   - $[\lambda_w] \leftarrow [\lambda_u] \oplus [\lambda_v]$.
   - $\langle \mathbf{k}_{w,0} \rangle \leftarrow \langle \mathbf{k}_{u,0} \rangle \oplus \langle \mathbf{k}_{v,0} \rangle$ and $\langle \mathbf{k}_{w,1} \rangle \leftarrow \langle \mathbf{k}_{w,0} \rangle \oplus \langle \Delta \rangle$.
5. For every wire w in the circuit which is the output of an AND gate g (with input wires u and v), for $\alpha, \beta \in \{0,1\}$,
   (a) Parties call $\mathcal{F}_{\mathsf{MPC}}$ to compute $[\epsilon_{w,\alpha,\beta}] \leftarrow ([\lambda_u] \oplus \alpha) \cdot ([\lambda_v] \oplus \beta) \oplus [\lambda_w]$,
   (b) Parties call the command $\langle \epsilon_{w,\alpha,\beta} \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([\epsilon_{w,\alpha,\beta}])$. [a]
   (c) Parties locally compute $\langle \mathbf{k}_{w,\alpha,\beta} \rangle \leftarrow \langle \mathbf{k}_{w,0} \rangle \oplus \langle \epsilon_{w,\alpha,\beta} \cdot \Delta \rangle$.
   (d) Each party P_i computes the encryptions $(C^{w,\alpha,\beta}, \mathbf{c}^{i,w,\alpha,\beta})$ given by
   $$\mathsf{Enc}_{\tau_e}\Big(\ (\ \Psi(\mathbf{k}^i_{w,\alpha,\beta}, \epsilon^i_{w,\alpha,\beta}),\ (g \| \alpha \| \beta)\ ),\ (\mathbf{k}^i_{u,\alpha}, \mathbf{k}^i_{v,\beta})\ \Big)$$
   where g is a unique gate identifier.
   (e) For every output gate g associated to a set of parties $\hat{\mathcal{P}} \subseteq \mathcal{P}$, with input wire u and output wire w, perform the following steps
      - Set $[\lambda_w] \leftarrow [\lambda_u]$.
      - For $\alpha \in \{0,1\}$, each $P_i \in \hat{\mathcal{P}}$ generates two random values $\xi^i_{w,\alpha} \in \{0,1\}^s$ and shares them as $\langle \xi^i_{w,\alpha} \rangle \leftarrow \mathsf{Input}^P(P_i)$.
      - For $\alpha \in \{0,1\}$ use the trick from step 5d above to construct the garbled row $\tilde{g}_\alpha = (C^{w,\alpha}, \mathbf{c}^{w,\alpha})$ corresponding to the encryption
      $$\mathsf{Enc}_{\tau_d}\Big(\ (\ (\xi^{i_1}_{w,\alpha} \| \ldots \| \xi^{i_{|\hat{\mathcal{P}}|}}_{w,\alpha}),\ (g \| \alpha \| 0)\ ),\ (\mathbf{k}_{u,\alpha}, \mathbf{0})\ \Big)$$
6. Reveal to each P_i their input and output wire masks: $\lambda_w \leftarrow \mathsf{Output}([\lambda_w], P_i), w \in W_{\mathsf{in}_i} \cup W_{\mathsf{out}_i}$.

Open Garbling:

1. Each P_i calls $\langle \mathbf{c}^{i,w,\alpha,\beta} \rangle \leftarrow \mathsf{Input}^P(P_i)$. All parties then computes $\langle \mathbf{c}^{w,\alpha,\beta} \rangle = \oplus_{i \in [n]} \langle \mathbf{c}^{i,w,\alpha,\beta} \rangle$ and reveal the result (using ℓ calls to Open^P) so that each party obtains the ciphertext $(C^{w,\alpha,\beta}, \mathbf{c}^{w,\alpha,\beta})$
2. The garbled gate is $\tilde{g}_{w,\alpha,\beta} = (C^{w,\alpha,\beta}, \mathbf{c}^{w,\alpha,\beta})$ for $\alpha, \beta \in \{0,1\}$.
3. Similarly, in output gates, for $\alpha \in \{0,1\}$ use the trick from step 1 in **Open Garbling** to reconstruct $\tilde{g}_{w,\alpha} = (C^{w,\alpha}, \mathbf{c}^{w,\alpha})$

[a] See Remark 1

Figure 7. The protocol for unauthenticated garbling, with n/c honest parties

5.1 Garbling

In this section we describe how to implement the $\mathcal{F}^{n/c}_{\text{Preprocessing}}$ functionality given in Figure 6. As this is a weaker functionality which allows the adversary to introduce additive errors in the garbled circuit, our implementing protocol will not need to produce the LPN ciphertexts and keys using a fully active implementation of $\mathcal{F}_{\mathsf{MPC}}$ as we did in Section 4.

The main idea of our unauthenticated garbling protocol is to use the homomorphic property of the LPN encryption scheme, i.e., abusing notation,

$$\Sigma_{i=1}^{n}\mathsf{Enc}_{\tau}^{\mathsf{XOF}}((\mathbf{m}^{i},\mathsf{nonce}),\mathbf{s}^{i}) = \mathsf{Enc}_{\tau'}^{\mathsf{XOF}}((\Sigma_{i=1}^{n}\mathbf{m}^{i},\mathsf{nonce}),\Sigma_{i=1}^{n}\mathbf{s}^{i}). \tag{8}$$

However, note that the Bernoulli distribution resulting from the sum has parameter $\tau' > \tau$. Additionally, even given only the sum of the encryptions, the adversary can use the above homomorphic property to "remove" his own encryptions and remain with only the sum of the honest parties' encryptions. Thus, the sum of the honest parties' encryptions must still be secure.

We thus proceed as follows: we let each party locally generate a 'weak' LPN encryption for the garbled gates. The garbled gates are computed by summing these 'weak' encryptions. The 'weak' ciphertexts are never seen by the adversary, as the parties compute their sum using additive secret-sharing. Intuitively, if the adversary cannot learn any information on the keys and messages from the sum, then this gives the adversary the possibility of (only) an additive attack. Hence, this scheme works as long as the sum of n 'weak' encryptions is decryptable and the sum of n/c 'weak' encryptions is secure.

We now look at how to achieve these requirements. We introduce τ_s to denote the parameter of the Bernoulli distribution that we want the sum of any n/c ciphertexts to achieve. For the local, weak encryptions, honest parties will use a parameter τ_e. Lastly, the sum of all n ciphertexts will have a Bernouilli distribution with a parameter that we will denote τ_d. Below we analyse the relationship between the three τ parameters and give an example of how to select them in practice. Our analysis makes use of the following lemma [Mat94].

Lemma 3 (Piling Up Lemma). *Let X be binary random variable which is equal to one with probability $p = 1/2 - \epsilon$, where ϵ is the bias approximation, then we have*

$$\Pr[x_1 + \cdots + x_n = 1 : x_i \leftarrow X] = \frac{1}{2} - 2^{n-1}\cdot\epsilon^n.$$

Recall we have n parties of which n/c are honest, and in our garbling protocol each honest party will generate an LPN ciphertext with τ equal to τ_e, with the adversary producing a ciphertext in any way it chooses. These ciphertexts are then secret shared, and the sum of all the n ciphertexts is then released.

As explained, the adversary can determine the sum of the n/c ciphertexts produced by the honest parties. These sum to a ciphertext whose underlying τ value, τ_s, can be evaluated by the Piling Up Lemma. Thus, we have

$$\tau_s = \frac{1}{2} - 2^{n/c-1}\cdot\left(\frac{1}{2} - \tau_e\right)^{n/c} = \frac{1}{2}\cdot\left(1 - (1 - 2\cdot\tau_e)^{n/c}\right).$$

Alternatively this gives us that

$$\tau_e = \frac{1}{2}\cdot\left(1 - (1-\tau_s)^{c/n}\right).$$

We also require that, if the adversarial parties follow the protocol, the resulting ciphertext sum can be decrypted correctly. In other words we need to set τ_d such that

$$\tau_d = \frac{1}{2} - 2^{n-1} \cdot \left(\frac{1}{2} - \tau_e\right)^n = \frac{1}{2} \cdot (1 - (1 - 2 \cdot \tau_e)^n),$$

or

$$\tau_e = \frac{1}{2} \cdot \left(1 - (1 - 2 \cdot \tau_d)^{1/n}\right).$$

Note that this gives us

$$\begin{aligned}\tau_s &= \frac{1}{2} \cdot \left(1 - \left(1 - 2 \cdot \left(\frac{1}{2} \cdot \left(1 - (1 - 2 \cdot \tau_d)^{1/n}\right)\right)\right)^{n/c}\right) \\ &= \frac{1}{2} \cdot \left(1 - \left((1 - 2 \cdot \tau_d)^{1/n}\right)^{n/c}\right) = \frac{1}{2} \cdot \left(1 - (1 - 2 \cdot \tau_d)^{1/c}\right).\end{aligned}$$

Therefore, we have proved the following fact.

Lemma 4. *Let τ_s, τ_e, τ_d be LPN parameters, as described above. For fixed τ_d the value of τ_s does not depend on the number of parties, but only on the proportion c which is honest.*

Starting with a τ_d, a desired security parameter sec and a proportion c, we can derive the LPN parameters k, τ_s and τ_e. First, using τ_d and c, it is possible to derive τ_s. Then, given sec and τ_s, we can compute k using Equation (3). Finally, τ_e, that parties use for encryption, is derived from τ_s and the number of parties n. For example, if we take $\tau_d = 1/8$ and a proportion of 20% honest parties, i.e. $c = 5$, then we find that $\tau_s = 0.02796$. For $\mathsf{sec} = 128$ this implies we need to select $k = 3129$. For $n = 100$ parties we then have that the honest parties need to encrypt with parameter $\tau_e = 0.001436$. For more examples, consider Table 3 (in Section 6) for $\mathsf{sec} = \{128, 256\}$.

Using the above observations we define, in Figure 7, the garbling protocol when n/c parties are honest. Our protocol makes use of an operation, which allows us to compute an unauthenticated sharing of $\langle x \cdot \Delta \rangle$ given an authenticated sharing of a bit $[x]$, where $\Delta \in \{0, 1\}^k$ is a global shared value. We denote this operation by

$$\langle x \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([x]).$$

We could naïvely implement this operation using Tiny-OT, but this would be highly inefficient since $\Delta \in \mathbb{F}_2^k$ and k is very large as it is the dimension of the secret key space $\mathcal{K}$ of the underlying LPN encryption scheme. For this reason, in Section 7, we show a more efficient bit-string multiplication protocol, that is still based on Tiny-OT. The new protocol requires that $n/c \geq s$, where s is the statistical security parameter. Since c is a constant, this requirement holds for sufficiently large n.[7] We describe in Figure 13 the ideal functionality $\mathcal{F}_{\mathrm{BS}}$ implemented by our $\mathsf{Bit} \times \mathsf{String}$ protocol.

Remark 1. Note that the way that the $\mathsf{Bit} \times \mathsf{String}$ operation is described in Section 7, the shares of Δ are chosen inside the $\mathsf{Bit} \times \mathsf{String}$ protocol. However, this would make the unauthenticated garbling protocol description in Figure 7 cumbersome. To simplify the presentation, we let the parties choose their shares of Δ at the beginning of the unauthenticated garbling protocol; this is possible since the Δ shares are used only locally before the $\mathsf{Bit} \times \mathsf{String}$ operation.

Protocol $\Pi^{n/c}_{\mathsf{Evaluate}}$

Inputs: Each party P_i has input $\{\rho_w\}_{w \in W_{\mathsf{in}_i}}$.

Preprocessing: This sub-task is performed as follows.

- Call **Garbling** on $\mathcal{F}^{n/c}_{\text{Preprocessing}}$ with input C_f.
- For every wire $w \in W_{\mathsf{in}_i}$, P_i obtains the wire mask λ_w. For every $w \in W_{\mathsf{in}} = \{W_{\mathsf{in}_j}\}_{j \in [n]}$, P_i obtains the key shares $\{\mathbf{k}^i_{w,\alpha}\}_{w \in W_{\mathsf{in}}, \alpha \in \{0,1\}}$.
- For every $w \in W_{\mathsf{out}_i}$, P_i obtains the wire mask λ_w and the guard values $\{\xi^i_{w,\alpha}\}_{\alpha \in \{0,1\}}$.

Online Computation: This sub-task is performed as follows.

1. $\forall\ w \in W_{\mathsf{in}_i}$, P_i broadcasts its masked input as $\epsilon_w \leftarrow \rho_w \oplus \lambda_w$.
2. Each P_i broadcasts its corresponding key share $\mathbf{k}^i_{w,\epsilon_w}$ $\forall\ w \in W_{\mathsf{in}}$.
3. Parties call **Open Garbling** on $\mathcal{F}^{n/c}_{\text{Preprocessing}}$
4. Each party sets, for every input wire, $\mathbf{k}_{w,\epsilon_w} \leftarrow \bigoplus_{i \in [n]} \mathbf{k}^i_{w,\epsilon_w}$.
5. Each P_i evaluates the garbled circuit as follows:
   - For all XOR gates parties locally compute the public values via $\epsilon_w \leftarrow \epsilon_u \oplus \epsilon_v$ and the key values via $\mathbf{k}_{w,\epsilon_w} \leftarrow \mathbf{k}_{u,\epsilon_u} \oplus \mathbf{k}_{v,\epsilon_v}$.
   - For AND gates parties perform the following steps:
     - Decrypt the ciphertext $(C^{w,\epsilon_u,\epsilon_v}, \mathbf{c}^{w,\epsilon_u,\epsilon_v})$ using the key $(\mathbf{k}_{u,\epsilon_u}, \mathbf{k}_{v,\epsilon_v})$ and denote the result as $\mathbf{m}$.
     - Invert Ψ on $\mathbf{m}$ to obtain $(\mathbf{k}_{w,\epsilon_w}, \epsilon_w)$, aborting if $\mathbf{m} \notin Im(\Psi)$.
   - If an output gate is associated with P_i, do as follows:
     - Decrypt the ciphertext $(C^{w,\epsilon_u}, \mathbf{c}^{w,\epsilon_u})$ using the key $\mathbf{k}_{u,\epsilon_u}$ and denote the result as $\mathbf{m}$.
     - Look for the string ξ^i_{w,ϵ_u} on its corresponding position in $\mathbf{m}$. If any different string is found in that position, P_i aborts and notifies every other party. Otherwise, P_i sets this output to be $\epsilon_u \oplus \lambda_w$.

Figure 8. The protocol for evaluation in the unauthenticated garbling case $\Pi^{n/c}_{\mathsf{Evaluate}}$

Compared with the evaluation phase of [HSS17], we cannot rely on individual pairs of keys, $\mathbf{k}^i_{w,0}, \mathbf{k}^i_{w,1}$, in order to let a party P_i decide whether to abort or not in the presence of errors in the garbled circuit. This is because only the sums of individual keys, $\mathbf{k}_{w,0}, \mathbf{k}_{w,1}$ are revealed, and these need to be hidden from all parties. Instead, we perform a check in the output gates as follows: given a set of parties $\hat{\mathcal{P}} \subseteq \mathcal{P}$ who receive an output of C_f on wire w, a garbled output gate g, with input wire u and output wire w, consists of the two following entries (one for each $\alpha \in \{0,1\}$):

$$g_\alpha \leftarrow \mathsf{Enc}^{\mathsf{XOF}}_\tau\Big(\ (\ (\xi^{i_1}_{w,\alpha} \| \dots \| \xi^{i_{|\hat{\mathcal{P}}|}}_{w,\alpha}),\ (g\|\alpha\|0)\ ),\ (\mathbf{k}_{u,\alpha}, \mathbf{0})\ \Big)$$

where $\xi^i_{w,\alpha} \in \{0,1\}^s$ is a secret random value chosen by party P_i.[8]

The security of our garbling protocol is then given by the following theorem.

