Full-Threshold Actively-Secure Multiparty Arithmetic Circuit Garbling

Eleftheria Makri[1,2] Tim Wood[1,3]

[1] imec-COSIC, KU Leuven, Belgium.
[2] ABRR, Saxion University of Applied Sciences, The Netherlands.
[3] University of Bristol, UK.

Abstract. In this work, we show how to garble arithmetic circuits with full active security in the general multiparty setting, secure in the full-threshold setting (that is, when only one party is assumed honest). Our solution allows interfacing Boolean garbled circuits with arithmetic garbled circuits. Previous works in the arithmetic circuit domain focused on the two-party setting, or on semi-honest security and assuming an honest majority – notably, the work of Ben-Efraim (Asiacrypt 2018) in the semi-honest, honest majority security model, which we adapt and extend. As an additional contribution, we improve on Ben-Efraim's selector gate. A selector gate is a gate that given two arithmetic inputs and one binary input, outputs one of the arithmetic inputs, based on the value of the selection bit input. Our new construction for the selector gate reduces the communication cost to almost half of that of Ben-Efraim's gate. This result applies both to the semi-honest and to the active security model.

1 Introduction

Garbled circuits have been an indispensable cryptographic tool in the field of secure computation since the seminal work of Yao [Yao82]. From a theoretical point of view, garbled circuits are important as they provide the means by which we can construct *constant-round* secure computation protocols, originally only in the two-party setting, but later generalised to the multiparty setting, following the paradigm of Beaver et al. [BMR90]. In the two-party setting, garbled circuits are typically *Boolean* circuits executed between two asymmetric parties – a garbler and an evaluator. However, many secure computation problems require arithmetic operations to emulate integer arithmetic, which are inefficient to realise with a Boolean circuit (e.g., requiring 1000 AND gates for an addition mod p and 100000 AND gates for a multiplication mod p, for $p \approx 2^{128}$). Towards the goal of efficient constant-round computation of arithmetic circuits, one theoretical approach was given by Applebaum et al. [AIK11] and more recently a practical solution was proposed by Ball et al. [BMR16], in the two-party setting.

In this work we focus on *multiparty* arithmetic garbling. The work of Ben-Efraim [Ben18] was the first to explore multiparty garbling in the context of arithmetic circuits, and gave protocols secure in the presence of a passive adversary in the honest-majority setting. The goal of multiparty arithmetic garbling

protocols is the functionality $\mathcal{F}_{\mathsf{AC}}$ for computing an arithmetic circuit, given in Figure 1. This functionality is essentially the goal of all Multiparty Computations (MPC), but offers only security *with abort* instead of full *robustness*, in which honest parties can always obtain the correct output after the initial inputs are provided, or *fairness*, in which honest parties always receive the output if the corrupt parties receive it.

Functionality $\mathcal{F}_{\mathsf{AC}}$

Let $\mathcal{S}$ denote the ideal-world adversary and $A \subsetneq [n]$ the indexing set of corrupt parties.

Evaluate On input $(\mathsf{Evaluate}, C, \mathbf{x}^i)$ from each party P_i, or $\mathcal{S}$ if $i \in A$, if C is an arithmetic circuit over $\mathbb{F}_p$ and $\mathbf{x}^i \in \mathbb{F}_p^{t_i}$ for all i such that $\sum_{i=1}^{n} t_i$ is the arity of C, send $y := C(\mathbf{x}^1, \ldots, \mathbf{x}^n)$ to $\mathcal{S}$ and await a message Abort or OK from $\mathcal{S}$. If the message is OK, then send y to all honest parties and halt; otherwise, send the message Abort to all honest parties and halt.

Fig. 1: Functionality $\mathcal{F}_{\mathsf{AC}}$ for evaluating an arithmetic circuit, secure *with abort*.

In our approach, we allow a (limited) combination of arithmetic and Boolean circuits, as this appears to be desirable for many real-world applications. From the simplest motivating example that one can consider, such as the one of conditional summation that Ben-Efraim [Ben18] suggests, to the most complicated computations, such as evaluation of Machine Learning (ML) algorithms, a combination of arithmetic with Boolean gates is required to yield an efficient solution. Machine Learning as a Service is becoming increasingly popular, and when privacy concerns arise, secure computation solutions should be deployed. The most commonly used ML algorithms (e.g., Support Vector Machines (SVMs) and Neural Networks) contain one or more components that require linear operations – for which arithmetic operations are more appropriate – and one or more components that require non-linear operations, such as argmax or sign computation – where Boolean computation is best. Thus it seems sensible to attempt to support both types of gates to achieve efficient solutions to realistic applications.

1.1 Related Work

Our work combines the work of Ben-Efraim [Ben18], and Ball et al. [BMR16], and extends them in such a way as to achieve full-threshold active security by using recent actively-secure secret-sharing-based MPC to construct the circuit, a technique initiated by Lindell et al. [LPSY15]. In the work of Ball et al., which is based on some of the techniques discussed also in the work of Malkin et al. [MPs16], the authors propose a two-party arithmetic garbling scheme, secure in the presence of a semi-honest adversary, where the arithmetic takes place in a ring isomorphic to a cyclic group of primorial modulus. They show how

to use a Chinese Remainder Theorem (CRT) representation of the inputs (and intermediate values) of the circuit to achieve great performance gains over the straightforward conversion of ring elements to binary. In this approach, garbling of linear gates (e.g., addition and scalar multiplication) requires no communication and can be viewed as an arithmetic analogue of the FreeXOR technique due to Kolesnikov and Schneider [KS08] for Boolean circuits; multiplication, exponentiation by (public) constant, and high fan-in gates are also significantly improved beyond the naïve implementations. However, operations such as comparison of two numbers remain challenging, and prohibitively costly in the CRT representation. To overcome this issue, Ball et al. suggested a method to convert CRT numbers to a positional number system other than the binary system, namely the primorial mixed radix (PMR) system. Although highly improved over the straightforward (convert to binary) approach, the solution is still costly.

The work of Ben-Efraim [Ben18] is secure in the presence of a passive adversary and assumes an honest majority, and involves a circuit construction comprising a mixture of arithmetic and Boolean gates. Ben-Efraim's construction also allows linear operations to be performed for free, while for multiplication gates a "designated" solution is proposed, inspired by the half-gates approach of Zahur et al. [ZRE15], extended to the multiparty setting. This is because projection gates (that is, gates that convert values in one ring to the equivalent values in another ring) are difficult to achieve in the multiparty setting, unlike the two-party setting, where as shown by Ball et al. [BMR16], general projection gates are feasible.

Unfortunately, row-reduction techniques [NPS99, PSSW09] in the Boolean setting, and also applied in [BMR16], cannot be directly applied in the multiparty setting as protocols for more than two parties are (usually) symmetrical – that is, every party acts both as garbler and evaluator. However, by elegantly re-applying a variation of the half-gates approach [ZRE15], Ben-Efraim proposes a construction for a "designated" selector gate solution (i.e., a gate which selects one out of two arithmetic inputs u and v, based on a third, binary input b) that reduces computation cost. Specifically, after describing the construction of a straightforward selector (projecting the bit to characteristic p, and then performing a multiplication using the standard multiplexing equation $u + (v - u)b$), Ben-Efraim demonstrates the designated selector gate, which has the same communication cost as the straightforward one ($2p + 2$ ciphertexts), but it improves the computation cost by 33% (i.e., 2 decryptions for the designated construction, instead of 3).

Concurrently and independently of our work, Ball et al. [BCM+19] propose a series of optimisations over the previous state-of-the-art in the two-party setting [BMR16], which is tailored to the garbling of neural networks. One of their main technical contributions is the new mixed-modulus half-gate, which allows efficiently multiplying circuit wires from different domains. This can be thought of as a generalisation of the alternative selector gate that we present in this work, as we can only multiply bit wires by arithmetic wires, while their construction is not limited to bits. While our method can only treat mixed-modulus half-

gate multiplications if one of the two domains is the $\mathbb{F}_2$, the approach of Ball et al. [BCM+19] is generalisable to multiplication of wires from any (different) domain. This is achieved by exploiting the asymmetry between the parties in the two-party case, where we can choose certain labels to only be used in one of the parties' half-gates. This does not extend to the multiparty garbling setting, which is our focus, because all parties play the role of the garbler. Still, we maintain that garbled multiplication of an integer by a bit is indeed the most commonly occurring mixed-modulus multiplication (e.g., selector gates). Note that the communication cost of our approach is almost the same as the cost of the approach of Ball et al. [BCM+19] (in the case of multiplying by a bit). The second contribution of that work is an improved mixed-radix addition, which is important for increasing the efficiency of the non-linear parts of a garbled neural network. Mixed-radix operations (other than the ones where the one operand is base 2) do not appear to extend readily to the multiparty case.

