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.

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 [15, 28, 25, 26] and SPD$\mathbb{Z}_{2^k}$ [13, 36], for the case of linear secret-sharing based MPC, and BMR-based protocols [22, 39, 40]. However, when we increase the number of parties this practicality drops off.

Secret-sharing based protocols [19, 37, 7, 14, 15], 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 [41] in the two-party setting and later generalized to the multiparty case by Beaver, Micali and Rogaway (BMR) [3], 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 [41] and then compute the garbling function via an n-party actively secure MPC system. The resulting 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 [6] to present a *passively secure* BMR-based protocol whose evaluation is independent of the number of parties and such that the

[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. [15].

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 [5] 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 [27].

Another relevant large-scale, garbled-circuit based protocol is that proposed by Hazay, Orsini, Scholl and Soria-Vazquez [21]. 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 [20], 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 [27].

Obtaining Free-XOR. Our construction takes inspiration from the work of Ben-Efraim et al. [6], 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 [1] in order to replace the random oracle with standard cryptographic assumptions in two-party, free-XOR garbled circuits. We would like

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

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 [38, 39] (referred as WRK in the rest of the paper) and more recently by Yang, Wang and Zhang [40]. However, while their preprocessing phase is explicitly based on TinyOT-like protocols [33, 17], 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) [22]. 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. [16] and instantiate the cryptosystem with concatenated codes (see the full version). 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. [40] 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 [39]. Setting the statistical security parameter to 40, as in [39], we report a running time for AES-128 of 1.72 sec (c.f. Table 2 in Section 6), compared to 1.87 sec in a LAN setting and 2.3 sec in a WAN setting reported in WRK [39] 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. [6] –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 [6] we see that, assuming a circuit consisting solely of AND gates, our protocol is roughly six times slower than that of [6]. 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. In the full version we also provide an estimation of the overall complexity of our protocols.

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 [3], which has been recently improved by a sequence of works [5, 22, 29, 30, 39] both in the case of passive and active security. In this paper we follow the approach described in [5, 22].

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 $\Delta^i \in \mathbb{F}_2^k$ 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 $\mathbf{k}^i_{w,1} = \mathbf{k}^i_{w,0} \oplus \Delta^i$ 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 $\mathbf{k}^i_{w,0} = \mathbf{k}^i_{u,0} \oplus \mathbf{k}^i_{v,0}$, $\mathbf{k}^i_{w,1} = \mathbf{k}^i_{w,0} \oplus \Delta^i$ 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:

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

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 Δ^j, we further need F to be a circular 2-correlation robust PRF [22].

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 $(\mathbf{k}^1_{u,\epsilon_u}, \ldots, \mathbf{k}^n_{u,\epsilon_u})$

and $(\mathbf{k}^1_{v,\epsilon_v},\ldots,\mathbf{k}^n_{v,\epsilon_v})$, it is possible to decrypt a single row of AND garbled gates obtaining $(\mathbf{k}^1_{w,\epsilon_w},\ldots,\mathbf{k}^n_{w,\epsilon_w})$. 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 [29] uses the actively secure SPDZ protocol [15] to create an *authenticated* secret-sharing of Equation (1), Hazay et al. [22] 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 $\Delta^j \cdot \big((\lambda_u \oplus \alpha)\cdot(\lambda_v \oplus \beta) \oplus \lambda_w\big)$ 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 [1], 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

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

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

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

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 [12]. 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 [1] and use the terminology of randomized encodings for garbled circuits [23, 24].

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 [32]:

– 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 [1, 2, 4, 9, 11] 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, [1]). *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 [18, 10] 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 [1], 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* (KeyGen^1_τ, Enc^1_τ, Dec^1_τ) *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 [34, 8].