Theorem 5. *Let $\mathcal{E}^{\mathsf{XOF}}_\tau$ be a XOF-based two-key LPN encryption scheme with parameter τ. Let $\mathcal{F}_{\mathrm{BS}}$ (Figure 6) be implemented by the* $\mathsf{Bit} \times \mathsf{String}$ *operation. The protocol $\Pi^{n/c}_{\mathsf{Garble}}$ described in Figure 7 UC-securely computes $\mathcal{F}^{n/c}_{\mathrm{Preprocessing}}$ (Figure 6) in the presence of a static, active adversary corrupting up to $(c-1)\cdot n/c$ parties in the $\{\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathrm{BS}}\}$-hybrid model, provided $n/c > s$ (where s is the statistical security parameter).*

Remark 2. By implementing the $\mathsf{Bit} \times \mathsf{String}$ operation in the naïve way, using TinyOT as in [HSS17], we could prove Theorem 5 in the $\{\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathrm{TinyOT}}\}$-hybrid model, without the $n/c > s$ requirement.

Proof. We describe a simulator $\mathcal{S}$ such that for any adversary $\mathcal{A}$ who corrupts a subset $I \subsetneq \mathcal{P}$ of parties, and any environment $\mathcal{Z}$ that chooses parties' inputs and sees all outputs, $\mathcal{Z}$ cannot distinguish between a real execution of the protocol between $\mathcal{A}$ and honest parties or a simulated execution of the protocol between $\mathcal{S}$ and the ideal functionality $\mathcal{F}^{n/c}_{\mathrm{Preprocessing}}$.

Description of the simulator $\mathcal{S}$. This simulator is very similar to the one described in the proof of Theorem 3.

The main difference is that here the garbled table per AND gates are not computed through $\mathcal{F}_{\mathsf{MPC}}$. Instead the simulator first emulates $\mathcal{F}_{\mathsf{MPC}}$ to provide the adversary with its shares of $[\epsilon_{w,\alpha,\beta}]$. Then, the simulator proceeds to emulate $\mathcal{F}_{\mathrm{BS}}$, receiving from the adversary its shares of Δ and a subset $S \subset L \times \mathcal{H}$ and corresponding errors $0 \neq \mathtt{E}_{\ell,j} \in \{0,1\}^{\mathrm{sec}}$ for $(\ell, j) \in S$.

The simulator chooses random elements $\chi^j_\ell \in \{0,1\}^{\mathrm{sec}}$ for $(\ell, j) \in S$ and gives them to the adversary. The adversary needs to guess $\bigoplus_{(\ell,j)\in S} b^j_\ell \cdot \chi^j_\ell \cdot \mathtt{E}_{\ell,j}$. After receiving the adversary's guess, the simulator sends it back the (correct) value $\bigoplus_{(\ell,j)\in S} b^j_\ell \cdot \chi^j_\ell \cdot \mathtt{E}_{\ell,j}$, and proceeds as follows.

If the adversary $\mathcal{A}$ guessed incorrectly, then the simulator sends an `abort` message to the functionality. If the adversary guessed correctly, then we consider two cases: (i) If $|S| \geq n/c > s$, then the simulator halts (admitting failure). We note that this event occurs with probability at most 2^{-s}. (ii) Otherwise, the simulator continues to provide the adversary with its output (i.e., gives the adversary its shares of $b_\ell \cdot \Delta$). In this case, the simulator also computes the additive error to send to the functionality in the Open Garbling phase. These errors are computed by the noise

[7] If the requirement does not hold, then this operation needs to be done using Tiny-OT directly as in [HSS17]. Hence, this optimization is mainly for large-scale MPC.

[8] For simplicity, we assume the message space is at least $|\hat{\mathcal{P}}| \cdot s$ bits long. If the message space was only of $|\hat{\mathcal{P}}| \cdot s/r$ bits, one would compute r ciphertext, each of them with the ξ^i values of $|\hat{\mathcal{P}}|/r$ parties.

sent by the adversary $\mathcal{A}$ to the simulator, together with the shares b^j_ℓ chosen for the honest parties by the simulator.

Note that the simulator can compute and store the corrupt parties' shares of the garbled circuit $\tilde{g}_{w,\alpha,\beta}$, for $i \in I$, as it has all the necessary values to compute these values from the messages it received in the previous steps (emulating $\mathcal{F}_{\mathsf{MPC}}$ and $\mathcal{F}_{\mathrm{BS}}$ for the adversary $\mathcal{A}$). After this, $\mathcal{S}$ sends to $\mathcal{A}$ honest shares of $\tilde{g}_{w,\alpha,\beta}$ by emulating the **Open** command in $\mathcal{F}_{\mathsf{MPC}}$. If $\mathcal{A}$ sends abort, it forwards abort to the functionality, otherwise receives corrupt shares of the garbled circuit.

Finally, $\mathcal{S}$ simulates the final check as follows: run **Input** emulating $\mathcal{F}_{\mathsf{MPC}}$ with the input provided by $\mathcal{A}$, and repeat the same steps as for garbling AND gates, to compute the garbled rows $\tilde{g}_\alpha$, $\alpha \in \{0,1\}$. After this, it receives the output wire masks λ_w from $\mathcal{F}^{n/c}_{\mathrm{Preprocessing}}$ and forward these values to the $\mathcal{A}$. Hence it forwards to the functionality abort, if $\mathcal{A}$ sends abort and terminates. This concludes the simulation.

Indistinguishability. It follows by inspection that given that the adversary did not guess the challenge correctly with $|S| \geq n/c$, ideal and real executions are indistinguishable. In particular, the honest parties' shares are identically distributed in both worlds because they are either uniformly random or cithertexts obtained by the LPN-based encryption scheme $\mathcal{E}$. Since the above event occurs with probability at most 2^{-s}, the theorem follows. □

5.2 Evaluation

We can now give the evaluation procedure in Figure 8. This involves no operations with respect to the MPC functionality, but it requires two rounds of broadcast. The security of our evaluation protocol is given by the following Theorem.

Theorem 6. *Let f be an n-party functionality and $\mathcal{E}^{\mathsf{XOF}}_\tau$ a XOF-based two-key LPN encryption scheme with parameter τ. The protocol $\Pi^{n/c}_{\mathsf{Evaluate}}$ described in Figure 8 UC-securely computes f in the presence of a static, active adversary corrupting up to $(c-1)\cdot n/c$ parties in the $\{\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}^{n/c}_{\mathrm{Preprocessing}}\}$-hybrid model.*

Proof. Along the proof, we let $W_\wedge \subset W$ denote the set of output wires of AND gates (set $C_\wedge$) in the boolean circuit C_f. In the same fashion, let $W_{\mathsf{out}} \subset W$ denote the set of circuit-output wires, i.e., the output wires of the output gates (set C_{out}).

Let $\mathcal{A}$ be a PPT adversary corrupting a subset of parties $I \subset [n]$, and let $\bar{I}$ denote the set of honest parties. We prove that there exists a PPT simulator $\mathcal{S}$ with access to an ideal functionality $\mathcal{F}$ that implements f, that simulates the adversary's view.

Description of the simulator $\mathcal{S}$.

1. Initialization. $\mathcal{S}$ incorporates the adversary $\mathcal{A}$, who controls the set of corrupt parties I, and internally emulates an execution of the honest parties running $\Pi^{n/c}_{\mathsf{Evaluate}}$ with $\mathcal{A}$:
2. Garbling. $\mathcal{S}$ emulates the garbling phase of functionality $\mathcal{F}^{n/c}_{\mathrm{Preprocessing}}$ on input $(\mathtt{Garbling}, C_f)$ from the adversary as follows:
   – For every $w \in \{W_{\mathsf{in}_i}\}_{i\in I}$, $\mathcal{S}$ receives from $\mathcal{A}$ a share of the input-wire key $\mathbf{k}^i_{w,0} \in \{0,1\}^{\mathsf{sec}}$ and a wire mask $\lambda_w \in \{0,1\}$.

– For each output gate $g \in C_{\mathsf{out}}$, with input wire u and output wire w, $\mathcal{S}$ samples a random mask $\lambda_w \in \{0,1\}$ and sets $\lambda_u = \lambda_w$. Additionally, for corrupted parties $i \in I$ with output wires $w \in W_{\mathsf{out}_i}$, it receives from $\mathcal{A}$ the guard values $\xi^i_{w,0}$ and $\xi^i_{w,1}$. For honest parties $i \in \bar{I}$ with output wires $w \in W_{\mathsf{out}_i}$, it samples at random the guard values $\xi^i_{w,0}$ and $\xi^i_{w,1}$.
– Upon receiving `OK` from $\mathcal{A}$, send λ_w to $\mathcal{A}$ for each $w \in \{W_{\mathsf{out}_i}\}_{i \in I}$.

3. Online computation. The simulator interacts with $\mathcal{A}$ in the online phase and generates a simulated garbled circuit, as follows.
   – For each $w \in \{W_{\mathsf{in}_i}\}_{i \in \bar{I}}$, $\mathcal{S}$ samples a random public value $\epsilon_w \in \{0,1\}$ and sends this to $\mathcal{A}$.
   – For each $w \in \{W_{\mathsf{in}_i}\}_{i \in I}$, $\mathcal{S}$ receives from $\mathcal{A}$ a public value ϵ_w and computes $x_w = \epsilon_w \oplus \lambda_w$. $\mathcal{S}$ sends the extracted inputs $\{x_w\}_{i \in I, w \in W_{\mathsf{in}_i}}$ to the ideal functionality computing f, receiving the output $\mathbf{y} = \{y_w\}_{w \in W_{\mathsf{out}}}$.
   – $\mathcal{S}$ samples at random the honest parties' shares of the input wire keys $\{k^i_{w,\epsilon_w}\}_{i \in \bar{I}, w \in W_{\mathsf{in}}}$, and sends these to $\mathcal{A}$. It receives back the adversary's shares of the input keys $\{\hat{k}^i_{w,\epsilon_w}\}_{i \in I, w \in W_{\mathsf{in}}}$ (which may be different to the shares received in the garbling phase).
   – Active path: External values. In this step the simulator generates the remaining external values for the active path that parties will follow when evaluating the garbled circuit. Let $W_{\mathsf{final}-\wedge} \subset W_\wedge$ denote the output wires of AND gates g where no successor of g is an AND gate.
     • For each $w \in W_{\mathsf{out}}$, let $\epsilon_w = y_w \oplus \lambda_w$ and $\epsilon_u = \epsilon_w$.
     • For each $w \in W_\wedge \setminus W_{\mathsf{final}-\wedge}$, sample $\epsilon_w \in \{0,1\}$.
     • For all wires $w \in W_{\mathsf{final}-\wedge}$ which are not the input of an output gate, sample ϵ_w at random subject to the constraint that these form a satisfying assignment to the circuit with outputs $\{\epsilon_w\}_{w \in W_{\mathsf{out}}}$.[9]
   – Active path: Ciphertext generation. In this step the simulator computes the sequence of keys and ciphertexts that will be observed by the adversary when following the external values of the active path.
     • AND gates: For every $g \in \mathcal{C}_\wedge$, with input wires (u,v) and output wire w, let $\mathcal{S}$ honestly generate the garbled row (ϵ_u, ϵ_v) by sampling $\mathbf{k}_{w,\epsilon_w} \leftarrow \{0,1\}^k$ and computing:

$$\tilde{g}_{\epsilon_u,\epsilon_v} \leftarrow \mathsf{Enc}_\mathcal{T}(\ (\Psi(\mathbf{k}_{w,\epsilon_w}, \epsilon_w), (g \| \epsilon_u \| \epsilon_v),\ (\mathbf{k}_{u,\epsilon_u}, \mathbf{k}_{v,\epsilon_v})\ )$$

       For $(\alpha,\beta) \neq (\epsilon_u,\epsilon_v)$, set the three remaining garbled rows as $\tilde{g}_{\alpha,\beta} = (H(g\|\alpha\|\beta), c_{\alpha,\beta})$, where $c_{\alpha,\beta} \leftarrow \{0,1\}^\ell$.
     • XOR gates: For every XOR gate with input wires (u,v) and output wire w, $\mathcal{S}$ sets $\mathbf{k}_{w,\epsilon_w} = \mathbf{k}_{u,\epsilon_u} \oplus \mathbf{k}_{v,\epsilon_v}$.
     • Output gates: For every $g \in C_{\mathsf{out}}$, with input wire u and output wire w, let $\mathcal{S}$ honestly generate the garbled row ϵ_u by computing:

$$\tilde{g}_{\epsilon_u} \leftarrow \mathsf{Enc}_{\mathcal{T}_d}\Big(\ (\ (\xi^{i_1}_{w,\epsilon_u} \| \dots \| \xi^{i_{|\mathcal{P}|}}_{w,\epsilon_u}),\ (g\|\epsilon_u\|0)\ ),\ (\mathbf{k}_{u,\epsilon_u}, \mathbf{0})\ \Big)$$

       For the inactive garbled row $\bar{\epsilon}_u$, set $\tilde{g}_{\bar{\epsilon}_u} = (H(g\|\bar{\epsilon}_u\|0), c_{\bar{\epsilon}_u})$, where $\mathbf{c}_{\bar{\epsilon}_u} \leftarrow \{0,1\}^\ell$.
   Importantly, notice that in the description above $\mathcal{S}$ never uses any inactive key $\mathbf{k}_{w,\bar{\epsilon}_w}$ in order to generate any of the garbled gates.

[9] This is needed to ensure that, for instance, if an output wire w comes from the XOR of two previous AND gates with output wires (u,v), then the public values ϵ_u, ϵ_v are chosen to satisfy $\epsilon_w = \epsilon_u \oplus \epsilon_v$, as required.

The Flip event

Flip happens whenever $\mathcal{A}$ succeeds in making honest parties deviate from the active path without aborting and, additionally, any of the two following events occur:

(F1) There exists an AND gate $g \in C_\wedge$ such that honest evaluating parties obtain the inactive wire key $\hat{\mathbf{k}}_{w,\bar{\epsilon}_w}$ for the output wire w of g.

(F2) There exists an output gate g with input wire u and output wire w such that at least one honest party P_i with $w \in W_{\mathsf{out}_i}$, when evaluating the garbled circuit, obtains guard value $\xi^i_{w,\bar{\epsilon}_u}$.

Figure 9. The Flip event that would break security.

– The simulator hands the adversary the complete garbled circuit. In case the adversary aborts, the simulator sends $\bot$ to the ideal functionality and aborts. Otherwise, the simulator obtains additive errors $\mathbf{e} = \{\{e_g^{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\},g\in C_\wedge}, \{e_g^{\alpha}\}_{\alpha\in\{0,1\},g\in C_{\mathsf{out}}}\}$ and computes the modified garbled circuit as $\tilde{g}_{\alpha,\beta} \oplus e_g^{\alpha,\beta}$ for each $g \in C_\wedge$ and $\alpha, \beta \in \{0,1\}$ and $\tilde{g}_\alpha \oplus e_g^\alpha$ for each $g \in C_{\mathsf{out}}$ and $\alpha \in \{0,1\}$.

– Fail or not fail: Finally, the simulator evaluates the modified circuit using the input wire keys $\hat{\mathbf{k}}_{w,\epsilon_w} = \bigoplus_{i\in I} \hat{\mathbf{k}}^i_{w,\epsilon_w} \oplus \bigoplus_{j\in\bar{I}} \mathbf{k}^j_{w,\epsilon_w}$ for $w \in W_{\mathsf{in}}$ and checks whether the honest parties would have aborted. Namely, for every $w \in \{W_{\mathsf{out}_i}\}_{i\in\bar{I}}$, it determines whether any honest P_i will find or not the guard value ξ^i_{w,ϵ_u} it is expecting on that wire,using the key $\hat{\mathbf{k}}_{w,\epsilon_w}$. If this check fails, then the simulator outputs fail and aborts.

Indistinguishability. Let $\mathbf{HYB}^{\mathcal{F}^{n/c}_{\text{Preprocessing}}}_{\Pi^{n/c}_{\mathsf{Evaluate}},\mathcal{A},\mathcal{Z}}(1^{\mathsf{sec}}, z)$ denote the output distribution of the adversary $\mathcal{A}$ and honest parties in a real execution with $\Pi^{n/c}_{\mathsf{Evaluate}}$ and let $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^{\mathsf{sec}}, z)$ denote the output distribution of $\mathcal{S}$ and the honest parties in an ideal execution. We now prove that $\mathbf{HYB}^{\mathcal{F}^{n/c}_{\text{Preprocessing}}}_{\Pi^{n/c}_{\mathsf{Evaluate}},\mathcal{A},\mathcal{Z}}(1^{\mathsf{sec}}, z) \overset{c}{\equiv} \mathbf{IDEAL}_{\mathcal{F}^{n/c}_{\text{Preprocessing}},\mathcal{S},\mathcal{Z}}(1^{\mathsf{sec}}, z)$.