1.2 Our Contribution

We continue the study of Ben-Efraim [Ben18] of multiparty garbling of circuits that contain both arithmetic and Boolean gates. Ben-Efraim [Ben18] showed how to construct a designated selector gate in this setting, based on an extension of the half-gate technique. The communication cost of Ben-Efraim's [Ben18] selector gate is the same as in the straightforward construction, while that work manages to reduce the computation cost by approximately 33% (i.e., 2 decryptions instead of 3 at evaluation time). We propose an *alternative designated selector gate*, which while it requires again 3 decryptions at evaluation time, it *reduces the communication cost to almost half* of that of Ben-Efraim's solution. We achieve this by making use of preprocessed data called daBits, proposed by Rotaru and Wood [RW19] and improved on in [AOR+19].

The other contribution of this work is to show how to perform multiparty garbling of both arithmetic circuits with *active security in the full-threshold multiparty setting*. We achieve this by using an authentication subprotocol akin to those in MASCOT [KOS16] and in SPD$\mathbb{Z}_{2^k}$ [CDE+18] to apply the Boolean circuit garbling approach by Hazay et al. [HSS17] to arithmetic garbling. One can view our contribution as extending the work of Hazay et al. [HSS17] to the arithmetic case and combining it with recent arithmetic garbling techniques.

2 Preliminaries

In this section we discuss the security model and present the main building blocks that we deploy or extend in our work. The goal is to evaluate an arithmetic circuit over a field of prime order p, denoted by $\mathbb{F}_p$.

2.1 Security Model

The protocols in this work are proved secure in the universal composability (UC) framework of Canetti [Can00], and we assume the reader's familiarity with it.

We consider an active, static adversary that can corrupt up to $n-1$ out of the n total parties. An active adversary may deviate arbitrarily from the protocol description, and a static adversary can choose which parties it will corrupt at the beginning of the protocol execution but not thereafter. Consequently, the functionalities are assumed to know at the beginning of their execution the set of corrupt parties: in the more general setting, the ideal-world adversary sends special "corruption" messages so that the functionality knows how to interact with different parties. Security is parameterised by the statistical security parameter, σ, and the computational security parameter, κ. We do not provide an implementation but typically one sets $\kappa \in \{64, 96, 128\}$ and $\sigma \in \{40, 80\}$ with $\sigma < \kappa$.

We will make use of the standard functionalities $\mathcal{F}_{\mathsf{Rand}}$ given in Figure 2 and $\mathcal{F}_{\mathsf{Commit}}$ given in Figure 3.

Functionality $\mathcal{F}_{\mathsf{Rand}}$

On input (Rand, X) from all parties, sample $x \xleftarrow{\$} X$ uniformly and send x to all honest parties and $\mathcal{S}$.

Fig. 2: Functionality $\mathcal{F}_{\mathsf{Rand}}$ for agreeing on random strings sampled uniformly from a specified domain.

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

The ideal-world adversary is denoted by $\mathcal{S}$ and the indexing set of corrupt parties by A.

Initialise On input $(\mathsf{Initialise}, \mathsf{sid})$ from all honest parties, initialise a database DB.

Commit On input $(\mathsf{Commit}, x, i, \mathsf{sid})$ from party P_i, sample $\mathsf{id}_{x,i}$, store $(\mathsf{id}_{x,i}, x)$ in DB, and send $\mathsf{id}_{x,i}$ to all honest parties and $\mathcal{S}$.

Open On input $(\mathsf{Open}, \mathsf{id}_{x,i}, \mathsf{sid})$ from all parties and $\mathcal{S}$, retrieve $(\mathsf{id}_{x,i}, x)$ in DB;

- If $i \in A$ then await a message OK or Abort from $\mathcal{S}$. If the message is OK then send x to all honest parties; otherwise, send the message Abort to all honest parties and halt.
- If $i \in [n] \setminus A$, then send x to all honest parties and $\mathcal{S}$ and continue.

Fig. 3: Standard commitment functionality.

2.2 Secret-Sharing

We use the notation $\langle x \rangle$ to denote that the secret x is additively shared amongst the n parties: that is, the dealer samples $\{x^i\}_{i=1}^{n-1}$ uniformly at random from $\mathbb{F}$, sets $x_n := x - \sum_{i=1}^{n-1} x^i$, and for each $i \in [n]$ sends x^i to party P_i.

We denote an *authenticated* shared value x by $[\![x]\!]$, which means that x is shared as above, and additionally there is some procedure for verifying that the sharing of x is not modified by the adversary. In the full-threshold setting, this is typically achieved by secret-sharing an information-theoretic Message Authentication Code (MAC) on every secret, as is done in BDOZ [BDOZ11], TinyOT [NNOB12] and SPDZ [DPSZ12]. The details of how secrets are authenticated in $\mathbb{F}_p$ and verified for correctness are not important for this work: it suffices to understand that if an error is introduced on any variable written as $[\![x]\!]$, this will be detected by the honest parties.

2.3 Garbling

We assume the reader is familiar with circuit garbling, but provide an overview here. A garbled circuit is a randomised version of a circuit that allows multiple parties to evaluate a function on the union of their private inputs without revealing anything more about their private inputs than what can be inferred from their own inputs and the output alone. In the two-party setting, this procedure is asymmetric; the high-level idea is as follows: one party, called the garbler, generates a "garbled" version of a circuit, hardwiring its own inputs in the circuit; then the other party, called the evaluator, evaluates the garbled circuit on its inputs (given some encoding information by the garbler that is provided in such a way that the garbler does not learn the evaluator's inputs) to obtain a "garbled" encoding of the output. At the end, the two parties communicate to reveal the final output to both.

Now we make things more concrete. Each fan-in-2 gate $g : \mathbb{F}^2 \to \mathbb{F}$ in the circuit with input wires u and v and output wire w is expressed as a table with one row for each $(\alpha, \beta) \in \mathbb{F}^2$ so that a row in the table has the form $(\alpha, \beta, g(\alpha, \beta))$. The garbler then samples a key for each possible value of α, β and $\gamma := g(\alpha, \beta)$. These keys typically live in some finite extension of the base field $\mathbb{F}^\ell$ where ℓ is $O(\kappa)$ so that the keys are $O(2^\kappa)$, but general garbling does not prescribe how these keys should look except that certain garbling optimisations constrain the encryption scheme to have certain properties. The values in the input/output table are replaced with their corresponding encryption keys. Finally, the keys corresponding to the output wire w of the table are encrypted first under the key corresponding to the input on wire u input, and then under the key corresponding to the input on wire v input. In practice, the encryption function is a pseudorandom one-time-pad using a pseudorandom function (PRF) taking two keys, and using the gate index as a nonce so that the entry for input (α, β) in the table representation of gate g is converted to a ciphertext:

$$\widetilde{g}_{\alpha,\beta} := F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \mathsf{k}_{w,g(\alpha,\beta)},$$

where g is a gate index and acts as a nonce for the encryption, and $\mathsf{k}_{w,g(\alpha,\beta)}$ is the key. All of these $|\mathbb{F}|^2$ ciphertexts (i.e., the final column of the table) are handed to the evaluator. To begin evaluating, the evaluator is handed keys corresponding to its inputs and decrypts gates by computing $\widetilde{g}_{\alpha,\beta} - F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g)$. This results in a key that can be used to decrypt the next gate in the circuit (after the evaluator has also obtained the output key of another gate from elsewhere in the circuit). The evaluation involves proceeding iteratively through the circuit in this way, decrypting using pairs of keys, until a final output key is obtained.

To hide the inputs of the evaluator from the garbler when obtaining the initial gate input keys, the keys are sent using oblivious transfer (OT). Oblivious transfer is a channel in which a sender sends many messages, and the receiver selects one, with the guarantees that the sender cannot know which option the receiver selected and the receiver learns nothing about the messages it did not pick. In circuit garbling, for each wire on which the evaluator has input, the garbler sends the $|\mathbb{F}|$ different possible keys and the evaluator chooses the one corresponding to its input.