Definition 5 (XOF-Based LPN Encryption). *Let* $\mathtt{m}, k, \ell = \mathsf{poly}(\mathsf{sec})$ *be three integers and* $\mathcal{K}, \mathcal{C}, \mathcal{M}$ *as in* Definition 4*. Let* $\tau \in (0, 1/2)$ *and* $G \in \mathbb{F}_2^{\ell \times \mathtt{m}}$ *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. [16], 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:

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

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*

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

- $\mathsf{Dec}((C, \mathbf{c}), (\mathbf{k}_u, \mathbf{k}_v))$*: 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

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

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 Δ), 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:

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

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 $\Delta, \sigma(\Delta)$ and $\Delta \oplus \sigma(\Delta)$. 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, the proof of which is given in the full version.

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.*

In our construction, the permutation σ will be chosen to map $(\delta_0, \ldots, \delta_{k-1}) \in \mathbb{F}_2^k$ to $(\delta'_0, \ldots, \delta'_{k-1})$, 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:

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

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:

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

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 $\Delta \in \mathbb{F}_2^k$ is of the form $(1, \Delta', 0)$.

Let $\Delta = (1, \Delta', 0)$ with $\Delta' \leftarrow \mathbb{F}_2^{k-2}$ 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 Δ, $\sigma(\Delta)$, and $\Delta \oplus \sigma(\Delta)$ 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 $\Psi(b \cdot \Delta, b)$. With this understanding we define the following oracles:

$$\begin{aligned} \mathsf{Real}_\Delta^\sigma : (\mathbf{k}, \alpha, \beta, \mathbf{m}, b, \mathsf{nonce}) \longmapsto& \\ &\mathsf{Enc}_\tau^{\mathsf{XOF}}(\ (\mathbf{m} \oplus \Psi(b\cdot\Delta, b),\ \mathsf{nonce}),\ \mathbf{k} \oplus \alpha\cdot\Delta \oplus \beta\cdot\sigma(\Delta)\ ) \\ \mathsf{Fake}_\Delta^\sigma : (\mathbf{k}, \alpha, \beta, \mathbf{m}, b, \mathsf{nonce}) \longmapsto& (H(\mathsf{nonce}),\ \mathbf{c}), \qquad \mathbf{c} \leftarrow \mathcal{C}, \end{aligned}$$

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* $\mathsf{Real}_\Delta^\sigma$ *and* $\mathsf{Fake}_\Delta^\sigma$ *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.* $\mathsf{Real}_\Delta^\sigma \overset{c}{\equiv} \mathsf{Fake}_\Delta^\sigma$.

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

Proof. For the proof of this result, see the full version. □

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.

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 $\mathcal{F}^{\mathsf{Auth}}_{\mathsf{MPC}}$), the parties can compute a secret shared ciphertext $(C, [\mathbf{c}])$, where C is in the clear, using a double-key encryption scheme $\mathcal{E}^{\mathsf{XOF}}_{\tau}$ as described in Definition 6. Since both the generation and opening of the garbled circuit are done using an active secure MPC system,

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 $\mathbf{k}_{w,0}^i \leftarrow \mathcal{K}$ and call $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$ on $[\mathbf{k}_{w,0}^i] \leftarrow \mathsf{Input}(P_i)$.
   - Set $[\mathbf{k}_{w,0}] \leftarrow \bigoplus_{i\in[n]}[\mathbf{k}_{w,0}^i]$ 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}

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 $[\mathbf{k}_{0,w}^i]$, $[\mathbf{k}_{1,w}^i]$ 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 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, and can be found in the full version.

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 the full version) in the presence of a static, active adversary corrupting up to $n-1$ parties in the $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{Auth}}$-hybrid model.*

4.2 Evaluation

The protocol Π_{Evaluate}, given in the full version, 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.*

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.