We start by considering an execution $\widetilde{\mathbf{IDEAL}}$ where a simulator $\tilde{\mathcal{S}}$ produces a view that is identical to the view produced by $\mathcal{S}$ in $\mathbf{IDEAL}$. Namely, the adversary's view is simulated exactly as in $\mathbf{IDEAL}$ by a simulator $\tilde{\mathcal{S}}$ with the exception that $\tilde{\mathcal{S}}$ further picks the global difference Δ and computes inactive keys $\mathbf{k}_{w,\bar{\epsilon}_w} = \mathbf{k}_{w,\epsilon_w} \oplus \Delta$ for $w \in W \setminus W_{\mathsf{out}}$ (which are never defined by $\mathcal{S}$). Moreover, the event for which $\tilde{\mathcal{S}}$ outputs fail and aborts is modified to also include whenever the Flip event happens (as described in Figure 9). Note that this event is well-defined, since so are all the wire keys in $\widetilde{\mathbf{IDEAL}}$. Hence, the only difference between $\mathbf{IDEAL}$ and $\widetilde{\mathbf{IDEAL}}$ is the event that $\mathcal{S}$ aborts whereas $\tilde{\mathcal{S}}$ does not abort, i.e. when Flip happens. We next show (Lemma 5) that this occurs at most with probability 2^{-s}, which implies that $\mathbf{IDEAL}$ and $\widetilde{\mathbf{IDEAL}}$ are indistinguishable.

The rest of the proof consists in showing that in $\mathbf{HYB}$ (Lemma 7), Flip would also occur only with negligible probability. Then, we show that as long as Flip does not happen, no environment can distinguish between the ideal and real worlds by reducing to the LIN RK-KDM$^\sigma$ security of $(\mathsf{KeyGen}^{\mathsf{XOF}}, \mathsf{Enc}^{\mathsf{XOF}}_\tau, \mathsf{Dec}^{\mathsf{XOF}})$ (Lemma 6).

Lemma 5. *The probability that* Flip *occurs in* $\widetilde{\mathbf{IDEAL}}$ *is no more than* 2^{-s}*.*

Proof. Recall first that the simulated garbling in $\mathbf{IDEAL}$ involves only generating a single key $\mathbf{k}_{w,\epsilon_w}$ per wire, and thus the simulator does not even need to choose a global difference Δ in order to complete the garbling. The simulator in $\widetilde{\mathbf{IDEAL}}$ does generate this extra value, but it never uses Δ for garbling the circuit. In other words, the simulated garbled circuit given to $\mathcal{A}$ is completely independent of Δ.

First, we prove that Flip can only happen with negligible probability.

Condition (F1): First, consider the problem with the two following simplifying assumptions:

(F11) The external values on g's input wires u and v obtained by honest parties are the right ones, i.e. ϵ_u and ϵ_v.

(F12) The keys for g's input wires u and v obtained by honest parties are the right ones, i.e. $\mathbf{k}_{u,\epsilon_u}$ and $\mathbf{k}_{v,\epsilon_v}$.

Let $b_w \in \{0,1\}$ be a bit of $\mathcal{A}$'s choice, which decides whether he wants to change ϵ_w to $\bar{\epsilon}_w$ or not. Then, the active row of the garbled gate has to be

$$\hat{g}_{\epsilon_u,\epsilon_v} = \mathsf{Enc}_{\tau_d}\left(\ \left(\Psi(\ \mathbf{k}_{w,0} \oplus \bar{\epsilon}_w \cdot \Delta,\ \epsilon_w \oplus b_w\ ),\ (g||\epsilon_u||\epsilon_v)\ \right),\ (\mathbf{k}_{u,\epsilon_u}, \mathbf{k}_{v,\epsilon_v})\ \right),$$

for which the adversary needs to introduce an error $e_g^{\epsilon_u,\epsilon_v}$ into the original row $\tilde{g}_{\epsilon_u,\epsilon_v}$ of the garbled gate given by:

$$e_g^{\epsilon_u,\epsilon_v} = \hat{g}_{\epsilon_u,\epsilon_v} \oplus \tilde{g}_{\epsilon_u,\epsilon_v} = G \cdot \Psi(\Delta, b_w) \oplus \tilde{\mathbf{e}} \oplus \mathbf{e},$$

where G is the encoding matrix of the error correcting code associated with Enc_τ, $\tilde{\mathbf{e}}$ is the original LPN error sampled to produce the ciphertext $\tilde{g}_{\epsilon_u,\epsilon_v}$ and $\mathbf{e}$ is a possible additional adversarially chosen error. We can rewrite $e_g^{\epsilon_u,\epsilon_v}$ as $\mathbf{v} = G \cdot \hat{\mathbf{m}} \oplus E$, where $E = \tilde{\mathbf{e}} \oplus \mathbf{e}$ and $\hat{\mathbf{m}} = \Psi(\Delta, b_w)$. We can see that the adversary can:

- Either guess Δ, which happens with probability 2^{-k+2}
- Or guess one of the value $\mathbf{v}_{\hat{\mathbf{m}}} = G\hat{\mathbf{m}} + E$, where $\hat{\mathbf{m}}$ is fixed and unknown, and such that $\mathsf{Decode}(G \cdot \hat{\mathbf{m}} + E) = \hat{\mathbf{m}}$. If Decode is a minimum decoding algorithm and L is the $[\ell, \mathfrak{m}, d]$ linear code used in the encryption scheme, we can bound the number of vectors that satisfy this relation by $\sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor = t} \binom{\ell}{i} = V_2(\ell, t)$. Since $2^{\mathfrak{m}} \cdot V_2(\ell, t) \leq 2^\ell$, we have that the probability of a successful guess is bounded by $2^{-\mathfrak{m}}$. We assume $\mathfrak{m} \geq k > \mathsf{sec}$, thus this probability is negligible. More generally, given a linear code L that is (ℓ, τ)-correcting in the LPN encryption scheme, we have that $V_2(\ell, \tau\ell)/2^\ell \leq 2^{\eta\ell}$, where $\eta = (H_2(\tau) - 1)$ and $H_2(\cdot)$ is the binary entropy function. So this probability is negligible for any reasonable choice of L in the LPN-based encryption scheme.

We can now look at the problem without its simplifying assumptions.

If we remove (F11), i.e. if we allow for external values $(\epsilon_u \oplus b_u, \epsilon_v \oplus b_v)$ where $(b_u, b_v) \neq (0,0)$, we know from the simulation that the row parties will try to decrypt is the ciphertext

$$\hat{g}_{\epsilon_u \oplus b_u, \epsilon_v \oplus b_v} = (H(g||\epsilon_u \oplus b_u||\epsilon_v \oplus b_v),\ c_{\epsilon_u \oplus b_u, \epsilon_v \oplus b_v} \oplus e_g^{\epsilon_u \oplus b_u, \epsilon_v \oplus b_v}),$$

where $c_{\epsilon_u \oplus b_u, \epsilon_v \oplus b_v} \leftarrow \{0,1\}^\ell$. It is easy to see that, after 'decryption' of the uniformly random value $c_{\epsilon_u \oplus b_u, \epsilon_v \oplus b_v} \oplus e_g^{\epsilon_u \oplus b_u, \epsilon_v \oplus b_v}$, parties would obtain exactly the inactive key $\mathbf{k}_{w,\bar{\epsilon}_w} = \mathbf{k}_{w,\epsilon_w} \oplus \Delta \in \{0,1\}^k$ sampled by $\tilde{\mathcal{S}}$ with probability at most 2^{-k}. This is irrespective of whether we also remove (F12) or not.

If we only remove (F12), i.e. if we allow for any pair of keys $\mathbf{k}_{u,\epsilon_u} + e_u, \mathbf{k}_{u,\epsilon_v} + e_v$ but parties hold ϵ_u, ϵ_v, it follows that the error the adversary needs to introduce in the original garbled row would now be of

$$e_g^{\epsilon_u,\epsilon_v} = \hat{g}_{\epsilon_u,\epsilon_v} \oplus \tilde{g}_{\epsilon_u,\epsilon_v} = G \cdot \Psi(\Delta, b_w) \oplus H(g||\epsilon_u||\epsilon_v) \cdot (e_u \oplus \sigma(e_v))$$

which does not help $\mathcal{A}$ in comparison with the simple scenario, as he is just introducing a further additive error of $H(g||\epsilon_u||\epsilon_v) \cdot (e_u \oplus \sigma(e_v))$.

Condition (F2): This case can be further broken down into two. In the first one, we assume that honest parties hold the right external value ϵ_u for the input wire u of the output gate. Applying the same reasoning as for Condition (F1) with the assumption (F11), the only chance for $\mathcal{A}$ is to guess the honest parties' guard values he tries to flip.

In the second case, $\mathcal{A}$ has introduced an error in the circuit which wrongly convinces honest parties that the external value for the input wire u of the output gate is $\bar{\epsilon}_u$. We show that $\mathcal{A}$ cannot furthermore change the honest guard values with noticeable probability. It is easy to see we have two different cases:

Parties hold $\mathbf{k}_{u,\bar{\epsilon}_u}$: This implies Condition (F1) of Flip has been satisfied, which only happens with probability $2^{-k+2} < 2^{-s}$ as we have just shown.

Parties hold $\mathbf{k}_{u,\bar{\epsilon}_u} + e_u$, where $e_u \neq 0$: In this case, we know from the simulation that the row parties will try to decrypt is $\hat{g}_{\bar{\epsilon}_u} = (H(g\|\bar{\epsilon}_u\|0), c_{\bar{\epsilon}_u} + e_g^{\bar{\epsilon}_u})$, where $c_{\bar{\epsilon}_u} \leftarrow \{0,1\}^\ell$. It is easy to see that, after 'decryption' of the uniformly random value $c_{\bar{\epsilon}_u} + e_g^{\bar{\epsilon}_u}$, P_i would obtain the exact string $\xi^i_{w,\bar{\epsilon}_u} \in \{0,1\}^s$ in the right position with probability at most 2^{-s}.

This concludes the proof of the lemma. □

We prove that the ideal and real (hybrid) executions are indistinguishable, conditioned on the event Flip not occurring. Recall that the difference between both executions are that in $\widetilde{\mathbf{IDEAL}}$, the three inactive rows of every gate are uniformly random, whereas in $\mathbf{HYB}$ they are computed according to $\mathcal{F}^{n/c}_{\text{Preprocessing}}$.

Lemma 6. *Conditioned on the event $\overline{\mathsf{Flip}}$, the following two distributions are computationally indistinguishable:*

- $\{\mathbf{HYB}^{\mathcal{F}^{n/c}_{\text{Preprocessing}}}_{\Pi^{n/c}_{\mathsf{Evaluate}},\mathcal{A},\mathcal{Z}}(1^{\mathsf{sec}}, z)\}_{\mathsf{sec}\in\mathbb{N},z\in\{0,1\}^*}$
- $\{\widetilde{\mathbf{IDEAL}}_{\mathcal{F},\tilde{\mathcal{S}},\mathcal{Z}}(1^{\mathsf{sec}}, z)\}_{\mathsf{sec}\in\mathbb{N},z\in\{0,1\}^*}$

Proof. We begin by defining a slightly modified experiment $\widetilde{\mathbf{HYB}}$, which we need in order to prove the above result by contradiction. More concretely, we want to construct a distinguisher for the LIN RK-KDM$^\sigma$ security game of ($\mathsf{KeyGen}^{\mathsf{XOF}}$, $\mathsf{Enc}^{\mathsf{XOF}}_\tau$, $\mathsf{Dec}^{\mathsf{XOF}}$). $\widetilde{\mathbf{HYB}}$ is identically distributed to the real execution $\mathbf{HYB}$, but differs in two ways. Firstly, we move the creation of the garbled circuit from the preprocessing stage to the online computation stage, after the parties have broadcast their masked inputs. Secondly, we modify the creation of the wire keys so that on receiving the parties' inputs $\{x_w\}_{w\in W_{\mathsf{in}}}$, we first evaluate the circuit C_f, computing the actual bit ℓ_w to be transferred through each $w \in W$, where W is the set of wires of C_f. We then choose, for all $i \in \bar{I}$, two key shares $\mathbf{k}^i_{w,0}, \mathbf{k}^i_{w,1}$ for $w \in W_{\mathsf{in}}$ and a random bit λ_w for $w \in W_{\mathsf{in}_i} \cup W_{\mathsf{out}_i}$. The rest of this hybrid is identical to the real execution. Note that the garbled circuit is still computed according to $\mathcal{F}^{n/c}_{\text{Preprocessing}}$, and the rest of the protocol is identical to $\mathbf{HYB}$, which induces the same view for the adversary.

Let $\widetilde{\mathbf{HYB}}^{\mathcal{F}^{n/c}_{\text{Preprocessing}}}_{\Pi^{n/c}_{\mathsf{Evaluate}},\mathcal{A}}(1^{\mathsf{sec}}, z)$ denote the output distribution of the adversary $\mathcal{A}$ and honest parties. Assume by contradiction the existence of an environment $\mathcal{Z}$, an adversary $\mathcal{A}$ and a non-negligible function $p(\cdot)$ such that

$$\big|\Pr[\mathcal{Z}(\widetilde{\mathbf{HYB}}^{\mathcal{F}^{n/c}_{\text{Preprocessing}}}_{\Pi^{n/c}_{\mathsf{Evaluate}},\mathcal{A},\mathcal{Z}}(1^{\mathsf{sec}},z))=1]-\Pr[\mathcal{Z}(\widetilde{\mathbf{IDEAL}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^{\mathsf{sec}},z))=1]\big|\geq\frac{1}{p(\mathsf{sec})}$$

for infinitely many sec's. We show how to then construct a distinguisher $\mathcal{D}'$ which would break the LIN RK-KDM$^{\sigma}$ security of LPN, which would be in contradiction with Theorem 2. Distinguisher $\mathcal{D}'$ receives the environment's input z and internally invokes $\mathcal{Z}$ and simulator $\mathcal{S}$, playing the role of functionality $\mathcal{F}_{\mathsf{MPC}}$. It interacts with an oracle $\mathcal{O}$ which implements either $\mathsf{Real}^{\sigma}_{\Delta}$ or $\mathsf{Fake}^{\sigma}_{\Delta}$, for a $\Delta=(1,\Delta',0)$ such that $\Delta'\in\{0,1\}^{k-2}$ is secret. $\mathcal{D}'$ runs the following algorithm:

- It receives from $\mathcal{Z}$ the honest parties' inputs $\{x_w\}_{i\in\bar{I},w\in W_{\mathsf{in}_i}}$.
- $\mathcal{D}'$ emulates the communication with the adversary (controlled by $\mathcal{Z}$) in the initialization, preprocessing and garbling steps as in the simulation with $\tilde{\mathcal{S}}$.
- For each wire u, let $\ell_u\in\{0,1\}$ be the actual value on wire u. Note that these values, as well as the output of the computation y, can be determined since $\mathcal{D}'$ knows the actual input of all parties to the circuit (where the adversary's input is extracted as in the simulation with $\tilde{\mathcal{S}}$).
- It next constructs the garbled circuit as follows. For each wire $w\in W_{\wedge}$, it computes the public value ϵ_w and samples the keys $\mathbf{k}_{w,\epsilon_w}$ as $\tilde{\mathcal{S}}$ would. Using the internal values ℓ_w, we can also compute the masks $\lambda_w=\ell_w\oplus\epsilon_w$.
- For each wire that is the output of an XOR gate with input wires u and v and output wire w, the distinguisher sets $\epsilon_w=\epsilon_u\oplus\epsilon_v$ and $\mathbf{k}_{w,\epsilon_w}=\mathbf{k}_{u,\epsilon_u}\oplus\mathbf{k}_{v,\epsilon_v}$.
- For each gate $g\in C_{\wedge}$ with input wires u,v and output wire w, the distinguisher computes the four ciphertexts of the garbled gate as follows:
  - First, the entry in the (ϵ_u,ϵ_v)-th row is computed as

$$\tilde{g}_{\epsilon_u,\epsilon_v}\leftarrow\mathsf{Enc}_{\mathcal{T}}(\ (\Psi(\mathbf{k}_{w,\epsilon_w},\epsilon_w),(g\|\epsilon_u\|\epsilon_v),\ (\mathbf{k}_{u,\epsilon_u},\mathbf{k}_{v,\epsilon_v})\ )$$