The circuit has the values of the garbler hardwired in. This is achieved, for example, by only encrypting under the "v" keys if the garbler provides the input on wire u for a given gate. However, at the moment the *order* of the ciphertexts may reveal to the evaluator the input of the garbler. To hide the garbler's input from the evaluator, the ciphertexts are randomly permuted using so-called *permutation* or *masking* values chosen by the garbler. In the arithmetic case, this is essentially a rotation of the table rows. In order to evaluate the gates correctly, when evaluating a gate, in addition to learning the output key, the evaluator must learn a so-called *external* or *signal* value, which is the real value v masked with the masking value λ, that is, $e := v + \lambda$, so that it knows which ciphertexts to decrypt for each gate despite the rows being permuted. The ciphertexts are then

$$\widetilde{g}_{\alpha,\beta} := F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \left(\mathsf{k}_{w,g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w} \middle\| (g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w\right),$$

where $g(\alpha - \lambda_u, \beta - \lambda_v) + \lambda_w$ is the masked output wire (i.e., external) value. (The reader should think of the key as being in $\mathbb{F}^\ell$ for some ℓ which is $O(\kappa)$, and the external value as being in $\mathbb{F}$, and $F : \mathbb{F}^\ell \times \mathbb{F}^\ell \times \{0,1\}^{\log_2(|g|)} \to \mathbb{F}^{\ell+1}$.) The reader is referred to the original work of Beaver et al. [BMR90] for a complete discussion of the permutation method (known as *point-and-permute*).

A technique known as FreeXOR, generalised for arithmetic circuits by Ben-Efraim et al. [BLO16], can be employed to allow linear gates to be evaluated for free: the garbler chooses a global difference R and then for every non-linear gate, the wire key for the value 0 is a random element $\mathsf{k}_{w,0}$ of $\mathbb{F}$ and the wire key for each value $\gamma \in \mathbb{F}_p \setminus \{0\}$ is set to $\mathsf{k}_{w,\gamma} := \mathsf{k}_{w,0} + \gamma \cdot R$. Then for linear (i.e., addition) gates, the output 0 wire key is defined as $\mathsf{k}_{w,0} := \mathsf{k}_{u,0} + \mathsf{k}_{v,0}$ and the corresponding mask as $\lambda_w := \lambda_u + \lambda_v$. Other gates are computed as:

$$\begin{aligned}\widetilde{g}_{\alpha,\beta} :=& F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \\ &+ \left(\mathsf{k}_{w,0} + (g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w R) \middle\| (g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w)\right).\end{aligned}$$

Note that instead of encrypting a concatenation of the masking bit with the key, the garbler can use some form of authenticated encryption, and then the evaluator decrypts ciphertexts until it finds a valid decrypted message and considers this the output key. This technique will be used in the garbling described later.

Half Gates During the evaluation of the circuit, the signal values learnt by the evaluator "contain" the real values (in the sense that they are linearly dependent on them); likewise, the keys contain information regarding the real values. The idea behind half-gates is to exploit this information to reduce the amount of garbling required: during evaluation, the evaluator can compute the product of a signal value e_u with a key k_{v,e_v} to obtain "almost" a key for the product $v_u \cdot v_v$, and then can correct the errors that arise from the masking values using garbled gates (i.e., ciphertexts) in the more usual way[1]. In a sense, the difficult part of the multiplication gate, namely the cross-term $v_u \cdot v_v$ in the output key $\mathsf{k}_{w,e_w} = \mathsf{k}_{w,0} + (\lambda_w + v_u v_v)R$, is computed by computing $e_u \cdot \mathsf{k}_{v,e_v}$. The reason this is useful is that the errors that must be corrected in the product are each functions in the value of only *one* of the two real wire values v_u or v_v (and a combination of the (fixed) masking values). This means that the ciphertexts containing the corrections can be generated independently for each pair of inputs in $\mathbb{F}_p^2$ into the gate, which means only $p + p$ ciphertexts are needed, rather than $p \cdot p$ as required by garbling in the conventional manner.

To design a half gate, one observes what can be obtained from products of signal value with keys of input wires, namely from $e_u \cdot \mathsf{k}_{v,e_v}$, or from $e_v \cdot \mathsf{k}_{u,e_u}$. For example,

$$\begin{aligned} e_u \mathsf{k}_{v,e_v} &= (v_u + \lambda_u)(\mathsf{k}_{v,0} + (v_v + \lambda_v)R) \\ &= v_u \mathsf{k}_{v,0} + \lambda_u \mathsf{k}_{v,0} + v_u v_v R + \lambda_u v_v R + v_u \lambda_v R + \lambda_u \lambda_v R \\ &= v_u v_v R + \underbrace{v_u \mathsf{k}_{v,0} + v_u \lambda_v R}_{\text{Dependent on } v_u} + \underbrace{\lambda_u v_v R}_{\text{Dependent on } v_v} + \underbrace{\lambda_u \mathsf{k}_{v,0} + \lambda_u \lambda_v R}_{\text{Dependent on neither}} \end{aligned}$$

Now since the goal is to obtain $\mathsf{k}_{w,e_w} = \mathsf{k}_{w,0} + (\lambda_w + v_u v_v)R$, for every $\gamma \in \mathbb{F}_p$ the garbler generates two ciphertexts: one encrypting

$$\mathsf{k}_{w,\mathsf{g},0} + \lambda_w R - ((\gamma - \lambda_u)(\mathsf{k}_{v,0} + \lambda_v R) + (\lambda_u \mathsf{k}_{v,0} + \lambda_u \lambda_v R))\,,$$

and the other encrypting

$$\mathsf{k}_{w,\mathsf{e},0} - (\gamma - \lambda_v)(\lambda_u R)\,.$$

The output wire key is set to $\mathsf{k}_{w,0} := \mathsf{k}_{w,\mathsf{g},0} + \mathsf{k}_{w,\mathsf{e},0}$. The evaluator will decrypt the ciphertexts corresponding to $\gamma = e_u$ for the first half gate and $\gamma = e_v$ for

[1] This is analogous to the key-switching operation required for relinearisation of ciphertexts in somewhat-homomorphic encryption (SHE) schemes, where one first does a "naïve" multiplication, and then corrects the errors.

the second; since $e_u - \lambda_u = v_u$ and $e_v - \lambda_v = v_v$, they will obtain the correct key by summing the two resulting plaintexts and the value $e_u \mathsf{k}_{v,e_v}$. Note that in the original two-party protocols, one gate input was assumed to come from the garbler and the other from the evaluator, so the evaluator would also be involved in the garbling of the half gates. This results in reduced communication since each party knows one of the wire masks, which makes the MPC computations more straightforward. In the multiparty setting described later, no party knows the wire masks, so the main saving comes from reducing the quadratic cost p^2 to the linear cost $2 \cdot p$.

Some recent papers evaluate over a ring of primorial modulus rather than over a prime field in order to reduce the size of multiplication gates from $(\sum_{i=1}^{t} p_i)^2$ to $\sum_{i=1}^{t} p_i^2$ total ciphertexts. However, using the half-gate technique, the cost is the same regardless of the modulus, at $2 \cdot \sum_{i=1}^{t} p_i$ ciphertexts. Another place where the CRT approach is useful is for performing non-linear operations such as computing powers. These operations are quite expensive even in the passive security setting. While it may be useful to have an actively-secure protocol for arithmetic circuits over a composite modulus ring, there are difficult challenges to overcome arising from the presence of zero divisors; thus we leave this to future work.

We evaluate the garbled circuits in $\mathbb{F}_p$, for which the straightforward garbling approach requires that p be small enough to allow parties to send $O(p)$ ciphertexts per multiplication gate, but large enough so that the PRF keys used for encryption are secure against a computationally-bounded adversary. To do this, we evaluate circuits in $\mathbb{F}_p$, but take keys in an extension field, specifically $\mathbb{F}_{p^{\ell_\kappa}}$, where $\ell_\kappa := 1 + \lceil \kappa / \log p \rceil$.