5.1 Garbling

In this section we describe how to implement the $\mathcal{F}_{\text{Preprocessing}}^{n/c}$ functionality given in the full version. 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,

$$\Sigma_{i=1}^{n} \mathsf{Enc}_{\tau}^{\mathsf{XOF}}((\mathbf{m}^i, \mathsf{nonce}), \mathbf{s}^i) = \mathsf{Enc}_{\tau'}^{\mathsf{XOF}}((\Sigma_{i=1}^{n} \mathbf{m}^i, \mathsf{nonce}), \Sigma_{i=1}^{n} \mathbf{s}^i). \tag{6}$$

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

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

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 [31].

Lemma 2 (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

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

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

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

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 3. *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 example for $\mathsf{sec} = \{128, 256\}$ see the full version.

Using the above observations we define, in Figure 4, 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 the full version, 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]

Remark 1. Note that the way that the $\mathsf{Bit} \times \mathsf{String}$ operation is described in the full version, 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 4 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.

Compared with the evaluation phase of [22], 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 $\alpha \in \{0,1\}$):

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

[7] If the requirement does not hold, then this operation needs to be done using Tiny-OT directly as in [22]. Hence, this optimization is mainly for large-scale MPC.

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, the proof of which is given in the full version.

Theorem 5. *Let $\mathcal{E}^{\mathsf{XOF}}_\tau$ be a XOF-based two-key LPN encryption scheme with parameter τ. Let $\mathcal{F}_{\mathrm{BS}}$ be implemented by the* $\mathsf{Bit} \times \mathsf{String}$ *operation. The protocol $\Pi^{n/c}_{\mathsf{Garble}}$ described in Figure 4 UC-securely computes $\mathcal{F}^{n/c}_{\mathrm{Preprocessing}}$ 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 [22], we could prove Theorem 5 in the $\{\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathrm{TinyOT}}\}$-hybrid model, without the $n/c > s$ requirement.

5.2 Evaluation

The evaluation procedure is given in the full version. 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, the proof of which is given in the full version.

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

Our proof follows the blueprint of the online proof of Hazay et al. [22]. More concretely, after the description of the simulator, we show that the adversary can succeed in introducing errors in the garbled circuit only with negligible probability, so ruling out this possibility we show that the ideal and real executions are indistinguishable trough a reduction to the LIN-RK-KDM$^\sigma$ security of the LPN-based encryption scheme $\mathcal{E}^{\mathsf{XOF}}$. Although the general idea of the proof is similar to [22], in our proof we need to take care of our new method of garbling AND gates, and prove that if the adversary introduces some errors such that the some value is not correct during the evaluation, then the final checks will fail with overwhelming probability.

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 the full version and compare it with the recent work of Yang et al. [40].

[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.

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

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 [35]. 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 [34]. The third variant is based on the Kangaroo-12 XOF from [8], which is 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. We use concatenated codes as our error correcting code. While they are not the fastest or offer the highest rate, we can easily calculate the exact failure probability, unlike the alternatives such as LDPC codes. This makes selecting a code according to the LPN parameters convenient. The concatenated codes we use has a Reed-Solomon outer code and a general linear inner code. The details of concatenated codes and their concrete instantiation is presented in the full version. We set the decoding failure probability to 2^{-s}, and run experiments with $s = 40$ and 80. While finding the best error correcting code is not the goal of this work, we expect the performance to improve significantly when using a more efficient family of codes such as LDPC or quasi-cyclic LDPC.

Online Implementation Results. The expensive parts of the algorithms are the parts related to the evaluation of the garbled circuit; 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.

For the authenticated garbling (resp. unauthenticated garbling) variant of our algorithm, we obtained the run-times presented in Table 2 (resp. Table 3) with decryption failure $s = 40$. For equivalent runtimes when $s = 80$ see the full version of the paper. In these tables the security level refers to the security of the underlying LPN function. Observe that the choice of the underlying method to

	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 2. 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 the full version.

	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 3. 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 the full version.