  - Next, for for all $(\alpha,\beta)\in\{0,1\}^2$ such that $(\alpha,\beta)\neq(\epsilon_u,\epsilon_v)$ the distinguisher sets $\ell_{\alpha,\beta}=0$ if $(\alpha\oplus\lambda_u)\cdot(\beta\oplus\lambda_v)=\ell_w$, and sets $\ell_{\alpha,\beta}=1$ otherwise. $\mathcal{D}'$ can then use the oracle to compute the inactive rows $(\alpha,\beta)\neq(\epsilon_u,\epsilon_v)$ as:

$$\tilde{g}_{\alpha,\beta}\leftarrow\mathcal{O}\Big(\ \mathbf{k}_{u,\epsilon_u}\oplus\sigma(\mathbf{k}_{v,\epsilon_v}),\epsilon_u\oplus\alpha,\epsilon_v\oplus\beta,\Psi(\mathbf{k}_{w,\epsilon_w},\epsilon_w),\ell_{\alpha,\beta},(g\|\alpha\|\beta)\ \Big)$$

- Finally, for each output gate, with input wire u and output wire w, the distinguisher sets ϵ_w as in the simulation and computes the two ciphertexts of the garbled output gate as follows:
  - The active entry (i.e. the ϵ_u-th row) is computed as

$$\tilde{g}_{\epsilon_u}\leftarrow\mathsf{Enc}_{\mathcal{T}_d}\Big(\ (\ (\xi^{i_1}_{w,\epsilon_u}\|\ldots\|\xi^{i_{|\mathcal{P}|}}_{w,\epsilon_u}),\ (g\|\epsilon_u\|0)\ ),\ (\mathbf{k}_{u,\epsilon_u},\mathbf{0})\ \Big)$$

  - The inactive entry is set as

$$\tilde{g}_{\bar{\epsilon}_u}\leftarrow\mathcal{O}\Big(\ \mathbf{k}_{u,\epsilon_u},1,0,(\xi^{i_1}_{w,\bar{\epsilon}_u}\|\ldots\|\xi^{i_{|\mathcal{P}|}}_{w,\bar{\epsilon}_u}),0,(g\|\bar{\epsilon}_u\|0)\ \Big)$$

- $\mathcal{D}'$ hands the adversary the complete description of the garbled circuit and concludes the execution as in the simulation with $\tilde{\mathcal{S}}$.
- $\mathcal{D}'$ outputs whatever $\mathcal{Z}$ does.

Note first that $\mathcal{D}'$ only makes legal queries to its oracle. Furthermore, if $\mathcal{O} = \mathsf{Real}^{\sigma}_{\Delta}$, then the view of $\mathcal{A}$ is identically distributed to its view in the real execution of the protocol on the given inputs. On the other hand, if $\mathcal{O} = \mathsf{Fake}^{\sigma}_{\Delta}$ then $\mathcal{A}$'s view is distributed identically to the output of $\tilde{\mathcal{S}}$ described previously. Hence, as a distinguishing $\mathcal{Z}$ implies breaking the LIN RK-KDM$^{\sigma}$ security of $(\mathsf{KeyGen}^{\mathsf{XOF}}, \mathsf{Enc}^{\mathsf{XOF}}_{\tau}, \mathsf{Dec}^{\mathsf{XOF}})$ and this contradicts Theorem 2, we conclude this is impossible. □

The following lemma completes the proof.

Lemma 7. Flip *only happens in* **HYB** *with negligible probability.*

Proof. First, we note that if Condition (F1) of Flip is not satisfied then the only option for $\mathcal{A}$ to check Condition (F2) is to guess the guard value $\xi^{i}_{w,\bar{\epsilon}_u}$, which can be done only with negligible probability. Hence, we restrict our attention to Condition (F1) for the rest of the proof. We proceed by showing that if Flip occurs in the real execution with a non-negligible probability, then we can leverage this distinguishing gap in order to break the LIN RK-KDM$^{\sigma}$ assumption.

More formally, assume towards contradiction that

$$\Pr[\mathsf{Flip} \text{ occurs in } \mathbf{HYB}] \geq \frac{1}{q(\mathsf{sec})}$$

for some non-negligible function $q(\cdot)$ and infinitely many sec's. We construct our distinguisher $\mathcal{D}$ as follows.

1. Distinguisher $\mathcal{D}$ is identically defined as the distinguisher in the proof of Lemma 6, externally communicating with an oracle $\mathcal{O}$ that either realizes the function $\mathsf{Real}^{\sigma}_{\Delta}$ or $\mathsf{Fake}^{\sigma}_{\Delta}$, while internally invoking $\mathcal{A}$.
2. Upon receiving the modified garbled circuit from $\mathcal{A}$, $\mathcal{D}$ evaluates the circuit on the parties' inputs and compares every active key $\hat{\mathbf{k}}_{w,\epsilon_w}$ that is revealed during the execution with the actual active key $\mathbf{k}_{w,\epsilon_w}$ that was created by $\mathcal{D}$ in the garbling phase. If these are different for some $w \in W$, then $\mathcal{D}$ stores such $\hat{\mathbf{k}}_{w,\epsilon_w}$ for the upcoming step.
3. Next, $\mathcal{D}$ queries its oracle as follows:

$$\hat{c} \leftarrow \mathcal{O}(\hat{\mathbf{k}}_{w,\epsilon_w}, 0, 1, 0, 0, \mathsf{nonce}_w)$$
$$c \leftarrow \mathcal{O}(\mathbf{k}_{w,\epsilon_w}, 1, 1, 0, 0, \mathsf{nonce}_w)$$

   for some unique nonce nonce_w. Given these outputs, $\mathcal{D}$ tries to decrypt $\hat{c} \oplus c$, if the result is zero, then then $\mathcal{D}$ outputs $\mathsf{Real}^{\sigma}_{\Delta}$.
4. Upon concluding the execution so that $\mathcal{D}$ did not output $\mathsf{Real}^{\sigma}_{\Delta}$, it returns $\mathsf{Fake}^{\sigma}_{\Delta}$.

Clearly, whenever $\mathcal{O} = \mathsf{Real}^{\sigma}_{\Delta}$ then the view of $\mathcal{A}$ is as in **HYB**, so Flip occurs with probability at least $1/q(\mathsf{sec})$. On the other hand, as in the claim made in Lemma 6, when $\mathcal{O} = \mathsf{Fake}^{\sigma}_{\Delta}$ then the adversary's view is as in **IDEAL**, so the probability that Flip occurs is negligible, and thus $\mathcal{D}$ outputs $\mathsf{Real}^{\sigma}_{\Delta}$ only with negligible probability. This implies a non-negligible gap with respect to the event occurring in the two executions and concludes the proof of the lemma. □

This lemma also concludes the proof of the theorem.

□

5.3 Overall Complexity Unauthenticated Variant

Just as in Section 4, we discuss the overall complexity in terms of calls to the underlying base MPC functionality.

- Step 1 requires each party to execute $k-2$ calls to Input^P.
- Recall, each call to GenBit requires one party to call Input, note this is the actively secure input procedure. Hence, Step 3 requires $(|C_\wedge|+|C_i|)$ calls to Input and $k \cdot (|C_\wedge|+|C_i|)$ calls to Input^P. All these inputs can be performed in parallel thus they require r_i rounds to complete.
- Step 5a requires $|C_\wedge|$ calls to Mult (in order to compute $[\lambda_u] \cdot [\lambda_v]$ for every AND gate). This requires only r_m rounds as all multiplications can be done in parallel.
- Next steps consist of completely local operations, bar the one round of communication needed to execute 5b. Note, $4 \cdot |C_\wedge|$ can all be performed in parallel, meaning this only costs us one round of communication in total.
- Step 1 requires $4 \cdot \ell \cdot |C_\wedge|$ calls to Open^P and calls to for $\mathcal{F}_{\text{Zero}}$ (which we discount as they are non-interactive in nature).
- Step 5e requires $|C_o| \cdot \ell$ calls to Open^P. Step 1 and 5e can be performed in parallel, and hence they require together r_o rounds of communication.

Thus the cost per party of this garbling and evaluation procedure is given in the following table. We easily see that the cost of generating the garbled circuit is *potentially* less in this case than in the authenticated garbling variant. The number of actively secure multiplications only depends on the circuit size and not on the LPN parameters. In terms of evaluation it would appear that this variant is more efficient, no MPC sub-procedures are needed to execute the evaluation phase. However, in our current implementation, the parameters required for the codes in order to avoid decryption failures are very big. We investigate this relationship more in the following section.

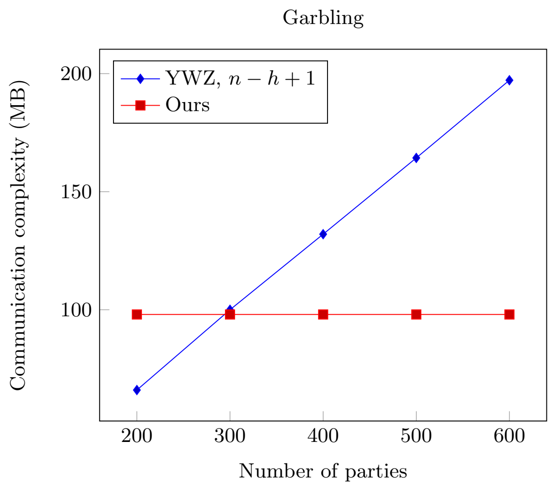
	Preprocessing	Evaluation
Input^P	$(k-2)+\lvert C_i\rvert+\lvert C_\wedge\rvert+\ell\cdot\lvert C_o\rvert$	–
Input	$k\cdot(\lvert C_\wedge+C_i\rvert)$	–
Open^P	$4\cdot\ell\cdot(\lvert C_\wedge\rvert+\lvert C_o\rvert)$	–
Output^P	$\lvert C_i\rvert+\lvert C_o\rvert$	–
Mult	$\lvert C_\wedge\rvert$	–
$\mathsf{Bit}\times\mathsf{String}$	$4\cdot\lvert C_\wedge\rvert$	–
Rounds	$r_i+r_m+r_o+1$	2 Broadcast

5.4 Communication Complexity

Given the previous analysis, we estimated the communication complexity of our preprocessing protocol, consisting of a function-independent phase that we instantiate with a TinyOT-like protocol, and a function-dependent phase (garbling). As in previous works, we moved the garbled circuit opening to the preprocessing to have a simpler online evaluation.

We consider the protocol in Section 5, with $c=5$ and different number of parties. In particular, we assume $n \geq 200$, so that the number of honest parties h is larger than 40, which allows us to apply the bit/string product optmization described in Section 7.

We compare our results with YWZ [YWZ19], running with $n-h+1$ parties. Figure 10 reports the communication estimates for secure evaluation of AES-128, with 6400 AND gates. Note that



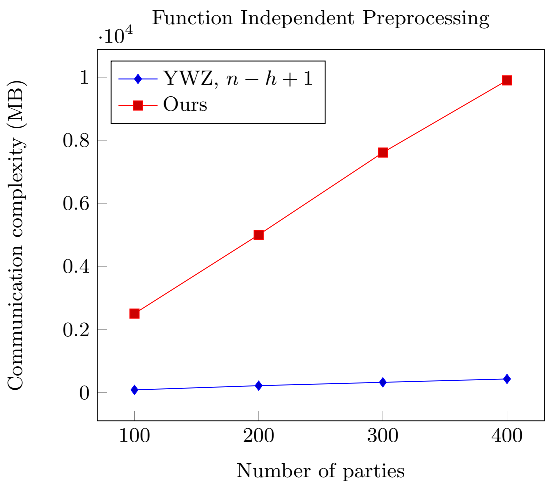


Fig. 10. Preprocessing communication estimates for AES-128 evaluation with our protocol and YWZ. Cost is the maximum amount of data sent by each party, per execution. The security parameters are set to sec $= 128$ and $s = 40$

[YWZ19] proposes several optmizations for the preprocessing phase of BMR-style protocols like HHS and WRK. While some of these are not directly applicable to our constructions, e.g. the half-gates technique, other, like the improvements on the authenticated bits generations, can be also used in our protocols.

We can see that our garbling becomes cheaper as soon as $n \geq 300$, but our function independent step is always more expensive than in YWZ. This is because we have to compute on values over $\mathbb{F}_2^k$ rather that $\mathbb{F}_2^{\text{sec}}$, where $k \gg \text{sec}$.

One way to improve the efficiency of this function-independent preprocessing, when the number of parties is bigger than 200, would be to run the protocol by smaller committees of parties.

In any case, the efficiency of both garbling and function-independent preprocessing are heavily influenced by the underlying error correcting code used in the LPN encryption, namely by the values ℓ and k. Finding a better code would imply an immediate improvement in the efficiency both in terms of communication and computation.

6 Implementation and Experimental Results

To demonstrate the practicality of our design, we implemented the circuit evaluation step for both of our protocols, and tested them on a number of 'standard' test circuits, given in Table 1. For the preprocessing phase, we give an estimation of the communication complexity in Section 5.4 and compare it with the recent work of Yang et al. [YWZ19].

The test circuits consisted of a combination of AND, XOR and INV gates. The SHA-256 and SHA-512 circuits implemented the compression function f only for a single block message m. Further, we compare our results with existing work at the end of this section.

The hash function H used to define our nonce-based LPN encryption function (Definition 6) is implemented using three variants. The first variant is based on the AES-KDF from NIST [NIS18]. This is very fast but it is not indifferentiable from a random oracle, and thus not strictly a true XOF. The second variant is based on the SHA-3 based XOF derived from KMAC128 and KMAC256 given in [NIS16]. The third variant is based on the Kangaroo-12 XOF from [BDP+18], which is

Circuit	No. ANDs	No. XORs	No. Invs
AES-128(k, m)	6400	28176	2087
AES-192(k, m)	7168	32080	2317
AES-256(k, m)	8832	39008	2826
Keccak-f(m)	38400	115200	38486
SHA-256-f(H, f)	22573	110644	1856
SHA-512-f(H, f)	57947	286724	4946

Table 1. Standard Test Circuits

also based on SHA-3 which provides 128-bits of security. For our two SHA-3 variants we used the library provided by the Keccak team https://keccak.team/. For the AES based KDF variant we used code using the Intel AES-NI instructions.

Code Instantiation. The codes we propose are based on the concatenation of $L = L_o \circ L_i$ over $\mathbb{F}_2$. Using the standard notation for error correcting codes, let L_o be a $[\ell_o, \mathfrak{m}_o, d_o]$ outer code (or big code) for the $\mathbb{F}_{2^f}$ alphabet:

$$L_o : \mathbb{F}_{2^f}^{\mathfrak{m}_o} \to \mathbb{F}_{2^f}^{\ell_o}.$$

Let L_i be a $[\ell_i, \mathfrak{m}_i, d_i]$ inner code (or small code) for the binary alphabet:

$$L_i : \mathbb{F}_2^{\mathfrak{m}_i} \to \mathbb{F}_2^{\ell_i}.$$

The resulting concatenation code is

$$L = L_o \circ L_i : \mathbb{F}_2^{\mathfrak{m}_i \mathfrak{m}_o} \to \mathbb{F}_2^{\ell_i \ell_o}.$$

Note that the inner code encodes one field element in $\mathbb{F}_{2^f}$ into ℓ_i bits. So we set f to $\mathfrak{m}_i$ since $|\mathbb{F}_{2^f}| = 2^{\mathfrak{m}_i}$. In other words, the dimension of the inner code is equal to the number of bits in the field element of the outer code.

Decoding is done by first decoding the inner code, mapping each ℓ_i bits into one $\mathfrak{m}_i$ bit field element. Then decoding the outer code, mapping ℓ_o field elements into $\mathfrak{m}_o$ field elements. Note that typically the inner code can only correct a small number of erroneous bits, therefore it decodes incorrectly with non-negligible probability. As long as the number of incorrect decodings can be handled by the outer code, the original message is correctly recovered.