Multiparty Garbling In multiparty garbling, originally developed by Beaver et al. [BMR90], all parties act as garbler and evaluator. Lindell et al. [LPSY15] showed how to use actively-secure secret-sharing-based MPC to compute a multiparty garbled circuit with active security. Using MPC, each party generates keys for one circuit, and the masking values are chosen at random and are unknown to the parties. The result is that for each gate, each party holds the following n ciphertexts, indexed by j:

$$\widetilde{g}^j_{\alpha,\beta} := \sum_{i=1}^{n} F_{\mathsf{k}^i_{u,\alpha}, \mathsf{k}^i_{v,\beta}}(g, j) + \left(\mathsf{k}^j_{w,0} + (g(\alpha - \lambda_u, \beta - \lambda_v) + \lambda_w) R^j \right).$$

Since each party P_i generates one set of keys (those indexed by i), the external values on the wires can be learnt by each party examining the output plaintext m^i from its own circuit and setting $e_w := (m^i - \mathsf{k}^i_{w,0}) \cdot R^{i^{-1}}$ and $\mathsf{k}^i_{w,e_w} := m^i$.

In many ways, the protocol we present in this work is a straightforward generalisation of garbling protocols over $\mathbb{F}_2$. Notice that for a Boolean circuit, the half-gate approach is no more efficient than the naïve approach, unless we are in the two-party setting in which one party is the garbler and one the evaluator, rather than all being both as in the multiparty setting.

PRF Assumption To encrypt a gate, a single-keyed PRF is evaluated on a nonce and used to one-time-pad encrypt a key. To make use of the (generalised) FreeXOR technique, the following assumption is required.

Let $F : \mathbb{F}_{p^{\ell_\kappa}} \times \mathbb{N} \to \mathbb{F}_{p^{\ell_\kappa}}$ be a keyed pseudorandom function (PRF). Define the oracle $\mathcal{O}_{F,R}$ in the following way:

$$\begin{aligned}\mathcal{O}_{F,R} : \mathbb{F}_{p^{\ell_\kappa}} \times \mathbb{F}_p \times \mathbb{N} \times \mathbb{F}_p &\to \mathbb{F}_{p^{\ell_\kappa}} \\ \mathcal{O}_{F,R}(\mathsf{k}, \gamma, x, \delta) &\mapsto F_{\mathsf{k}+\gamma \cdot R}(x) + \delta \cdot R\end{aligned}$$

Now define $\mathcal{F}_{\mathsf{RO}}$ to be an oracle that, on input a query $m = (\mathsf{k}, \gamma, x, \delta) \in \mathbb{F}_{p^{\ell_\kappa}} \times \mathbb{F}_p \times \mathbb{N} \times \mathbb{F}_p$, if m has not been queried before, samples $r \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and outputs r, and otherwise outputs whatever was sampled previously.

The following definition was given by Hazay et al. [HSS17] for Boolean functions, and a similar definition for arithmetic circuits was given by Ball et al. [BMR16].

Definition 1 (Circular Correlation Robustness). *For the oracles above, define* legal queries *as those with inputs in the correct domain, and additionally:*

1. *The oracle may not be queried when $\gamma = 0$.*
2. *The oracle may not be queried twice for the same δ unless at least one other variable changes.*

Then we say that F is circular correlation robust *if for all probabilistic polynomial-time distinguishers $\mathcal{D}$, it holds that*

$$\left| \Pr_{R \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}} [\mathcal{D}^{\mathcal{O}_{F,R}}(1^\kappa)] - \Pr[\mathcal{D}^{\mathcal{F}_{RO}}(1^\kappa)] \right| = O(2^{-\kappa})$$

In the garbling protocols, the PRF is queried on values (g, j), where $g \in \mathbb{N}$ is the gate index and $j \in [n]$ is the party index, parsed as a natural number $\lceil \log n / \log 10 \rceil \cdot g + j$.

The choice for this definition comes from the fact that parties should not be able to distinguish between keys generated using global differences and uniform keys in the field. Note that while the keys generated for each wire are only in some coset $\mathsf{k}_{w,0} + \{\gamma R : \gamma \in \mathbb{F}_p\}$ of $\mathbb{F}_{p^{\ell_\kappa}}$, the distinguisher is only allowed to query once per key per nonce for a fixed δ. This corresponds to the fact that in the garbling, the evaluator(s) can only decrypt a single ciphertext.

3 Full-Threshold Active Security

We can define an n-party arithmetic garbling protocol by extending the state-of-the-art techniques used by Hazay et al. [HSS17] for Boolean garbling to arithmetic garbling, using actively-secure MPC over $\mathbb{F}_p$ as a black box, and using the half-gate techniques described for arithmetic circuits by Ben-Efraim [Ben18]. In this section we will describe the actively-secure garbling of the "standard" multiplication gate, since using the classical garbling techniques one can replace

the multiplication function with any gate $g : \mathbb{F}_p^2 \to \mathbb{F}_p$; our techniques for active security also apply to other gates, and indeed in the protocol later we garble multiplication half gates. Many of the techniques due to Hazay et al. [HSS17] apply almost immediately to the arithmetic case and so the exposition here closely follows theirs. We will first explain the components of the garbling protocol at a high level, then discuss how to realise these different parts, and finally we will give the complete protocol.

3.1 Overview

In the arithmetic analogue of the multiparty garbling protocol of Beaver et al. [BMR90], with the optimisations described in Section 2, the goal is to produce a set of $p^2 \cdot n$ ciphertexts, indexed by $j \in [n]$ and $(\alpha, \beta) \in \mathbb{F}_p^2$, for each multiplication gate, of the form

$$\widetilde{g}^j_{\alpha,\beta} := \left(\sum_{i=1}^{n} F_{\mathsf{k}^i_{u,\alpha}, \mathsf{k}^i_{v,\beta}}(g, j) \right) + \mathsf{k}^j_{w,0} + R^j \cdot ((\alpha - \lambda_u) \cdot (\beta - \lambda_v) + \lambda_w) \,,$$

where the wire masks λ_u, λ_v and λ_w are not known to any party and the keys indexed by i are generated by P_i. For now, the reader can think of $\mathsf{k}_{u,\alpha}$, $\mathsf{k}_{v,\beta}$, $\mathsf{k}_{w,0}$ and R^j as lying in a finite extension of $\mathbb{F}_p$ – the same space as the codomain of the PRF. The approach of Hazay et al. for Boolean circuits to produce these ciphertexts with active security is to generate a secret-shared version of $\widetilde{g}^j_{\alpha,\beta}$ for every $j \in [n]$ and open them, in the following way:

1. Use a generic "Bit-MPC" functionality, $\mathcal{F}_{\mathsf{BitMPC}}$, for parties to obtain authenticated secret-shared random bits $[\![\lambda_u]\!]$, $[\![\lambda_v]\!]$ and $[\![\lambda_w]\!]$ and to compute $[\![\lambda_u \cdot \lambda_v]\!]$.
2. Use correlated oblivious transfer (COT) to compute the products by the global differences, that is, for each $j \in [n]$ to compute secret-shared versions of
$$R^j \cdot \lambda_u, \quad R^j \cdot \lambda_v, \quad R^j \cdot (\lambda_w + \lambda_u \cdot \lambda_v).$$
3. *Locally* combine the secret-shared values with local PRF evaluations to obtain a sharing of each gate $\widetilde{g}^j_{\alpha,\beta}$.
4. Open all the sharings.

A key observation, first made by Lindell et al. [LPSY15], is that the sharings need not be authenticated, as the parties will abort during circuit evaluation with overwhelming probability if the adversary introduces errors. Crucially, this means that the PRF evaluations need neither be authenticated, nor proved correct using a zero-knowledge proof. Authentication is required on the wire masks only to ensure the multiplication is performed correctly. Thus, only *one* secure Bit-MPC multiplication is required per AND gate, along with an amortised COT operation.

Our approach here is to give the simple generalisation for the field $\mathbb{F}_p$, noting that the keys must live in the space $\mathbb{F}_{p^{\ell_\kappa}}$, where $\ell_\kappa := 1 + \lceil \kappa / \log p \rceil$. We first describe the replacement of $\mathcal{F}_{\mathsf{BitMPC}}$ with MPC over a field, denoted by $\mathcal{F}_{\mathsf{MPC}}$, and

second show how to replace COT with correlated oblivious product evaluation (COPE) (also known as vector oblivious linear function evaluation (vOLE)).