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 [6], i.e. in the passive case, with no free-XOR, has a throughput of roughly 45 microseconds per *gate* (also with $s = 40$). 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 [6] 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 [39] that reports an online running time of 2.3 sec for AES with 128 parties in the WAN setting.

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

1. Applebaum, B.: Garbling XOR gates “for free” in the standard model. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 162–181. Springer, Heidelberg (Mar 2013)
2. Applebaum, B., Harnik, D., Ishai, Y.: Semantic security under related-key attacks and applications. In: Chazelle, B. (ed.) ICS 2011. pp. 45–60. Tsinghua University Press (Jan 2011)
3. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC. pp. 503–513. ACM Press (May 1990)
4. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (May 2003)
5. Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 578–590. ACM Press (Oct 2016)

6. Ben-Efraim, A., Lindell, Y., Omri, E.: Efficient scalable constant-round MPC via garbled circuits. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 471–498. Springer, Heidelberg (Dec 2017)
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988)
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V., Viguier, B.: KangarooTwelve: Fast hashing based on Keccak-p. In: Preneel, B., Vercauteren, F. (eds.) ACNS 18. LNCS, vol. 10892, pp. 400–418. Springer, Heidelberg (Jul 2018)
9. Black, J., Rogaway, P., Shrimpton, T.: Encryption-scheme security in the presence of key-dependent messages. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 62–75. Springer, Heidelberg (Aug 2003)
10. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (Aug 1994)
11. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
13. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 769–798. Springer, Heidelberg (Aug 2018)
14. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (Aug 2007)
15. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012)
16. Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 486–514. Springer, Heidelberg (Aug 2017)
17. Frederiksen, T.K., Keller, M., Orsini, E., Scholl, P.: A unified approach to MPC with preprocessing using OT. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 711–735. Springer, Heidelberg (Nov / Dec 2015)
18. Goldreich, O., Krawczyk, H., Luby, M.: On the existence of pseudorandom generators. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 146–162. Springer, Heidelberg (Aug 1990)
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)
20. Hazay, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 86–117. Springer, Heidelberg (Dec 2018)
21. Hazay, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: TinyKeys: A new approach to efficient multi-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 3–33. Springer, Heidelberg (Aug 2018)

22. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 598–628. Springer, Heidelberg (Dec 2017)
23. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st FOCS. pp. 294–304. IEEE Computer Society Press (Nov 2000)
24. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Widmayer, P., Ruiz, F.T., Bueno, R.M., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 244–256. Springer, Heidelberg (Jul 2002)
25. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 830–842. ACM Press (Oct 2016)
26. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 158–189. Springer, Heidelberg (Apr / May 2018)
27. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (Jul 2008)
28. Larraia, E., Orsini, E., Smart, N.P.: Dishonest majority multi-party computation for binary circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 495–512. Springer, Heidelberg (Aug 2014)
29. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant round multi-party computation combining BMR and SPDZ. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 319–338. Springer, Heidelberg (Aug 2015)
30. Lindell, Y., Smart, N.P., Soria-Vazquez, E.: More efficient constant-round multi-party computation from BMR and SHE. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part I. LNCS, vol. 9985, pp. 554–581. Springer, Heidelberg (Oct / Nov 2016)
31. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (May 1994)
32. Maurer, U.M.: Indistinguishability of random systems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 110–132. Springer, Heidelberg (Apr / May 2002)
33. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (Aug 2012)
34. 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
35. 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
36. Orsini, E., Smart, N.P., Vercauteren, F.: Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 254–283. Springer, Heidelberg (Feb 2020)
37. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC. pp. 73–85. ACM Press (May 1989)

38. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 21–37. ACM Press (Oct / Nov 2017)
39. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 39–56. ACM Press (Oct / Nov 2017)
40. Yang, K., Wang, X., Zhang, J.: More efficient MPC from improved triple generation and authenticated garbling. Cryptology ePrint Archive, Report 2019/1104 (2019), https://eprint.iacr.org/2019/1104
41. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)