In our work we consider an efficient concrete concatenated code where the outer code is based on Reed-Soloman, and the inner code is taken from a list of best-possible linear error-correcting codes in small dimension.

Outer Code (based on Reed-Soloman): Reed-Solomon (RS) codes work over a finite field $\mathbb{F}_q$ ($q = p^f$ where p is a prime). An RS code encodes $L_o : \mathbb{F}_q^{\mathfrak{m}_o} \to \mathbb{F}_q^{q-1}$ and can correct up to $\lfloor \frac{q-\mathfrak{m}_o}{2} \rfloor$ erroneous field elements. Note that $\mathfrak{m}_o$ here is the number of *field elements* in the message, also known as the dimension. We select the parameter based on the message size and the inner code. We use $p = 2$ to match the inner code. The field size f is set to be $\mathfrak{m}_i$ as mentioned above. Then we can fix the code length $\ell_o = 2^f - 1$ which is typical for RS codes and the dimension $\mathfrak{m}_o = \lceil k/f \rceil$ so that we can have enough dimension to encode the whole message; where k above is a parameter from the LPN cryptosystem from Section 2.3 that represents the original dimension. Decoding is done using the standard way, i.e., via the Berlekamp–Massey algorithm [Mas69].

sec	τ	$k \geq$	$\mathfrak{m}$	s	$[\ell_i, \mathfrak{m}_i, d_i]$	$[\ell_o, \mathfrak{m}_o, d_o]$	ℓ	rate	$\Pr_{\text{fail}}$
128	$\frac{1}{8}$	664	672	> 40	$[28, 8, 11]$	$[255, 84, 172]$	7140	0.094	2^{-51}
128	$\frac{1}{16}$	1374	1376	> 40	$[19, 8, 7]$	$[255, 172, 84]$	4845	0.284	2^{-58}
256	$\frac{1}{8}$	1328	1332	> 40	$[29, 9, 11]$	$[511, 148, 364]$	14819	0.09	2^{-98}
256	$\frac{1}{16}$	2750	2754	> 40	$[17, 9, 5]$	$[511, 306, 206]$	8687	0.317	2^{-45}
128	$\frac{1}{8}$	664	672	> 80	$[35, 8, 15]$	$[255, 84, 172]$	8925	0.075	2^{-120}
128	$\frac{1}{16}$	1374	1376	> 80	$[25, 8, 9]$	$[255, 172, 84]$	6375	0.093	2^{-81}
256	$\frac{1}{8}$	1328	1332	> 80	$[29, 9, 11]$	$[511, 148, 364]$	14819	0.090	2^{-98}
256	$\frac{1}{16}$	2750	2754	> 80	$[20, 9, 7]$	$[511, 306, 206]$	10220	0.270	2^{-153}

Table 2. Concatenated code parameters for the authenticated garbling variant.

Inner Code (linear code for the binary alphabet): Our outer RS code encodes words over a finite field to correct erroneous *field elements*. However, our added error for the LPN security requires adding an error independently on each *bit* (more details in Section 2.3). Therefore, after encoding using the outer RS code, we encode each field element (i.e., every f or $\mathfrak{m}_i$ bits) using L_i. This ensures that the added errors to the bits of the final code translate, after decoding the inner code, to few erroneous field elements on the RS-encoded message. These few erroneous field elements are then corrected by the outer RS code resulting in the original message. The concrete inner code is flexible. Thus we search over all available codes from Code Tables[10] to find the optimal one. We give concrete details below for various parameters.

The decoding algorithm depends on the parameters. For codes that are small, we can use syndrome decoding with pre-computed syndrome table, which is fast and easy to implement. The tables have size $2^{\ell_i - \mathfrak{m}_i}$ which must fit into the computer memory. Otherwise, the Berlekamp–Massey algorithm [Mas69] is used if code happens to be a BCH code.

Instantiations: Finding concrete instantiations of L_i and L_o can be stated as an optimization problem. Given the LPN dimension k and the bit-error probability τ, maximize $(\mathfrak{m}_i \cdot \mathfrak{m}_o)/(\ell_i \cdot \ell_o)$ in the concrete instantiations of L_i and L_o such that the decoding error probability is less than some target, e.g., 2^{-s}, where $s = 40$ or $s = 80$. In other words, we maximize the *rate* while keeping the decoding error probability low. We also require the code to be big enough to encode the LPN message, i.e., we want the dimension $\mathfrak{m} = \mathfrak{m}_i \cdot \mathfrak{m}_o$ to satisfy $\mathfrak{m} \geq k + 1$. The problem is solved using the following strategy.

1. For every linear $[\ell_i, \mathfrak{m}_i, d_i]$ code from Code Tables do the following.
   (a) Use the linear code as L_i.
   (b) Compute the decoding error probability $p = \Pr(X \geq \lfloor \frac{d-1}{2} \rfloor)$ of L_i using τ as the probability of a bit-flip.
   (c) Fix the outer RS code $[\ell_o = 2^{\mathfrak{m}_i} - 1, \mathfrak{m}_o = \lceil k/\mathfrak{m}_i \rceil, d_o = \ell_o - \mathfrak{m}_o + 1]$.
   (d) Tail bound the decoding error probability $\Pr(X \geq \lfloor \frac{d_o-1}{2} \rfloor)$ of L_o using p as the probability of an incorrect field element.
2. Remove the codes that have a decoding error probability greater some target (e.g., 2^{-80}) in the outer code.
3. Output L_i and L_o with the largest rate $(\mathfrak{m}_i \cdot \mathfrak{m}_o)/(\ell_i \cdot \ell_o)$.

[10] http://www.codetables.de/

sec	τ_d	c	τ_s	k	s	$\mathfrak{m}$	$[\ell_i, \mathfrak{m}_i, d_i]$	$[\ell_o, \mathfrak{m}_o, d_o]$	ℓ	rate	$\Pr_{\text{fail}}$
128	$\frac{1}{8}$	2	0.06699	1280	> 40	1287	$[29, 9, 11]$	$[511, 143, 369]$	14819	0.087	2^{-103}
128	$\frac{1}{8}$	5	0.02796	3130	> 40	3140	$[30, 10, 11]$	$[1023, 314, 710]$	30690	0.102	2^{-143}
128	$\frac{1}{8}$	10	0.01418	6213	> 40	6220	$[39, 10, 15]$	$[1023, 622, 402]$	39897	0.156	2^{-50}
256	$\frac{1}{8}$	2	0.06699	2560	> 40	2565	$[37, 9, 15]$	$[511, 285, 227]$	18907	0.136	2^{-65}
256	$\frac{1}{8}$	5	0.02796	6259	> 40	6260	$[39, 10, 15]$	$[1023, 626, 398]$	39897	0.157	2^{-48}
256	$\frac{1}{8}$	10	0.01418	12426	> 40	12430	$[40, 11, 15]$	$[2047, 1130, 918]$	81880	0.152	2^{-126}
128	$\frac{1}{8}$	2	0.06699	1280	> 80	1287	$[29, 9, 11]$	$[511, 143, 369]$	14819	0.087	2^{-103}
128	$\frac{1}{8}$	5	0.02796	3130	> 80	3140	$[30, 10, 11]$	$[1023, 314, 710]$	30690	0.102	2^{-143}
128	$\frac{1}{8}$	10	0.01418	6213	> 80	6215	$[31, 11, 11]$	$[2047, 565, 1483]$	63457	0.098	2^{-260}
256	$\frac{1}{8}$	2	0.06699	2560	> 80	2565	$[42, 9, 17]$	$[511, 285, 227]$	21462	0.12	2^{-82}
256	$\frac{1}{8}$	5	0.02796	6259	> 80	6270	$[31, 11, 11]$	$[2047, 570, 1478]$	63457	0.099	2^{-256}
256	$\frac{1}{8}$	10	0.01418	12426	> 80	12430	$[40, 11, 15]$	$[2047, 1130, 918]$	81880	0.152	2^{-126}

Table 3. Choices of LPN parameters and codes for various security levels and choices of c, for the unauthenticated garbling variant.

Using the strategy above, we obtain some possible codes in Table 2 for our authenticated garbling variant. Note that we set the error probability 2^{-s} to be 2^{-40} in the first four rows and 2^{-80} in the last four. We only use $\tau = \frac{1}{8}$ or $\frac{1}{16}$ because when the denominator is a power of 2 then then it is easy to to generate the error vector in MPC.

The codes for the unauthenticated garbling variant is in Table 3, again for both $s = 40$ and $s = 80$. We only use $\tau_d = \frac{1}{8}$ because it offers a good balance between the code size and decodability.

Online Implementation Results. The expensive parts of the algorithms are the lines in Step 4 of Figure 5 in the authenticated garbling method, and the lines in Step 5 of Figure 8 for the unauthenticated garbling method; thus these were the parts of the algorithm we timed. Experiments were run on a Intel i7-7700K CPU 4.20GHz machine with 32GB of RAM.

	Execution Time (sec)			
	128-bit Security			256-bit Security
Circuit	AES-KDF	KMAC128	Kangaroo	KMAC256
AES-128(k, m)	1.72	6.64	4.04	35.4
AES-192(k, m)	1.92	7.41	4.51	39.9
AES-256(k, m)	2.35	9.13	5.58	48.9
Keccak-f(m)	10.2	39.7	24.3	214
SHA-256-f(H, f)	6.02	23.3	14.3	128
SHA-512-f(H, f)	15.6	60.0	36.8	327

Table 4. Evaluation (in sec) of various circuits in the authenticated garbling case. Setting $\mathsf{sec} = 128$ and $s = 40$, the LPN parameters are $(k, \mathfrak{m}, \ell, \tau) = (664, 672, 7140, 1/8)$ and we use the error correcting given by ($L_o = [255, 84, 172], L_i = [28, 8, 15]$). For 256 bit security, the LPN parameters are $(k, \mathfrak{m}, \ell, \tau) = (1328, 1332, 14819, 1/8)$ and the error correcting code is given by ($L_o = [511, 148, 364], L_i = [29, 9, 11]$). Details of these codes are given in Table 2.

For the authenticated garbling (resp. unauthenticated garbling) variant of our algorithm, we obtained the run-times presented in Table 4 (resp. Table 6) with decryption failure $s = 40$. For equivalent runtimes when $s = 80$ see respectively Table 5 and Table 7. In these tables the security level refers to the security of the underlying LPN function. Observe that the choice of the underlying

	Execution Time (sec)			
	128-bit Security			256-bit Security
Circuit	AES-KDF	KMAC128	Kangaroo	KMAC256
AES-128(k, m)	3.10	9.19	5.96	35.4
AES-192(k, m)	3.42	10.3	6.67	39.9
AES-256(k, m)	4.21	12.7	8.21	48.9
Keccak-f(m)	18.3	55.1	35.7	214
SHA-256-f(H, f)	10.9	32.4	21.9	128
SHA-512-f(H, f)	27.6	82.3	54.2	327

Table 5. Evaluation of various circuits in the authenticated garbling case. Setting sec = 128 and $s = 80$, the LPN parameters are $(k, m, \ell, \tau) = (664, 672, 8925, 1/8)$ and the error correcting code is given by $(L_o = [255, 84, 172], L_i = [35, 8, 15])$. Setting sec = 256 and $s = 80$, the LPN parameters are $(k, \mathfrak{m}, \ell, \tau) = (1328, 1332, 14819, 1/8)$ and we use the error correcting code given by $(L_o = [511, 148, 364], L_i = [29, 9, 11])$.

	Execution Time (s)					
	128-bit Security			256-bit Security		
Circuit	$c = 2$	$c = 5$	$c = 10$	$c = 2$	$c = 5$	$c = 10$
AES-128(k, m)	10.5	50.4	77.5	16.9	80.2	538
AES-192(k, m)	11.7	56.3	86.7	18.9	89.3	602
AES-256(k, m)	14.4	69.1	106	23.4	110	742
Keccak-f(m)	64.4	309	474	104	490	3333
SHA-256-f(H, f)	36.7	176	271	59.5	284	1899
SHA-512-f(H, f)	94.0	451	692	152	725	4848

Table 6. Evaluation of various circuits in the unauthenticated garbling variant, using the AES-KDF, and $s = 40$. For the parameters for the LPN scheme, and the associated error correcting code we used those given in Table 3.

method to generate the LPN matrix has a key effect on the performance of the system, with an AES based KDF being the most efficient. For the unauthenticated garbling variant, we only present runtimes using the efficient AES based KDF function. Concretely, when using AES-KDF, a majority (81%) of the CPU time is spent in decoding. When using KMAC128, the majority (84%) of the time is spent on KMAC128. Thus, the performance bottleneck varies with the choice of H.

We compare our scheme with some related work. In the authenticated garbling case, and the fastest implementation using an AES-KDF based for the function H, we obtain a throughput of roughly 266 microseconds per AND gate for $s = 40$. The experiments from [BLO17], i.e. in the passive case, with no free-XOR, has a throughput of roughly 45 microseconds per *gate* (also with

	Execution Time (sec)					
	128-bit Security			256-bit Security		
Circuit	$c = 2$	$c = 5$	$c = 10$	$c = 2$	$c = 5$	$c = 10$
AES-128(k, m)	10.5	50.4	304	20.3	304	538
AES-192(k, m)	11.7	56.3	341	22.8	341	602
AES-256(k, m)	14.4	69.1	417	27.7	418	742
Keccak-f(m)	64.4	309	1858	124	1859	3333
SHA-256-f(H, f)	36.7	176	1059	70.8	1062	1899
SHA-512-f(H, f)	94.0	451	2727	181	2714	4848

Table 7. Evaluation of various circuits in the unauthenticated garbling case, using the AES-KDF, and $s = 80$. The parameters for the LPN scheme, and the associated error correcting code we used are given in Table 3.

$s = 40$). Ignoring the fact we can perform free-XOR, this gives a cost of a factor of six for using our actively secure variant. However, this cost decreases when we look at typical circuits. For example the AES-128 circuit has $34,675$ AND and XOR gates, thus the protocol in [BLO17] would take around 1.5 seconds, compared to our runtime of 1.72 seconds. Thus, the ability to cope with free-XOR means we only pay an extra 15% in performance for active security.

As a means of comparison with 'traditional' n-party garbled circuits via actively secure BMR with free-XOR, we extrapolated known run times of evaluating AES-128 using the HSS protocol. It would appear that our algorithm will provide a faster *evaluation* stage when the number of parties exceeds about 100 in the authenticated garbling case. This is confirmed by a comparison with [WRK17b] that reports an online running time of 2.3 sec for AES with 128 parties in the WAN setting.

7 Instantiation of $\langle b \cdot \varDelta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \varDelta \rangle}([b])$ via Tiny-OT

We first note that the functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ can be efficiently implemented using an n-party TinyOT-like protocol, see [BLN+15, KOS15, FKOS15, HSS17, WRK17b] for the basic blueprint and various optimizations. In particular, these protocols produce the authenticated value $[x]$ as follows:

$$[x] = \{x^i, \bar{\varDelta}^i, \{\mathbf{m}^{i,j}, \mathbf{k}^{i,j}\}_{j \neq i}\}_{i \in [n]}, \quad \mathbf{m}^{j,i} = \mathbf{k}^{i,j} \ \oplus \ x^j \cdot \hat{\varDelta}^i,$$

where each party P_i holds the $n-1$ MACs $\{\mathbf{m}^{i,j}\} \in \{0,1\}^{\mathsf{sec}}$ on x^i, as well as the keys $\mathbf{k}^{i,j} \in \{0,1\}^{\mathsf{sec}}$ on each x^j, for $j \neq i$, and a global key $\bar{\varDelta}^i \in \{0,1\}^{\mathsf{sec}}$.

As noticed by Hazay et al. [HSS17], assuming $\varDelta = \oplus_i \bar{\varDelta}^i \in \mathbb{F}_2^k$ has been fixed, TinyOT authentication allows to efficiently compute unauthenticated sharing of $\langle x \cdot \varDelta \rangle$ given a sharing of a bit $[x]$. We denote this operation by

$$\langle x \cdot \varDelta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \varDelta \rangle}([x])$$

The problem with using the solution of [HSS17] for the $\mathsf{Bit} \times \mathsf{String}$ operation in our protocol, is that to get the most efficient base TinyOT protocol we select sec to be the desired (symmetric) security level. Thus, one can think of sec as being equal to 128 or 256. However, in our protocols we require k, the length of $\varDelta$, to be very large, as it corresponds to the dimension of the secret key space for an LPN encryption scheme. Hence, the problem is that we have $k \gg \mathsf{sec}$.