3.2 Wire Mask Arithmetic

For arithmetic circuits, the bit masks are replaced with masks in $\mathbb{F}_p$ and hence $\mathcal{F}_{\mathsf{BitMPC}}$ is replaced with a generic $\mathcal{F}_{\mathsf{MPC}}$ protocol, which we give in Figure 4. In the garbling protocol, just as in the work of Hazay et al. [HSS17], to obtain a wire mask λ_u, each party samples $\lambda_u^i \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ to create an authenticated sharing of this value; then they call the **Add** procedure to obtain $[\![\lambda_u]\!] = \sum_{i=1}^n [\![\lambda_u^i]\!]$. They do similarly for λ_v and λ_w so that the parties obtain $[\![\lambda_u]\!]$, $[\![\lambda_v]\!]$ and $[\![\lambda_w]\!]$, and then call the **Multiply** procedure to multiply $[\![\lambda_u]\!]$ and $[\![\lambda_v]\!]$, and obtain $[\![\lambda_{uv}]\!] = [\![\lambda_u \cdot \lambda_v]\!]$.

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

If a command is received where sid differs from what was sent during **Initialise**, ignore the command and await the next. At any point, the ideal-world adversary $\mathcal{S}$ can send the message Abort and the functionality sends the message Abort to all honest parties and halts. To save on notation, vectors of secrets may also be stored and computed on (where multiplication is componentwise).

Initialise On input $(\mathsf{Initialise}, \mathbb{F}, \mathsf{sid})$, store the field $\mathbb{F}$ and initialise a new database, $\mathsf{DB} := \varnothing$.

Input On input $(\mathsf{Input}, i, \mathsf{id}_x, x, \mathsf{sid})$ from P_i and $(\mathsf{Input}, i, \mathsf{id}_x, \bot, \mathsf{sid})$ from all other parties, where $x \in \mathbb{F}$ and id_x is a new identifier, append the entry (id_x, x) to the database DB.

Add On input $(\mathsf{Add}, \mathsf{id}_x, \mathsf{id}_y, \mathsf{id}_z, \mathsf{sid})$ where id_x and id_y are identifiers in the database and id_z is a new identifier, retrieve (id_x, x) and (id_y, y) from memory and append the entry $(\mathsf{id}_z, x + y)$ to the database DB.

Multiply On input $(\mathsf{Multiply}, \mathsf{id}_x, \mathsf{id}_y, \mathsf{id}_z, \mathsf{sid})$ where id_x and id_y are identifiers in the database and id_z is a new identifier, retrieve (id_x, x) and (id_y, y) from memory and append the entry $(\mathsf{id}_z, x \cdot y)$ to the database DB.

Output On input $(\mathsf{Output}, \mathsf{id}_x, \mathsf{sid})$ from all parties, retrieve the entry (id_x, x) from DB, send x to $\mathcal{S}$, and await a message OK or Abort; if the message is OK then send x to all honest parties, and otherwise send the message Abort to all honest parties and halt.

Fig. 4: Functionality $\mathcal{F}_{\mathsf{MPC}}$ for performing general MPC, secure *with abort*.

3.3 Wire Mask/Global Difference Products

In the garbling protocol, for every wire w the parties require (unauthenticated) sharings of $R^j \cdot \lambda_w$ for every $j \in [n]$. Since λ_w is additively shared amongst the parties, they actually compute sharings of $R^j \cdot \lambda_w^i$ for every $j \in [n]$ and

$i \neq j$. Since the global difference is fixed for all gates in the circuit, in the Boolean case such sharings can be generated using COT, in which a sender chooses a fixed correlation, namely R^j, and the receiver inputs their sharing of the mask λ_w^i; then the sender obtains some $q^{j,i}$ and the receiver some $t^{i,j}$ such that $q^{j,i} + t^{i,j} = \lambda_w^i \cdot R^j$. The protocol for computing the wire mask/global difference products is called $\Pi_{\mathsf{Bit}\times\mathsf{String}}$ in the work of Hazay et al. [HSS17], since $R^j \in \mathbb{F}_{2^k}$ and the masks are bits.

We can apply essentially the same techniques here, and correctness of the protocol follows in exactly the same way. The difference is that we are now interested in masks in $\mathbb{F}_p$ and global differences in $\mathbb{F}_{p^{\ell_\kappa}}$. Thus, we must use the correlated oblivious product evaluation (COPE) presented in Figure 5, which is notationally identical with the protocol $\Pi_{\mathsf{Bit}\times\mathsf{String}}$, but works in any finite field. Note that $\mathcal{F}_{\mathsf{COPE}}$ accepts inputs from the sender in $\mathbb{F}_{p^{\ell_\kappa}}$, but in our protocol the inputs are assumed to be in $\mathbb{F}_p$, as they are circuit wire masks. Thus a corrupt sender could send an element of $\mathbb{F}_{p^{\ell_\kappa}} \setminus \mathbb{F}_p$ in the instance of $\mathcal{F}_{\mathsf{COPE}}$. However, the proof of Lemma 1, which can be found in Appendix A, shows that the checks ensure secrets lie in $\mathbb{F}_p$. It is possible to use a functionality such as $\mathcal{F}_{\mathsf{OLE}}^{t,1}$ by Ghosh et al. [GNN17] that accepts input from the sender in a small field and from the receiver in an extension field and outputs a sharing in the larger field, but for a technical reason these are not amenable to OT extension [IKNP03] as is $\mathcal{F}_{\mathsf{COPE}}$ and are therefore less efficient when performing a large number of multiplications. Realising a product functionality more efficiently would improve the overall efficiency of the garbling protocol and we leave this for future work. The subprotocol for mutliplying global differences with wire masks is given in Figure 6.

For active security, it is necessary to check that each P_j provides the same global difference R^j with every other P_i, and that every P_i provides the same sharing λ_w^i with every other P_j. Observe that

$$\left(x^j \cdot R^j + \sum_{i\neq j} q^{j,i}\right) + \left(\sum_{i\neq j} t^{i,j}\right) = R^j \cdot \left(x^j + \sum_{j\neq i}(q^{j,i} + t^{i,j})\right) = R^j \cdot \left(\sum_{i=1}^{n} x^i\right)$$

where the first summand is computed by party P_j and for each $i \neq j$, $t^{i,j}$ is held by P_i. The fact that this relationship must hold (indeed, it holds by design) can be used to check correctness of a *batch* of secrets $\{[\![x_k]\!]\}_{k=1}^m$ as follows: parties can take an additional mask $[\![x_{m+1}]\!]$, reveal a random linear combination $c := x_{m+1} + \sum_{k=1}^m \chi_k x_k$ (where $\chi_k \in \mathbb{F}_p$ for all k), and check for all $j \in [n]$ that $\langle z^j \rangle$ defined by

$$z^{i,j} := t_{m+1}^{i,j} + \sum_{k=1}^{m} \chi_k \cdot t_k^{i,j} \qquad (i \neq j)$$

and

$$z^{j,j} := -c \cdot R^j + \left(x_{m+1}^j \cdot R^j + \sum_{i\neq j} q_{m+1}^{j,i}\right) + \sum_{k=1}^{m} \chi_k \cdot \left(x_k^j \cdot R^j + \sum_{i\neq j} q_k^{j,i}\right)$$

is an additive sharing of 0. It will be shown in the proof of Lemma 1 that the probability that parties are inconsistent but all of the n sharings $\{\langle z^i \rangle\}_{i=1}^n$ are zero is bounded above by p^{-1}; thus the check is performed independently $\ell_\sigma := \lceil \sigma / \log p \rceil$ times in parallel to ensure at least σ bits of statistical security.

Concrete instantiation One of the reasons that the protocol of Hazay et al. [HSS17] is so efficient is that the functionality $\mathcal{F}_{\mathsf{BitMPC}}$ can be realised using the n-party variant [BLN+15] of the TinyOT [NNOB12] protocol, in which bits are authenticated exactly via sharings of $b^i \cdot R^j$, where R^j is taken to be the secret key of P_j. Thus sharings of the wire mask/global difference products are immediately available to the parties by the correctness of the $\mathcal{F}_{\mathsf{BitMPC}}$ functionality, without the need for a separate $\Pi_{\mathsf{Bit}\times\mathsf{String}}$ protocol. However, currently the most efficient protocols in the setting of a large prime field use a different form of authentication and so this optimisation cannot be directly applied here. Instead, we can use, for example, the most recent version of the SPDZ protocol [DPSZ12] known as Overdrive [KPR18]. Note that in MASCOT [KOS16], pairwise MACs are generated and then combined to create global MACs, so it may be that this approach, which then obviates the need to perform the protocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ separately, is better in practice.

Lemma 1, states that an adversary succeeds in cheating without detection in $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ with only negligible probability in the statistical security parameter, σ.