Naïvely, one could use the above solution for $\mathsf{Bit} \times \mathsf{String}$ by running TinyOT with keys of length k. However, this would increase the cost of oblivious operations. It turns out that when the number of *honest* parties is bigger than the statistical security parameter s, as in the setting of our second variant when the number of parties is large, there exists a significantly more efficient solution.

We thus give an efficient method for stretching TinyOT authenticated bits (with global keys $\bar{\varDelta}^i \in \{0,1\}^{\mathsf{sec}}$) to enable the $\mathsf{Bit} \times \mathsf{String}$ operation above, with a global $\varDelta$ value in $\{0,1\}^k$, for any desired value of k. The functionality associated with our method is slightly different than standard, in order to increase efficiency based on the specific needs of our protocol. In particular, we allow the adversary to do an additive attack (i.e., add errors to the output as long as the errors do not depend on sensitive information), which is later caught in our main protocol. Additionally, the protocol might leak a few of the *shares* of the honest parties of the secret-shared bits. However, we show that by assuming the number of honest is greater than the statistical security, this leakage is not a security concern in our protocol. The formal description of the functionality is given in Figure 13.

Main step of $\langle b \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([b])$

Setup: The parties had run TinyOT on L shared bits $\langle b_\ell \rangle_{\ell=1}^{L}$, where $L \gg \mathsf{sec}$ is the number of $\mathsf{Bit} \times \mathsf{String}$ operations to be performed.
Denote by $\mathbf{k}_\ell^{i,j}$, resp. $\mathbf{m}_\ell^{j,i}$, the key corresponding to zero, resp. the received key, for parties P_i as sender and P_j as receiver on the ℓ bit, and by $\bar{\Delta}^i$ the TinyOT global key offset of party P_i.

Compute:
1. Each party P_i chooses a random $\Delta^i \leftarrow \{0,1\}^k$. Denote $\Delta = \bigoplus_{i=1}^{n} \Delta^i$.
2. For every shared bit $b_\ell, \ell \in [L]$, each pair of parties P_i,P_j performs:
   (a) P_i sends to P_j the value $u_{i,j,\ell} \leftarrow \mathsf{PRG}(\mathbf{k}_\ell^{i,j}) \oplus \mathsf{PRG}(\mathbf{k}_\ell^{i,j} \oplus \bar{\Delta}^i) \oplus \Delta^i$.
   (b) P_j sets $\mathbf{x}_{\ell_1}^{j,i} \leftarrow \mathsf{PRG}(\mathbf{m}_\ell^{j,i}) \oplus b_\ell^j \cdot u_{i,j,\ell}$.
   (c) P_i sets $\mathbf{x}_{\ell_2}^{i,j} \leftarrow \mathsf{PRG}(\mathbf{k}_\ell^{i,j})$.

Output: The parties run the MACCheck protocol in Figure 12. If no abort happened, each party P_i outputs $\oplus_{j \neq i}(\mathbf{x}_{\ell_1}^{i,j} \oplus \mathbf{x}_{\ell_2}^{i,j}) \oplus b^i \Delta^i$ for each $\ell \in [L]$.

Figure 11. Protocol $\langle b \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([b])$, main step

The method consists of two parts: the main part computes the $\mathsf{Bit} \times \mathsf{String}$ operation in the $\mathcal{F}_{\text{TinyOT}}$-hybrid model and the MACCheck enables malicious security. The communication complexity for computing a $\mathsf{Bit} \times \mathsf{String}$ operation is essentially each party sending a single message of length k to every other party, with an additional small overhead created by the MACCheck. Computationally, our MACCheck protocol is slightly heavier than similar MACCheck protocols (e.g., in [BLN+15, KOS16, KPR18]) and requires approx. n multiplications for each $\mathsf{Bit} \times \mathsf{String}$ operation.

To perform $\mathsf{Bit} \times \mathsf{String}$, we use a key-correlation-robust[11] pseudo-random generator

$$\mathsf{PRG} : \{0,1\}^{\mathsf{sec}} \to \{0,1\}^k.$$

We can now define the operation of the function $\mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([b])$. We let $\{b^i\}_{i \in [n]} \in \mathbb{F}_2$ be the additive shares of $[b]$, and we define $\Delta = \oplus \Delta^i$ to be a global random constant with $\Delta^i \in \mathbb{F}_2^k$ a uniformly random bit-string known only to party P_i. Clearly, if we can produce an additive sharing $\langle b \cdot \Delta^i \rangle$, we can obtain $\langle b \cdot \Delta \rangle$ by summing these values. Thus, the idea is as follows:

1. Party P_i sends to party P_j the value $u_{i,j} \leftarrow \mathsf{PRG}(\mathbf{k}^{i,j}) \oplus \mathsf{PRG}(\mathbf{k}^{i,j} \oplus \bar{\Delta}^i) \oplus \Delta^i$.
2. P_j sets $\mathbf{x}_1^{j,i} \leftarrow \mathsf{PRG}(\mathbf{m}^{j,i}) \oplus b^j \cdot u_{i,j}$.
3. P_i sets $\mathbf{x}_2^{i,j} \leftarrow \mathsf{PRG}(\mathbf{k}^{i,j})$.

This gives us

$$\begin{aligned} \mathbf{x}_1^{j,i} \oplus \mathbf{x}_2^{i,j} &= \mathsf{PRG}(\mathbf{m}^{j,i}) \oplus b^j \cdot u_{i,j} \oplus \mathsf{PRG}(\mathbf{k}^{i,j}) \\ &= b^j \cdot \Delta^i. \end{aligned}$$

Thus, the sharing $\langle b \cdot \Delta \rangle$ is given by each player P_i using the value $\oplus_{j \neq i}(\mathbf{x}_1^{i,j} \oplus \mathbf{x}_2^{i,j}) \oplus b^i \Delta^i$ (note that each party performs the above step with each other party both as sender and receiver). The formal protocol is given in Figure 11.

This suffices in the semi-honest model. In order to achieve malicious security (see Proposition 1) the parties run the MACCheck protocol at the end of the protocol, after the parties executed Step 2 for all the shared bits and additional sec shared bits $[r_h]$. These additional bits are used in order

[11] By key-correlation-robust PRG we mean that for random $k_1, \ldots, k_\ell$, the outputs $\mathsf{PRG}(k_1), \ldots, \mathsf{PRG}(k_\ell)$ remain computationally random to a distinguisher that is given the keys $k_1 \oplus \bar{\Delta}, \ldots, k_\ell \oplus \bar{\Delta}$ for a fixed global random $\bar{\Delta}$.

MACCheck for $\langle b \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([b])$

3. The parties call F_{rand} to obtain nL random elements $\chi_1^1, \ldots, \chi_L^n \in \mathbb{F}_{2^{\sec}}$
4. Each party P_j computes, over $\mathbb{F}_{2^{\sec}}$, the value

$$C^j = \bigoplus_{\ell=1}^{L} \chi_\ell^j \cdot b_\ell^j \oplus \bigoplus_{h=1}^{\sec} X^{h-1} \cdot r_h^j.$$

Denote $C = \bigoplus C^j$.

5. The parties rerandomize their additive share C^j of C to obtain new additive shares $\widetilde{C^j}$ (i.e., $C = \bigoplus C^j = \bigoplus \widetilde{C^j}$)[a]
6. Each party P_j broadcasts $\widetilde{C^j}$.
7. Each party P_j computes the value

$$Z_j^j = \bigoplus_{i \neq j} \bigoplus_{\ell=1}^{L} \chi_\ell^j \mathbf{x}_{\ell 1}^{j,i} \oplus \bigoplus_{i \neq j} \bigoplus_{h=1}^{\sec} X^{h-1} \cdot \mathbf{x}_{h 1}^{j,i} \oplus (C^j \oplus \widetilde{C^j}) \cdot \Delta^j \tag{9}$$

and, for each $i \neq j$, computes the value

$$Z_j^i = \bigoplus_{\ell=1}^{L} \chi_\ell^i \mathbf{x}_{\ell 2}^{j,i} \oplus \bigoplus_{h=1}^{\sec} X^{h-1} \cdot \mathbf{x}_{h 2}^{j,i} \oplus \widetilde{C^i} \cdot \Delta^j \tag{10}$$

Denote $\mathcal{Z}_j := Z_j^j + \bigoplus_{i \neq j} Z_j^i$ and $\mathcal{Z} = \bigoplus_{j=1}^{n} \mathcal{Z}_j$.

8. The parties rerandomize their additive share $\mathcal{Z}_j$ of $\mathcal{Z}$ to obtain new additive shares $\widetilde{\mathcal{Z}}_j$.[a]
9. Each party P_j commits to the value $\widetilde{\mathcal{Z}}_j$.
10. The parties open their commitments and check that $\bigoplus_{j=1}^{n} \widetilde{\mathcal{Z}}_j = 0$.

[a] This step can be done non-interactively, see, e.g., [HSS17].

Figure 12. Protocol for $\langle b \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle \Delta \rangle}([b])$, MACCheck part

to mask the linear combination $\bigoplus_{j=1}^{n} \bigoplus_{\ell=1}^{L} \chi_\ell^j \cdot b_\ell^j$. This is done by multiplying each bit r_h with a string containing '1′ only in the $(h-1)$th bit, denote this string by X^{h-1}, and then XORing the results. The MACCheck protocol is formally given in Figure 12. It uses the standard functionalities $\mathcal{F}_{\text{rand}}$ for generating common random elements and $\mathcal{F}_{\text{commit}}$ for commitment, see, e.g., [BLN+15] for more details and protocol descriptions.

It is important to note that the MACCheck in Figure 12 differs from standard MACChecks in similar protocols (e.g., [BLN+15, KOS16, KPR18]), because the parties generate a different random element χ_ℓ^j for each *share* b_ℓ^j, whereas in [BLN+15, KOS16, KPR18] the same random element is used for all the shares of the same *shared bit* $[b_\ell]$. While this difference increases the computational complexity of the MACCheck ($O(n)$ field multiplications instead of (1) per $\mathsf{Bit} \times \mathsf{String}$), it is crucial for security in our case – if the same random element is used for all the shares of the same shared bit $[b_\ell]$ in the MACCheck below, the adversary can mount a selective failure attack and learn some of the shared bits with non-negligible probability, see Remark 4

Before proceeding to prove security of the protocol, we make a few observations and notation. The corrupt subset is denoted by $\mathcal{A} \subset [n]$ and the honest subset by $\mathcal{H} := [n] \setminus \mathcal{A}$, the view of the adversary in the above protocol is denoted $\mathsf{view}_{\mathcal{A}}$. The outputs of the protocol in an honest/malicious execution are denoted by by out and $\widehat{\mathsf{out}}$, respectively, and the error by err, i.e., $\mathsf{err} = \widehat{\mathsf{out}} \oplus \mathsf{out}$. We further denote $\Delta_{\mathcal{A}} := \bigoplus_{i \in \mathcal{A}} \Delta^i$ and $\Delta_{\mathcal{H}} := \bigoplus_{i \notin \mathcal{A}} \Delta^i$ (of course, the adversary may choose by itself

Δ^i for each $i \in \mathcal{A}$, and, additionally, the adversary may also try to use different Δ^i's with different parties and/or for different b_ℓ's in Step 2 as we explain below).

We note that if no cheating has occurred, then in Step 10 it holds that

$$\bigoplus_{j=1}^{n} \widetilde{\mathcal{Z}}_j = \bigoplus_{j=1}^{n} \mathcal{Z}_j = \bigoplus_{j=1}^{n} \left(Z_j^j + \bigoplus_{i \neq j} Z_j^i \right) = \bigoplus_{j=1}^{n} Z_j^j + \bigoplus_{j=1}^{n} \bigoplus_{i \neq j} Z_j^i$$

$$= \bigoplus_{j=1}^{n} Z_j^j + \bigoplus_{j=1}^{n} \bigoplus_{i \neq j} Z_i^j = \bigoplus_{j=1}^{n} \left(Z_j^j + \bigoplus_{i \neq j} Z_i^j \right)$$

$$= \bigoplus_{j=1}^{n} \left((C^j + \widetilde{C^j})\Delta^j \oplus \bigoplus_{i \neq j} \left(\left(\bigoplus_{\ell=1}^{L} \chi_\ell^j b_\ell^j \oplus \bigoplus_{h=1}^{\text{sec}} X^{h-1} \cdot r_h^j \right) \Delta^i + \widetilde{C^j} \Delta^i \right) \right)$$

$$= \bigoplus_{j=1}^{n} \left((C^j + \widetilde{C^j})\Delta^j \oplus \bigoplus_{i \neq j} \left(C^j \Delta^i + \widetilde{C^j} \Delta^i \right) \right) = C\Delta \oplus C\Delta = 0$$

We further observe that in the main part of the protocol (Step 2), each party only sends a single message to each other party . If P_i is honest, then P_j has only one of $\{\mathbf{k}^{i,j}, \mathbf{k}^{i,j} \oplus \bar{\Delta}^i\}$. Thus, intuitively, assuming a key-correlation-robust PRG is used, $\mathsf{PRG}(\mathbf{k}^{i,j}) \oplus \mathsf{PRG}(\mathbf{k}^{i,j} \oplus \bar{\Delta}^i)$ is indistinguishable from random to an adversary corrupting P_j (and any other party other than P_i) and, therefore, so is $u_{i,j}$.

We note that $\mathsf{view}_{\mathcal{A}}$ consists of:

1. The randomness of P_i for $i \in \mathcal{A}$,
2. The messages $u_{i,j,\ell}$'s from Step 2, for every $i \notin \mathcal{A}$ and $j \in \mathcal{A}$.
3. The random bits χ_ℓ^js, the linear combination C, and the sum of the commitments $\mathcal{Z}$.

The view also contains shares of the honest parties, e.g., $\widetilde{C}_j$, $\widetilde{\mathcal{Z}}_j$ for every $j \notin \mathcal{A}$, but these leak no additional information and are independent of everything else, so we usually implicitly ignore them. Additionally, observe that $\mathcal{Z} \neq 0$ implies that the honest parties abort.

Next, we list the possible deviations from the protocol by the adversary.[12]

1. Choose by itself $\widehat{\Delta^i} \neq \Delta^i$ for each $i \in \mathcal{A}$,
2. For each corrupt P_i and honest P_j, send incorrect $\widehat{u}_{i,j,\ell} = u_{i,j,\ell} + e_{i,j,\ell}$ in Step 2a.
3. Broadcast erroneous $\widehat{C^i} = \widetilde{C^i} + EC_i$ for corrupt P_i in Step 6.
4. Commit to erroneous $\widehat{\mathcal{Z}_i} = \widetilde{\mathcal{Z}}_i + E_i$ for corrupt P_i in Step 9.
5. Output shares $\widehat{\mathbf{x}}_1^{i,j} \neq \mathbf{x}_1^{i,j}$ and $\widehat{\mathbf{x}}_2^{i,j} \neq \mathbf{x}_2^{i,j}$, for $i \in \mathcal{A}$.

We remark that we have simplified the proof of security because we do not consider here that in Step 2 the adversary may also try to cheat on the OTs of the r_h's – we explain this in Remark 3.

Option (1) is captured by the functionality – the requirement that Δ is random remains true regardless of the Δ^i's of the adversary (since the Δ^i's of the honest parties are random). In the following, Δ^i for $i \in \mathcal{A}$ will often denote the adversary's chosen $\widehat{\Delta^i}$, and $\widehat{u}_{i,j,\ell} = u_{i,j,\ell} + e_{i,j,\ell}$, where $u_{i,j,\ell}$ is computed using the adversary's chosen Δ^i.

[12] We ignore deviations that result in immediate abort, e.g., sending messages of incorrect format.

Note that Option (5) allows the adversary to add any error of its choice to its output. Additionally, Option (2) might add an error to the honest parties' output. However, we show that the error introduced is computationally independent of Δ, $\bar{\Delta}^i$ for $i \notin \mathcal{A}$, and b_ℓ. The messages in Options (3) and (4) are used only in the MACCheck and do not affect the output.