Functionality $\mathcal{F}_{\mathsf{COPE}}$ (from [KOS16])

Let $\mathbf{g} : \mathbb{F}^{\lceil \log |\mathbb{F}| \rceil} \to \mathbb{F}$ be any map such that for every $x \in \mathbb{F}$, if $\mathbf{x} \in \{0,1\}^{\lceil \log |\mathbb{F}| \rceil}$ represents its bit-decomposition, then $\mathbf{g}(\mathbf{x}) = x$. Let $\mathbf{g}^{-1}(x)$ denote the bit-decomposition of x, which is well-defined by uniqueness of decomposition.

Initialise On receiving the message $(\mathsf{Initialise}, \mathbb{F}, P_j, P_i, \mathsf{sid}_{j,i})$ from parties P_i and P_j, await $\Delta \in \mathbb{F}$ from P_j, store Δ, and set $\boldsymbol{\Delta} := \mathbf{g}^{-1}(\Delta)$.

Extend On receiving the message $(\mathsf{Extend}, \mathsf{sid}_{j,i})$ from both parties,

1. – If P_i and P_j are honest then await $x \in \mathbb{F}$ from P_i, sample $q \xleftarrow{\$} \mathbb{F}$ and set
$$t = x \cdot \Delta - q$$
– If P_i is corrupt and P_j is honest then await $t \in \mathbb{F}$ and $\mathbf{x} \in \mathbb{F}^{\lceil \log |\mathbb{F}| \rceil}$ from $\mathcal{S}$ and set
$$q = \mathbf{g}(\mathbf{x} * \boldsymbol{\Delta}) - t$$
where $*$ denotes the coordinatewise product.
– If P_i is honest and P_j is corrupt then await $x \in \mathbb{F}$ from P_i and $q \in \mathbb{F}$ from $\mathcal{S}$ and compute
$$t := x \cdot \Delta - q.$$
2. Send t to P_i and q to P_j.

Fig. 5: Functionality for Correlated Oblivious Product Evaluation.

Subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$

Initialise For every ordered pair of parties (P_j, P_i), call an instance of $\mathcal{F}_{\mathsf{COPE}}$, denoted by $\mathcal{F}_{\mathsf{COPE}}^{(j,i)}$ with P_i as the sender and P_j as the receiver, with input $(\mathsf{Initialise}, \mathbb{F}_{p^{\ell_\kappa}}, P_j, P_i, \mathsf{sid}_{j,i})$, and input R^j from P_j.

Multiply To compute unauthenticated sharings $(\langle x_k \cdot R^i \rangle)_{k=1}^m$ from authenticated sharings $([\![x_k]\!])_{k=1}^m$ for which the parties additionally hold $(\langle x_k \rangle)_{k=1}^m$, the parties do the following:

1. **Mask** The parties generate $\ell_\sigma := \lceil \sigma / \log p \rceil$ masks: for each $l \in [\ell_\sigma]$,
   (a) For each $i \in [n]$, party P_i samples $x_{m+l}^i \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with input $(\mathsf{Input}, i, x_{m+l}^i, \mathsf{id}_{x_{m+l}^i})$ while each party P_j, $j \neq i$, provides corresponding input $(\mathsf{Input}, i, \bot, \mathsf{id}_{x_{m+l}^i})$.
   (b) The parties obtain $[\![x_{m+l}]\!] = \sum_{i=1}^n [\![x_{m+l}^i]\!]$ by creating a new identifier $\mathsf{id}_{x_{m+l}}$ and calling the **Add** procedure of $\mathcal{F}_{\mathsf{MPC}}$ multiple times.
2. **Generate** For each $j \in [n]$,
   (a) For every $i \neq j$,
      i. P_i and P_j call $\mathcal{F}_{\mathsf{COPE}}^{(i,j)}$ with input $(\mathsf{Extend}, \mathsf{sid}_{i,j})$:
         A. P_i provides $x_1^i, \ldots, x_{m+l}^i$ as input.
         B. P_i receives $(t_k^{i,j})_{k=1}^{m+l}$ and P_j receives $(q_k^{j,i})_{k=1}^{m+l}$.
      ii. It holds that $q_k^{j,i} + t_k^{i,j} = x_k^i R^j$. Party P_i sets $z_k^{i,j} := t_k^{i,j}$.
   (b) Party P_j sets $z_k^{j,j} := x_k^j R^j + \sum_{i \neq j} q_k^{j,i}$.
3. **Check**
   (a) Call $\mathcal{F}_{\mathsf{Rand}}$ with input $(\mathsf{Rand}, \mathbb{F}_p^{\ell_\sigma \times m})$ to obtain a matrix $H = (\chi_{l,k})_{l \in [\ell_\sigma], k \in [m]}$.
   (b) Let $\mathbf{x} := (x_k)_{k=1}^m$ and $\hat{\mathbf{x}} := (x_{m+l})_{l=1}^{\ell_\sigma}$. The parties compute $[\![\mathbf{c}]\!] := H \cdot [\![\mathbf{x}]\!] + [\![\hat{\mathbf{x}}]\!]$ and call $\mathcal{F}_{\mathsf{MPC}}$ with input $(\mathsf{Output}, \mathsf{id}_{\mathbf{c}}, \mathsf{sid})$ to obtain $\mathbf{c}$. If it aborts, then the parties abort.
   (c) Each party P_i computes $\mathbf{c}^{i,j} := H \cdot (z_k^{i,j})_{k=1}^m + (z_{m+l}^{i,j})_{l=1}^{\ell_\sigma}$ and $\mathbf{c}^{i,i} := -\mathbf{c} \cdot R^i + H \cdot (z_k^{i,i})_{k=1}^m + (z_{m+l}^{i,i})_{l=1}^{\ell_\sigma}$.
   (d) Each party P_i calls $\mathcal{F}_{\mathsf{Commit}}$ with input $(\mathsf{Commit}, \mathbf{c}^{i,j}, i, \mathsf{sid})$ for all $j \in [n]$.
   (e) When $\mathsf{id}_{\mathbf{c}^{i,j}}$ has been received from $\mathcal{F}_{\mathsf{Commit}}$ for all $i, j \in [n]^2$, call $\mathcal{F}_{\mathsf{Commit}}$ with input $(\mathsf{Open}, \mathsf{id}_{\mathbf{c}^{i,j}}, \mathsf{sid})$.
   (f) Check that $\sum_{i=1}^n \mathbf{c}^{i,j} = \mathbf{0}$ for all $j \in [n]$. If so, then each party P_i (locally) outputs $(z_k^{i,j})_{k \in [m], j \in [n]}$; otherwise, they abort.

Fig. 6: Subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ for multiplying global differences with wire masks.

The subprotocol for garbling is given in Figure 7, and the one for evaluation in Figure 8.

Lemma 1. *For the outputs $(z_k^{i,j})_{i,j \in [n]}$ of the subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ it holds that $\sum_{i=1}^n z_k^{i,j} = x_k^i \cdot R^j$ for all j except with probability at most $2^{-\sigma}$.*

Notice that in order to establish a unique signal value after decrypting ciphertexts, it is necessary to multiply by R^{-1}, which means that R must be invertible.

However, since R is sampled from a field, the random choice is invertible except if it is 0, which happens with probability $p^{-\ell_\kappa} < 2^{-\kappa} < 2^{-\sigma}$.

Theorem 1. *The execution of the subprotocol Π_{Garble} followed by the execution of the subprotocol Π_{Eval}, making use of the subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$, UC-securely realises the functionality $\mathcal{F}_{\mathsf{AC}}$ in the presence of a static, active adversary that corrupts up to $n-1$ parties, in the $\mathcal{F}_{\mathsf{Commit}}$,$\mathcal{F}_{\mathsf{COPE}}$,$\mathcal{F}_{\mathsf{MPC}}$,$\mathcal{F}_{\mathsf{Rand}}$-hybrid model, assuming the PRF F satisfies correlation-robustness.*

We define a security game in which a successful adversary breaks the circular correlation robustness assumption of a PRF F in the following way:

Game

1. The challenger $\mathcal{C}$ samples a bit $b \xleftarrow{\$} \{0,1\}$; if $b = 0$ then it initialises the oracle $\mathcal{O} := \mathcal{F}_{\mathsf{RO}}$; if $b = 1$ then it samples $R \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and initialises the oracle $\mathcal{O} := \mathcal{O}_{F,R}$.
2. The adversary $\mathcal{D}$ is provided with black-box access to $\mathcal{O}$.
3. After polynomially-many queries to $\mathcal{O}$, $\mathcal{D}$ outputs a bit b' to $\mathcal{C}$.
4. The adversary $\mathcal{D}$ wins the game if $b' = b$.

We are now ready to prove the theorem.

Proof. Note that, in contrast to the work of Hazay et al. [HSS17], the output masks in our work are *not* revealed after garbling; instead, the parties remove the masks *after* evaluating the circuit. Opening the wire masks after the evaluation is a common approach, taken for example by Wang et al. in their recent multiparty Boolean garbling protocol [WRK17] and leads to a more straightforward proof.

To prove that the protocol UC-securely realises the functionality $\mathcal{F}_{\mathsf{AC}}$ under the assumption that the PRF is correlation-robust, we will construct a simulator interacting with the real-world adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\mathsf{AC}}$ such that if an environment can determine that $\mathcal{A}$ is interacting with $\mathcal{S}$ instead of real honest parties then it must have been able to break the assumption on the PRF. To do this, we will construct a situation in which distinguishing between worlds immediately leads to a way to break the PRF assumption. Consider the simulator $\mathcal{S}$ defined as follows:

1. Execute Π_{Garble} honestly, sampling keys, masks and global differences as honest parties would in the execution of $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ and honestly executing internal copies of the oracles $\mathcal{F}_{\mathsf{Commit}}$, $\mathcal{F}_{\mathsf{COPE}}$, $\mathcal{F}_{\mathsf{MPC}}$ and $\mathcal{F}_{\mathsf{Rand}}$.
2. Sample inputs on behalf of emulated honest parties, compute signal values honestly, and send these signal values to $\mathcal{A}$; await the broadcasts from corrupt parties and extract the inputs using knowledge of the masks from the calls to $\mathcal{F}_{\mathsf{MPC}}$ in Step 1, and send the inputs to $\mathcal{F}_{\mathsf{AC}}$. Store the signal values for all of these input wires.
3. Await the final circuit output from $\mathcal{F}_{\mathsf{AC}}$.
4. Determine the "evaluation path" through the circuit based on all the broadcasted input signal values and the wire masks determined in the execution of Π_{Garble}, and then fix the wire mask for the final circuit output wire w to

be $e_w - v_w$, where v_w is the output from $\mathcal{F}_{\mathsf{AC}}$. (If there are multiple output wires then do similarly for each.)

5. For all honest parties, for all gates and wires, sample new wire keys uniformly at random.
6. Fix the shares of honest parties so that the gates are consistent with keys and ciphertexts from Step 5 instead of the ones generated in the honest execution of Π_{Garble} in Step 1. Specifically, for each honest party P_i, for every $j \in [n] \setminus \{i\}$, for every $\gamma \in \mathbb{F}_p$, set

$$\widetilde{g}^{j,i}_{\mathsf{g},\gamma} = F_{\mathsf{k}^i_{u,\gamma}}(g,j) + \rho^i_{j,g,\mathsf{g},\gamma}$$
$$\widetilde{g}^{j,i}_{\mathsf{e},\gamma} = F_{\mathsf{k}^i_{v,\gamma}}(g,j) + \rho^i_{j,g,\mathsf{e},\gamma}$$

   and then for every $\gamma \in \mathbb{F}_p$, set

$$\widetilde{g}^{i,i}_{\mathsf{g},\gamma} = F_{\mathsf{k}^i_{u,\gamma}}(g,i) + \mathsf{k}^i_{w,e_w} - \sum_{j \neq i} \rho^j_{i,g,\mathsf{g},\gamma}$$
$$\widetilde{g}^{i,i}_{\mathsf{e},\gamma} = F_{\mathsf{k}^i_{v,\gamma}}(g,i) - e_u \mathsf{k}^i_{v,e_v} - \sum_{j \neq i} \rho^j_{i,g,\mathsf{e},\gamma}$$

   (Note that $\mathcal{S}$ knows the values of $\rho^j_{i,g,\mathsf{e},e_v}$ for all $(i,j) \in [n]^2$ because it emulates the copies of $\mathcal{F}_{\mathsf{COPE}}$ locally, from which these values are computed.)
7. Open the gates honestly by sending the shares of (emulated) honest parties to corrupt parties.
8. Await the call to $\mathcal{F}_{\mathsf{MPC}}$ to open the final output mask(s) and execute this honestly. If an emulated honest party would abort, send Abort to $\mathcal{F}_{\mathsf{AC}}$, and otherwise send OK.

The execution of Π_{Garble} is simulated perfectly by the simulator, which samples contributions to masks and keys for emulated honest parties. The fact that the simulator samples inputs on behalf of emulated honest parties in Step 2 does not affect the correctness of simulation or change the distribution as viewed by the environment in any way, because each broadcasted external wire value is a real value masked by a uniform wire mask not revealed to the environment, and the final output is fixed by the simulator to the correct value regardless of these sampled values and the path traversed through the garbled circuit. In Step 6, the simulator fixes shares as honest parties would for the contributions to ciphertexts encrypting keys generated by corrupt parties, but for honest parties it fixes the shares so that the parties will compute the uniformly-sampled keys from Step 5 instead of keys according to a global difference that was chosen in Step 1. The reason for fixing keys and ciphertexts for honest parties in this way is that the honest party's actual global difference will be different from the simulator's sampled random global difference (used in the simulation of $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$) with high probability, so instead of arguing that a set of ciphertexts generated using keys with a fixed global difference is indistinguishable from another set of ciphertexts generated using keys with a *different* global difference (which would be an alternative way to define the simulator), we will show that such a set of ciphertexts is instead indistinguishable from a set of ciphertexts generated using

uniformly-randomly sampled keys. This latter assumption is essentially exactly the correlation-robustness assumption. Furthermore, since this is the *only* point in the distribution as viewed by the environment that is potentially different from a real execution, this is the *only* way to distinguish. Thus any environment that can distinguish between the hybrid and ideal worlds must do so by observing a difference in the distributions of ciphertexts generated, which breaks the PRF assumption.

It remains to show how to use a distinguishing environment $\mathcal{Z}$ to construct an adversary $\mathcal{D}$ (for the game outlined above). To this end, we first define a modified simulator $\mathcal{S}'$ that uses the oracle $\mathcal{O}$ provided to $\mathcal{D}$. The simulator $\mathcal{S}'$ executes exactly as $\mathcal{S}$, but it fixes an arbitrary choice of honest party P_{i^*}, and when it fixes the keys and ciphertexts in Step 6, it alters the shares of P_{i^*} for each ciphertext as follows:

For every $j \in [n] \setminus \{i^*\}$, for every $\gamma \in \mathbb{F}_p \setminus \{0\}$, set

$$\widetilde{g}^{j,i^*}_{\mathsf{g},e_u+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{u,e_u}, \gamma, (g,j), 0\right) + \rho^{i^*}_{j,g,\mathsf{g},e_u+\gamma}$$
$$\widetilde{g}^{j,i^*}_{\mathsf{e},e_v+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{v,e_v}, \gamma, (g,j), 0\right) + \rho^{i^*}_{j,g,\mathsf{e},e_v+\gamma}$$

and set

$$\widetilde{g}^{j,i^*}_{\mathsf{g},e_u} = F_{\mathsf{k}^{i^*}_{u,e_u}}(g,j) + \rho^{i^*}_{j,g,\mathsf{g},e_u}$$
$$\widetilde{g}^{j,i^*}_{\mathsf{e},e_v} = F_{\mathsf{k}^{i^*}_{v,e_v}}(g,j) + \rho^{i^*}_{j,g,\mathsf{e},e_v}.$$

Then set

$$\widetilde{g}^{i^*,i^*}_{\mathsf{g},e_u+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{u,e_u}, \gamma, (g,i^*), 0\right) + \mathsf{k}^{i^*}_{w,e_w} - \sum_{j \neq i^*} \rho^{j}_{i^*,g,\mathsf{g},e_u}$$
$$\widetilde{g}^{i^*,i^*}_{\mathsf{e},e_v+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{v,e_v}, \gamma, (g,i^*), 0\right) - e_u\mathsf{k}^{i^*}_{v,e_v} - \sum_{j \neq i^*} \rho^{j}_{i^*,g,\mathsf{e},e_v}$$

and set

$$\widetilde{g}^{i^*,i^*}_{\mathsf{g},e_u} = F_{\mathsf{k}^{i^*}_{u,e_u}}(g,i^*) + \mathsf{k}^{i^*}_{w,e_w} - \sum_{j \neq i^*} \rho^{j}_{i^*,g,\mathsf{g},e_u}$$
$$\widetilde{g}^{i^*,i^*}_{\mathsf{e},e_v} = F_{\mathsf{k}^{i^*}_{v,e_v}}(g,i^*) - e_u\mathsf{k}^{i^*}_{v,e_v} - \sum_{j \neq i^*} \rho^{j}_{i^*,g,\mathsf{e},e_v}.$$

Now observe that all queries to the oracle $\mathcal{O}$ are legal because:

- In all queries, $\gamma \neq 0$.
- Although the fourth entry is 0 for all queries, the remainder of the query message is different in every query.