Before proceeding to the main part of the proof, we show that Option (3) can in some sense be ignored, because it causes abort with overwhelming probability. Denote $E_\mathcal{A} = \bigoplus_{i\in\mathcal{A}} E_i$, $EC_\mathcal{A} := \bigoplus_{i\in\mathcal{A}} EC_i$, and $e_{\mathcal{A},j,\ell} := \bigoplus_{i\in\mathcal{A}} e_{i,j,\ell}$. We now observe that in order for the check in Step 10 to pass, it must hold that[13]

$$0 = \bigoplus_{j\notin\mathcal{A}} \widetilde{\mathcal{Z}}_j + \bigoplus_{j\in\mathcal{A}} \widehat{\mathcal{Z}}_j = \cdots = E_\mathcal{A} + \bigoplus_{j\notin\mathcal{A}} \bigoplus_{\ell} b_\ell^j \cdot \chi_\ell^j \cdot e_{\mathcal{A},j,\ell} + EC_\mathcal{A}\Delta_\mathcal{H}. \tag{11}$$

Lemma 8. *If $EC_\mathcal{A} \neq 0$ and $\Delta_\mathcal{H}$ is computationally indistinguishable from random then Equation (11) does not hold with overwhelming probability.*

Proof. If $EC_\mathcal{A} \neq 0$ then $EC_\mathcal{A} \cdot \Delta_\mathcal{H}$ is computationally indistinguishable from random (since $\Delta_\mathcal{H}$ is), and therefore a computationally bounded adversary cannot choose E_i's and $e_{i,j,\ell}$'s such that equation (11) holds. To see this, observe that if such an adversary $\mathcal{A}$ exists, then there exists such an adversary $\mathcal{A}'$ that differs from $\mathcal{A}$ only in receiving all the $\{b_\ell^j\}_{j\notin\mathcal{A}}$ ($\mathcal{A}'$ simply discards these and acts as $\mathcal{A}$). However, the b_ℓ^j's are random and independent of $\Delta_\mathcal{H}$, but $\mathcal{A}'$ learns $EC_\mathcal{A} \cdot \Delta_\mathcal{H}$ (by computing $EC_\mathcal{A} \cdot \Delta_\mathcal{H} = E_\mathcal{A} + \bigoplus_{j\notin\mathcal{A}} \bigoplus_\ell b_\ell^j \cdot \chi_\ell^j \cdot e_{\mathcal{A},j,\ell}$), a contradiction. □

We remark that Lemma 8 requires that $\Delta_\mathcal{H}$ is computationally indistinguishable from random, which we show in the main proof below. We prove that for a malicious adversary deviating from the protocol without causing the honest parties to abort, both the view of the adversary and the introduced errors are computationally independent of Δ, the $\bar{\Delta}^i$'s for $i \notin \mathcal{A}$, and the shared b_ℓ's.

More formally, we show that there exists a simulator $\mathcal{S}_{BS}$ in the ideal world such that for every polynomial distinguisher $\mathcal{D}$ that receives $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})$ cannot distinguish between a protocol execution in the $\mathcal{F}_{\text{TinyOT}}$-Hybrid model and a simulated execution in the ideal world.

Proposition 1. *Let $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})_\mathcal{I}$ be the simulated adversary's view and the output of the functionality in the ideal world and similarly $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})_\mathcal{R}$ is the adversary's view and the output of the protocol execution in the $\{\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{commit}}, \mathcal{F}_{\text{TinyOT}}\}$-Hybrid model. There exists a simulator $\mathcal{S}_{\mathcal{BS}}$ that interacts with the ideal functionality F_{BS} in the ideal world, such that for any polynomial time distinguisher $\mathcal{D}$, it holds that*

$$\mathcal{D}((\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})_\mathcal{I}) \overset{C}{\equiv} \mathcal{D}((\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})_\mathcal{R}) \tag{13}$$

Proof. We first describe the simulator $\mathcal{S}_{BS}$ that interacts with the ideal functionality F_{BS} in the ideal world. The simulator simulates the messages of the honest parties by sending random messages in Steps 2-9 of the protocol, while extracting the set of indices S and corresponding errors from the messages sent by the corrupt parties. The guess to the challenge is then extracted from error in the committed values given by the corrupt parties to $\mathcal{F}_{\text{commit}}$, and opening the commitment is simulated by XORing the guess and the leakage received from the functionality. The formal description of $\mathcal{S}_{\mathcal{BS}}$ is given in Figure 14

[13] We slightly simplified the proof by omitting here the possibility for the adversary to cheat in Step 2 on the r_h's, cf. Remark 3.

Functionality $\mathcal{F}_{\mathrm{BS}}$

The functionality runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$. The adversary is assumed to have statically corrupted a subset $\mathcal{A}$ of the parties.

Input: The functionality receives from all the parties their additive shares of b_ℓ.
Adversary's additional input: The functionality allows the adversary to choose

- Its shares of Δ, i.e., $\widehat{\Delta}^i$ for $i \in \mathcal{A}$.
- A subset $S \subset [L] \times \mathcal{H}$ of size $|S| = c$ and corresponding errors $0 \neq \mathtt{E}_{\ell,j} \in \{0,1\}^{\mathrm{sec}}$ for $(\ell, j) \in S$.[a]

Challenge: The functionality chooses random elements $\chi_\ell^j \in \{0,1\}^{\mathrm{sec}}$ for $(\ell, j) \in S$ and gives to the adversary, and the adversary needs to guess $\bigoplus_{(\ell,j)\in S} b_\ell^j \cdot \chi_\ell^j \cdot \mathtt{E}_{\ell,j}$. If the adversary guessed correctly then the functionality continues to the output. Otherwise, the honest parties abort. In both cases, the leakage is given to the adversary.
Leakage: $\bigoplus_{(\ell,j)\in S} b_\ell^j \cdot \chi_\ell^j \cdot \mathtt{E}_{\ell,j}$.
Output: The functionality chooses a random $\Delta \in \{0,1\}^k$, and gives the adversary its shares of $b_\ell \cdot \Delta$. The functionality waits for `abort` or `proceed` from the adversary. Upon receiving `proceed`, the functionality gives the output to the honest parties:

1. Additive shares of Δ,
2. Shares of $b_\ell \cdot \Delta$ with the corresponding errors $\mathtt{E}_{\ell,i}$ for $(i, \ell) \in S$, i.e., $(b_\ell \cdot \Delta)_i + b_\ell^i \mathtt{E}_{\ell,i}$,

[a] We have simplified the functionality/proof, since the adversary may also cheat on the r_h's, which causes leakage on the bits of C; cf. Remark 3.

Figure 13. The ideal functionality for $\langle b \cdot \Delta \rangle \leftarrow \mathsf{Bit} \times \mathsf{String}_{\langle\Delta\rangle}([b])$

Simulator $\mathcal{S}_{BS}$

Setup Simulation: Receives the input shares $\{b_\ell\}_{\ell \in L}$ for TinyOT from the corrupt parties and simulates their output of TinyOT ($\mathbf{k}^{i,j}$, $\bar{\Delta}^i$, $\mathbf{m}^{j,i}$).
Protocol Simulation: 1. Receives input and randomness of $\mathcal{A}$.

2. Sends random messages to the adversary for each $u_{i,j,\ell}$ with $i \notin \mathcal{A}, j \in \mathcal{A}$.
3.
   - Receives from the adversary $\widehat{u}_{i,j,\ell}$ for $i \in \mathcal{A}, j \notin \mathcal{A}$ for every $\ell \in L$.
   - Computes $\widehat{\Delta}_{\mathcal{A}}$ to be the element that appears the most times in the multiset
   $$\{\Sigma_{i\in\mathcal{A}} \left(\hat{u}_{i,j,\ell} \oplus \mathsf{PRG}(\mathbf{k}_\ell^{i,j}) \;\oplus\; \mathsf{PRG}(\mathbf{k}_\ell^{i,j} \;\oplus\; \bar{\Delta}^i)\right)\}_{\ell\in L, j\in\mathcal{H}} \tag{12}$$
   I.e., the $\Delta_{\mathcal{A}} = \Sigma_{i\in\mathcal{A}} \widehat{\Delta}^i$ that results in the least amount of errors.
   - For every $\ell \in L$, computes the error $e_{\mathcal{A},j,\ell} = \Sigma_{i\in\mathcal{A}} \widehat{u}_{i,j,\ell} \oplus u_{\mathcal{A},j,\ell}$, where $u_{\mathcal{A},j,\ell}$ is computed using $\widehat{\Delta}_{\mathcal{A}}$ and the simulated keys from TinyOT. The subset S is the set of indices where $e_{\mathcal{A},j,\ell} \neq 0$.
4. $\mathcal{S}_{BS}$ gives the chosen subset S and errors $\mathtt{E}_{\ell,j} := e_{\mathcal{A},j,\ell}$ for $(i, \ell) \in S$ to the functionality.
5. $\mathcal{S}_{BS}$ simulates $\mathcal{F}_{\mathrm{rand}}$ as follows: receive random elements for the challenge indices, $\{\chi_\ell^j\}_{(j,\ell)\in S}$, from the functionality, then generates random elements for the rest of the indices, $\{\chi_\ell^j\}_{(j,\ell)\in([L]\times[n])\setminus S}$.
6. Sends to the adversary random messages $\widetilde{C}_i$ for $i \notin \mathcal{A}$, receives $\widetilde{C}_i$ for $i \in \mathcal{A}$ and computes $C = \bigoplus_{i=1}^n \widetilde{C}_i$.[a]
7. Simulates $\mathcal{F}_{\mathrm{commit}}$ by receiving $\widehat{\mathcal{Z}}_i$ of corrupt parties.
8. $\mathcal{S}_{BS}$ computes $E_{\mathcal{A}}$ – observe that $E_{\mathcal{A}} = \bigoplus_{i\in\mathcal{A}} (\widehat{\mathcal{Z}}_i \oplus \mathcal{Z}_i) = \bigoplus_{i\in\mathcal{A}} \widehat{\mathcal{Z}}_i \oplus \bigoplus_{i\in\mathcal{A}} \mathcal{Z}_i$, where $\mathcal{Z}_i$ is what P_i should have committed to according to its TinyOT input and output, the messages it sent and received, and its $\widehat{\Delta}^i$. Therefore, $\bigoplus_{i\in\mathcal{A}} \mathcal{Z}_i$ can be computed by the simulator, assuming $\widehat{\Delta}_{\mathcal{A}}$ was computed correctly.
9. $\mathcal{S}_{BS}$ inputs $E_{\mathcal{A}}$ to the functionality as the challenge answer, and receives the leakage $\bigoplus_{(\ell,j)\in S} b_\ell^j \cdot \chi_\ell^j \cdot \mathtt{E}_{\ell,j}$.
10. $\mathcal{S}_{BS}$ simulates the opening the commitments by summing the leakage received from the functionality and the adversary's guess $E_{\mathcal{A}}$, i.e., opens the value $E_{\mathcal{A}} \oplus \bigoplus_{(\ell,j)\in S} b_\ell^j \cdot \chi_\ell^j \cdot \mathtt{E}_{\ell,j}$.

[a] If $\mathcal{A}$ tries to cheat here (which can be detected by $\mathcal{S}_{BS}$ by calculating), then the simulator outputs a random element in Step 10 and sends `abort` to the functionality. By Lemma 8 this is computationally indistinguishable from a real execution.

Figure 14. The simulator $\mathcal{S}_{BS}$

Distinguisher $\mathcal{D}'$

Input: $(\mathfrak{D} = \{\mathfrak{D}^i_1, \ldots, \mathfrak{D}^i_L\}_{i\in\mathcal{H}}, \mathfrak{d} = \{\mathfrak{d}^i_1, \ldots, \mathfrak{d}^i_L\}_{i\in\mathcal{H}})$, where $(\mathfrak{D} = \mathfrak{G}, \mathfrak{d} = \mathfrak{K})$ or $(\mathfrak{D} = \mathfrak{R}, \mathfrak{d} = \mathfrak{r})$. Additionally, $\mathcal{D}'$ receives the inputs of the honest parties to the protocol on which $\mathcal{D}$ can distinguish.

Setup: Sends keys given by tinyOT by sending the adversary $\mathfrak{d}$. In the reverse direction (corrupt party acts as sender), generates random $\bar{\Delta}^i$ and random keys $\mathbf{k}^{i,j}$ and sends to the adversary. If both parties are honest runs/simulates TinyOT protocol as usual.

Protocol:
1. Receives the input and randomness of the adversary.
2. Chooses random $\Delta^i \in \{0,1\}^k$ for every $i \notin \mathcal{A}$.
3. Sends $u_{i,j,\ell} = \mathsf{PRG}(\mathfrak{d}^{i,j}_\ell) + \mathfrak{D}^{i,j}_\ell + \Delta^i$ to $\mathcal{A}$ for every $i \in \mathcal{H}, j \in \mathcal{A}$.
4. Receives (possibly erroneous) $\widehat{u}_{i,j}$'s from the corrupt parties, and computes $\widehat{\Delta}_\mathcal{A}$ in the same manner as the simulator does (the $\Delta_\mathcal{A}$ that results in the least amount of errors). Recall that $\mathtt{E}_{\ell,i} := \Sigma_{i\in\mathcal{A}}\widehat{u}_{i,j,\ell} \oplus u_{\mathcal{A},j,\ell}$.
5. Calls F_{rand} to generate χ^j_ℓ's and computes C,
6. Sends to the adversary random messages $\widetilde{C}_i$'s for $i \notin \mathcal{A}$ such that $C = \bigoplus_{i=1}^n \widetilde{C}_i$.
7. Receives $\widehat{C}_i$'s of corrupt parties.
8. Calls $\mathcal{F}_{\text{commit}}$ with random messages $\widetilde{\mathcal{Z}}_i$'s of honest parties that sum to $\bigoplus_{i\notin\mathcal{A}} \widetilde{\mathcal{Z}}_i$.
9. Receives committed $\widehat{\mathcal{Z}}_i$'s of corrupt parties.

Output: The committed $\widetilde{\mathcal{Z}}_i$'s are opened.
1. If $\bigoplus_{i\in[n]} \widetilde{\mathcal{Z}}_i \neq 0$ then abort is output.
2. If $\bigoplus_{i\in[n]} \widetilde{\mathcal{Z}}_i = 0$ the output of an honest party P_i is set to be Δ^i and $\oplus_{j\neq i}(\mathbf{x}_{\ell_1}^{i,j} \oplus \mathbf{x}_{\ell_2}^{i,j}) \oplus b^i_\ell \Delta^i$, where for $j \in \mathcal{A}$
   - $\mathbf{x}_{\ell_1}^{i,j} = \mathsf{PRG}(\mathbf{k}^{j,i} + b^i \bar{\Delta}^j) + b^i \cdot \hat{u}_{j,i,\ell}$, and
   - $\mathbf{x}_{\ell_2}^{i,j} = \begin{cases} b^j_\ell = 0 & \mathsf{PRG}(\mathfrak{d}^{\mathrm{i,j}}_\ell) \\ b^j_\ell = 1 & \mathfrak{D}^{\mathrm{i,j}}_\ell \end{cases}$

Figure 15. The Distinguisher $\mathcal{D}'$

We next show that if there exists a distinguisher $\mathcal{D}$ that distinguishes between $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})_\mathcal{R}$ in an execution of the protocol in the $TinyOT-Hybrid$ model and $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})_\mathcal{I}$ in a simulated execution of the protocol in the ideal world, then there exists a distinguisher $\mathcal{D}'$ that breaks the key-correlation-robust PRG assumption. By standard hybrid arguments, it is enough to show that $\mathcal{D}'$ distinguishes between receiving $\mathfrak{G} = \{\mathsf{PRG}(k^{i,j}_1), \ldots, \mathsf{PRG}(k^{i,j}_L)\}_{i\in\mathcal{H},j\in\mathcal{A}}$ and keys $\mathfrak{K} = \{k^{i,j}_1 \oplus \bar{\Delta}^i, \ldots, k^{i,j}_L \oplus \bar{\Delta}^i\}_{i\in\mathcal{H},j\in\mathcal{A}}$, for fixed global random $\bar{\Delta}^i$'s, or receiving random strings of the same length $\mathfrak{R} = \{R^{i,j}_1, \ldots, R^{i,j}_L\}_{i\in\mathcal{H},j\in\mathcal{A}}$ and $\mathfrak{r} = \{r^{i,j}_1, \ldots, r^{i,j}_L\}_{i\in\mathcal{H},j\in\mathcal{A}}$. We assume that $\mathcal{D}'$ knows the input shares $\{b^j_\ell\}_{\ell\in L,j\in[n]}$ on which $\mathcal{D}$ can distinguish.

Assume that $\mathcal{D}$ exists, then $\mathcal{D}'$ builds $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})$ as follows: it sets the adversary's output of TinyOT (as receiver) to be $\mathfrak{r}$ / $\mathfrak{K}$, then builds the adversary's view, where the $u_{i,j,\ell}$'s are computed using $(\mathfrak{G}, \mathfrak{K})$ / $(\mathfrak{R}, \mathfrak{r})$. The output of the honest parties is computed by running the protocol with the adversary. The distinguisher $\mathcal{D}'$ feeds this to $\mathcal{D}$, and returns as it does.

The formal algorithm of distinguisher $\mathcal{D}'$ is given in Figure 15. Thus, it remains to show that if $\mathfrak{D} = \mathfrak{G}$ and $\mathfrak{d} = \mathfrak{K}$ then $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})$ distributes as a real protocol execution in the *TinyOT-hybrid* model while if $\mathfrak{D} = \mathfrak{R}$ and $\mathfrak{d} = \mathfrak{r}$ then $(\mathsf{view}_\mathcal{A}, \widehat{\mathsf{out}})$ distributes as a simulated protocol execution in the ideal world. First note that the leakage distribution is computationally indistinguishable between a real protocol execution in the *TinyOT-hybrid* model and a simulated execution in the ideal world. This is because the leakage depends only on the errors added by the adversary (and the honest parties' inputs, which are distributed the same in both worlds), and this error can only depend on the adversary's view at this stage (Step 2a), which is only the TinyOT output and the messages sent by the honest parties in Step 2a. The latter differs only in using $\mathfrak{G}$ and $\mathfrak{R}$, respectively, so computationally indistinguishable under the key-correlation PRG assumption.

Claim. If $\mathfrak{D} = \mathfrak{G}$ and $\mathfrak{d} = \mathfrak{K}$ then, except with negligible probability, $(\mathsf{view}_{\mathcal{A}}, \widehat{\mathsf{out}})$ distributes as a real protocol execution in the *TinyOT-hybrid* model.

Proof. If $\mathfrak{D} = \mathfrak{G}$ and $\mathfrak{d} = \mathfrak{K}$ then $\mathcal{D}'$ carries out the protocol with $\mathcal{A}$ exactly as in the real world, assuming the output of TinyOT between honest P_i and corrupt P_j are the keys $k_\ell^{i,j} \oplus b_\ell^j \bar{\Delta}^i$ (i.e., the global offset of P_i is $\bar{\Delta}^i$). Note that these keys are random (since $k_\ell^{i,j}$ are random), and hence this matches the desired distribution of TinyOT keys. □

Claim. If $\mathfrak{D} = \mathfrak{R}$ and $\mathfrak{d} = \mathfrak{r}$ then, except with negligible probability, $(\mathsf{view}_{\mathcal{A}}, \widehat{\mathsf{out}})$ distributes as a simulated protocol execution in the ideal world.

Proof. If $\mathfrak{D} = \mathfrak{R}$ and $\mathfrak{d} = \mathfrak{r}$ then it is not hard to see that the simulated output of tinyOT (with $\mathcal{A}$ as receiver) and the messages received by $\mathcal{A}$ in Step 2 are completely random (XOR with random is random), matching the messages sent by $\mathcal{S}_{BS}$. In case the adversary does not send errenous $\widehat{C_i}$, the opening of the commitments is $E_{\mathcal{A}} + \bigoplus_{j \notin \mathcal{A}} \bigoplus_\ell b_\ell^j \cdot \chi_\ell^j \cdot e_{\mathcal{A},j,\ell}$, i.e., matches the simulation of $\mathcal{S}_{BS}$. In the case that $EC_{\mathcal{A}} \neq 0$, i.e., the adversary sent errenous $\widehat{C_i}$, then the opening of the commitments is $E_{\mathcal{A}} + \bigoplus_{j \notin \mathcal{A}} \bigoplus_\ell b_\ell^j \cdot \chi_\ell^j \cdot e_{\mathcal{A},j,\ell} + EC_{\mathcal{A}} \Delta_{\mathcal{H}}$ and the simulation by $\mathcal{S}_{BS}$ is a random element and abort. By Lemma 8, except with neg. probability, these are indistinguishable.

We remark that the above assumes that the $\Sigma_{i \in \mathcal{A}} \widehat{\Delta}^i$ of the $\widehat{\Delta}^i$'s chosen by the adversary (and used in the adversary's commitments) match the $\widehat{\Delta}_{\mathcal{A}}$ given to the functionality by the simulator. The simulator chooses the $\widehat{\Delta}_{\mathcal{A}}$ that results in the least amount of errors. If the adversary attempts to guess the challenge for a subset S' corresponding to a different $\Delta'_{\mathcal{A}} = \Sigma_{i \in \mathcal{A}} \widehat{\Delta}^i \neq \widehat{\Delta}_{\mathcal{A}}$, then $|S'| \gg s$, so it is not hard to see that the protocol aborts with overwhelming probability.

For the output, notice that the errors introduced from the $\hat{u}_{j,i\ell}$s match the errors given by the functionality. Apart from that, it only remains to show that the sum of the shares (excluding the errors) of all the parties (including the shares the corrupt parties should output) for each $\ell \in L$ is $b_\ell \Delta$. The computation works the same as in the protocol, except that $\mathsf{PRG}(r_\ell^{i,j} + \bar{\Delta}^i)$ are replaced by $R_\ell^{i,j}$ whenever $i \in \mathcal{A}, j \in \mathcal{H}$. □

This concludes the proof of Proposition 1. □

We next show that passing the challenge in functionality $\mathcal{F}_{\mathrm{BS}}$ by the adversary implies that, except with negligible probability, $c < s$. I.e., the number of errors is less than the statistical security parameter.

Claim. If the adversary passes the challenge, then except with negligible probability, $c < s$.

Proof. There there are 2^c possible combinations of the b_ℓ^j's, $(\ell, j) \in S$, each resulting in a possible random sum (possibly with duplicates) in $\{0,1\}^{\mathsf{sec}}$ (where sec is 128 or 256, so significantly larger than s). To maximize its probability of guessing correctly, the adversary should guess the sum $\bigoplus_{(\ell,j) \in S} b_\ell^j \cdot \chi_\ell^j \cdot \mathtt{E}_{\ell,j}$ that appears the most times (for different combinations of b_ℓ^j). Denote by M the maximal number of times that the same possible sum appears (for different combinations of b_ℓ^j). Thus, the probability of the adversary correctly guessing the sum is at most $\frac{M}{2^c}$. If $c \geq s$, then $\frac{M}{2^c} > 2^{-s}$ (equivalently, $M > 2^{c-s}$) happens with negligible probability, because the sums are in $\{0,1\}^{\mathsf{sec}}$ and $\mathsf{sec} \gg s$. □ □

Since $|\mathcal{H}| \geq s$, it follows that except with neg. probability, (if the honest parties do not abort) the leakage contains less than $|\mathcal{H}|$ of the b_ℓ^j's of the honest parties. In particular, this implies that there is no leakage on any of the b_ℓ's (and sums of b_ℓ's). Additionally, we see that except with neg. probability, there is no leakage on $\varDelta$ and it remains completely random. We note also that the errors introduced by the adversary cannot, therefore, be dependent on the b_ℓ's, $\varDelta$, or any of the $\bar{\varDelta}^i$'s of the honest parties.[14] If the adversary fails the challenge then the honest parties abort. As this protocol is run at the offline phase, before the inputs are given, in case of abort by the honest parties at this stage is not a violation of security regardless of the leakage.

Remark 3. Each party P_j runs Step 2 and additional sec times with random r_h^j's; denote $r_h = \bigoplus_{j=1}^n r_h^j$. These random r_h's are used to mask the bits of C. We note that the adversary may also try to learn information on the b_ℓ's indirectly, by adding errors to the $u_{i,j}$'s in Step 2 for these bits. Each successful cheating leaks, in this case, a single r_h (or a single sum of r_h's), and, consequently, a single bit of C (or a single sum of bits of C).

However, using similar considerations as in the main proof, the number of errors is bounded by the statistical security, and furthermore, the possibly leaked bits are chosen before C and the χ's are known to the adversary. We next prove that except with negligible probability, the shared bits b_ℓ's remain random, even leaking up to s bit of C (without leaking anything else).

The full proof of security of our protocol, i.e., modifying the functionality to accommodate also leakage on the bits of C, is slightly tedious, and deferred to the full version.

Claim. If $|\mathcal{H}| \geq s$ and up to s bits of C are leaked and nothing else, then except with negligible probability, $\forall \ell, b_\ell$ is computationally random to the adversary.

Proof. For simplicity, we assume that exactly s bits of C are leaked. Following the main proof, we may assume that except with negligible probability, without the leakage of the bits of C, all the bits b_ℓ^j for $j \notin \mathcal{A}$ are computationally random. So we now show that the leakage from the bits of C does not leak any information on any shared b_ℓ.

Note that since the adversary knows its own shares, this is equivalent to no leakage on any $\bigoplus_{j \notin \mathcal{A}} b_\ell^j$. Since the χ_ℓ^j's are known to the adversary, we may look at each leaked bit of C as a Boolean sum of the b_ℓ^j's for $j \notin \mathcal{A}$, where b_ℓ^j is in the sum if the corresponding bit in χ_ℓ^j is 1.

Denote by L the total number of unknown bits per party, i.e., the different b_ℓ^j for fixed j (note that this number is the same for all the parties). We now observe the following:

1. There are a total of $L \cdot |\mathcal{H}| \geq L \cdot s$ unknown independent random bits.
2. Thus, there are a total of $\geq 2^{L \cdot s}$ different possible sums of these bits.
3. The leakage from the bits of C is s independently random sums,[15] so using linear combinations, there are a total of 2^s sums known to the adversary.
4. There are a total of L unknown b_ℓ's, so there are 2^L *forbidden sums*, i.e., sums that are either b_ℓ or sums of b_ℓ's and learning them breaks security.
5. Thus, the probability of a single random sum being forbidden is $\leq \frac{2^L}{2^{L \cdot s}}$.
6. By taking a union bound on all the known sums, the probability of a known sum being forbidden is therefore $\leq \frac{2^L \cdot 2^s}{2^{L \cdot s}}$, which is negligible (i.e., $\frac{2^L \cdot 2^s}{2^{L \cdot s}} \ll 2^{-\sec}$) for standard parameters s and L.

[14] This is clearly true in the ideal world. Therefore, in the real world, even if such a dependence exists, no polynomial time distiguisher can "see" this dependence.

[15] Since the cheating is done before C and the χ's are known, we may assume that a single leaked bit of C is a random sum.

□

Remark 4. To show the necessity of using different random coefficients χ_ℓ^j for each share instead of using the same coefficient for all the shares of the same shared bit (i.e., as done in [BLN+15, KOS16, KPR18]), we show here a selective failure attack that succeeds with probability $\frac{1}{2}$ and leaks a shared bit:

Assume the same coefficient is used for all the shares of the same shared bit, i.e., $\chi_\ell^1 = \chi_\ell^2 = \cdots = \chi_\ell^n$. Applying to Equation (11), we see that in order for the check in Step 10 to pass, it should hold that:

$$0 = E_{\mathcal{A}} + \bigoplus_{j \notin \mathcal{A}} \bigoplus_{\ell} b_\ell^j \cdot \chi_\ell^j \cdot e_{\mathcal{A},j,\ell} + EC_{\mathcal{A}} \Delta_{\mathcal{H}} = E_{\mathcal{A}} + \bigoplus_{\ell} \chi_\ell \left(\bigoplus_{j \notin \mathcal{A}} b_\ell^j \cdot e_{\mathcal{A},j,\ell} \right) + EC_{\mathcal{A}} \Delta_{\mathcal{H}}. \tag{14}$$

Now, the adversary selects a shared bit $b_{\bar{\ell}}$ that it wants to learn, and sets $e_{\mathcal{A},j,\bar{\ell}} = 1$ for $j \notin \mathcal{A}$, $e_{\mathcal{A},j,\ell} = 0$ for $\ell \neq \bar{\ell}$, $EC_{\mathcal{A}} = 0$, and $E_{\mathcal{A}} = \chi_{\bar{\ell}}$ (note that $E_{\mathcal{A}}$ is chosen at Step 9, when the χ's are already known to the adversary). Substituting in Equation (14) above, we see that the check passes if

$$0 = \chi_{\bar{\ell}} \oplus \chi_{\bar{\ell}} \left(\bigoplus_{j \notin \mathcal{A}} b_{\bar{\ell}}^j \right). \tag{15}$$

Thus, if $\bigoplus_{j \notin \mathcal{A}} b_{\bar{\ell}}^j = 0$ the honest parties abort while if $\bigoplus_{j \notin \mathcal{A}} b_{\bar{\ell}}^j = 1$ the check passes. Since the adversary knows $b_{\bar{\ell}}^j$ for $j \in \mathcal{A}$, and learns $\bigoplus_{j \notin \mathcal{A}} b_{\bar{\ell}}^j$ from the behaviour of the honest parties, it can now compute $b_{\bar{\ell}} = \bigoplus_{j=1}^{n} b_{\bar{\ell}}^j$.

In other words, the adversary learns $b_{\bar{\ell}}$ in any case, but with probability $\frac{1}{2}$ the honest parties do not abort, breaking security (recall that this protocol is executed in offline phase, before the inputs are known, so learning the shared bits violates security only if it doesn't cause abort).

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070, FA8750-19-C-0502 and HR001120C0085, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via Contract No. 2019-1902070006, by the FWO under an Odysseus project GOH9718N, and by CyberSecurity Research Flanders with reference number VR20192203. Eduardo Soria-Vazquez was supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM). Aner Ben-Efraim and Eran Omri were supported by ISF grant 152/17, and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

AHI11. Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *ICS 2011: 2nd Innovations in Computer Science*, pages 45–60, Tsinghua University, Beijing, China, January 7–9, 2011. Tsinghua University Press.

App13. Benny Applebaum. Garbling XOR gates "for free" in the standard model. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 162–181, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

BDP$^+$18. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. KangarooTwelve: Fast hashing based on Keccak-p. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, volume 10892 of *Lecture Notes in Computer Science*, pages 400–418, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany.

BFKL94. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

BK03. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

BLN$^+$15. 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. Cryptology ePrint Archive, Report 2015/472, 2015. `http://eprint.iacr.org/2015/472`.

BLO16. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 578–590, Vienna, Austria, October 24–28, 2016. ACM Press.

BLO17. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Efficient scalable constant-round MPC via garbled circuits. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 471–498, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

BRS03. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer, Heidelberg, Germany.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

CDE$^+$18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in*

Computer Science, pages 572–590, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

EKM17. Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 486–514, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

FKOS15. Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 711–735, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

GKL90. Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 146–162, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

HOSS18a. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 86–117, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

HOSS18b. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 3–33, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 598–628, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

IK02. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP 2002: 29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256, Malaga, Spain, July 8–13, 2002. Springer, Heidelberg, Germany.

KOS15. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 724–741, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

LOS14. Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 495–512, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

LPSY15. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 319–338, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

LSS16. Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 554–581, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

Mas69. James Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

Mat94. Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397, Lofthus, Norway, May 23–27, 1994. Springer, Heidelberg, Germany.

Mau02. Ueli M. Maurer. Indistinguishability of random systems. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

NIS16. NIST National Institute for Standards and Technology. SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash, 2016. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf.

NIS18. NIST National Institute for Standards and Technology. Recommendation for key derivation through extraction- then-expansion rev.1, 2018. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-56c.pdf.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

OSV20. Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 254–283, San Francisco, CA, USA, February 24–28, 2020. Springer, Heidelberg, Germany.

RB89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, WA, USA, May 15–17, 1989. ACM Press.

WRK17a. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 21–37, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

WRK17b. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 39–56, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

YWZ19. Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. Cryptology ePrint Archive, Report 2019/1104, 2019. https://eprint.iacr.org/2019/1104.