Thus both requirements on the oracle queries are met.

We now show that the execution of $\mathcal{S}'$ with $\mathcal{Z}$ is the same as hybrid-world and ideal-world executions with $\mathcal{S}$.

Claim. The execution of $\mathcal{Z}$ with $\mathcal{S}'$ when $\mathcal{O} = \mathcal{F}_{\mathsf{RO}}$ is indistinguishable from the execution of $\mathcal{Z}$ with $\mathcal{S}$ in the ideal world.

Proof. If the oracle is $\mathcal{O} = \mathcal{F}_{\mathsf{RO}}$, then in the execution with $\mathcal{S}'$, every ciphertext indexed by i^* is a one-time-pad encryption. All the keys in $\mathcal{S}$ are uniformly sampled (i.e., they are not generated using a global random difference), so if there is a distinguisher between the distribution of each set of p ciphertexts under the keys $\{\mathsf{k}_{w,\gamma}^{i^*}\}_{\gamma \in p}$ and the uniform distribution, then there is a distinguisher for the PRF. Since this does not exist by assumption, the claim follows. ■

Claim. The execution of $\mathcal{Z}$ with $\mathcal{S}'$ when $\mathcal{O} = \mathcal{O}_{F,R}$ is indistinguishable from an execution in the $\mathcal{F}_{\mathsf{Commit}}$,$\mathcal{F}_{\mathsf{COPE}}$,$\mathcal{F}_{\mathsf{MPC}}$,$\mathcal{F}_{\mathsf{Rand}}$-hybrid world.

Proof. It is easy to verify that if the oracle is $\mathcal{O} = \mathcal{O}_{F,R}$, then the ciphertexts generated according to the simulation with $\mathcal{S}'$ follow exactly the distribution as in a $\mathcal{F}_{\mathsf{Commit}}$,$\mathcal{F}_{\mathsf{COPE}}$,$\mathcal{F}_{\mathsf{MPC}}$,$\mathcal{F}_{\mathsf{Rand}}$-hybrid-world execution, where the honest party P_{i^*} has global difference R that is the same as the fixed value R in the oracle's definition, where effectively the key $\mathsf{k}_{u,e_u}^{i^*}$ is sampled instead of the key $\mathsf{k}_{u,0}^{i^*}$, in order to make the oracle queries legal (which makes no difference to the distribution of the resulting keys). ■

Now we define the distinguisher $\mathcal{D}$ to execute the environment against the simulator $\mathcal{S}'$ using the oracle $\mathcal{O}$ provided the challenger of its game. If the environment guesses the execution was the ideal world, then the distinguisher guesses $b' = 0$; if the environment guesses the execution was the hybrid world, then the distinguisher guesses $b' = 1$. Thus if there is an environment that can distinguish between worlds with non-negligible advantage, then the distinguisher $\mathcal{D}$ defined above wins the security game defined above that breaks the correlation-robustness assumption on the PRF, and thus the protocol UC-securely realises $\mathcal{F}_{\mathsf{AC}}$ assuming the PRF satifies correlation-robustness. ■

Subprotocol Π_{Garble}

For simplicity, the session identifiers for functionalities are taken as implicit.

Initialise

1. Agree on a new session identifer, a computational and statistical security parameter, κ and σ, and a circuit C to evaluate, with circuit input wires W_{IN}, circuit output wires W_{OUT}, and a set of gates G comprised of a set of multiplication gates G_{MUL}, a set of addition gates G_{ADD}, and a set of selection gates G_{SEL}. Let $\texttt{PID}() : W_{\text{IN}} \to [n]$ denote the map determining which party provides input on which wire.
2. Set $\ell_\kappa := \lceil \kappa / \log p \rceil$.
3. For each $i \in [n]$, P_i samples $R^i \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and then the parties execute the procedure **Initialise** from $\Pi_{\mathsf{Mask} \times \mathsf{Diff}}$.
4. Call an instance of $\mathcal{F}_{\mathsf{MPC}}$ with input $(\mathsf{Initialise}, \mathbb{F}_p, \mathsf{sid})$.

Wire Masks and Keys

Circuit Input Wires For circuit input wire $w \in W_{\text{IN}}$, let $i := \texttt{PID}(w)$ and then do the following:

1. Party P_i samples $\lambda_w \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with this value as input.
2. Each party P_j, $j \in [n]$, samples a key $\mathsf{k}^j_{w,0} \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and for each $\alpha \in \mathbb{F}_p$ sets $\mathsf{k}^j_{w,\alpha} := \mathsf{k}^j_{w,0} + \alpha \cdot R^j$.

Addition Output Wires For each wire w that is an output of an addition gate with input wires u and v, do the following:

1. Compute $[\![\lambda_w]\!] = [\![\lambda_u + \lambda_v]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
2. For each $i \in [n]$, party P_i computes $\mathsf{k}^i_{w,0} := \mathsf{k}^i_{u,0} + \mathsf{k}^i_{v,0}$ and for each $\alpha \in \mathbb{F}_p$ sets $\mathsf{k}^i_{w,\alpha} := \mathsf{k}^i_{w,0} + \alpha \cdot R^i$.

Multiplication Output Wires For a wire w that is an output of a multiplication gate with input wires u and v,

1. For each $\mathsf{x} \in \{\mathsf{g}, \mathsf{e}\}$,
   (a) For each $i \in [n]$, party P_i samples $\lambda^i_{w,\mathsf{x}} \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with this value as input.
   (b) Compute $[\![\lambda_{w,\mathsf{x}}]\!] := [\![\sum_{i=1}^n \lambda^i_{w,\mathsf{x}}]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
   (c) For each $i \in [n]$, party P_i samples a key $\mathsf{k}^i_{w,\mathsf{x},0} \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and for each $\gamma \in \mathbb{F}_p$ sets $\mathsf{k}^i_{w,\mathsf{x},\gamma} := \mathsf{k}^i_{w,\mathsf{x},0} + \gamma \cdot R^i$.
2. For each $i \in [n]$, party P_i sets $\mathsf{k}^i_{w,0} := \mathsf{k}^i_{w,\mathsf{g},0} + \mathsf{k}^i_{w,\mathsf{e},0}$ and for all $\gamma \in \mathbb{F}_p$ sets $\mathsf{k}^i_{w,\gamma} := \mathsf{k}^i_{w,0} + \gamma \cdot R^i$.

Wire Mask/Global Difference Products

Multiplication Gates For each $g \in G_{\text{MUL}}$, let u and v be the input wires and w the output wire; then do the following:

1. Compute $[\![\lambda_{uv}]\!] := [\![\lambda_u \cdot \lambda_v]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
2. Execute the procedure **Multiply** from $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ on the set $\{\lambda_u, \lambda_v, \lambda_{uv}, \lambda_{w,\mathsf{g}}, \lambda_{w,\mathsf{e}}\}_{g\in G_{\text{MUL}}}$ to obtain, for all $i \in [n]$, (unauthenticated) sharings
$$\Big\{\langle R^i \cdot \lambda_u\rangle, \langle R^i \cdot \lambda_v\rangle, \langle R^i \cdot \lambda_{uv}\rangle, \langle R^i \cdot \lambda_{w,\mathsf{g}}\rangle, \langle R^i \cdot \lambda_{w,\mathsf{e}}\rangle\Big\}_{g\in G_{\text{MUL}}}.$$
3. For each $i \in [n]$, for each $\gamma \in \mathbb{F}_p$, set
$$\langle \rho_{i,g,\mathsf{g},\gamma}\rangle := -\gamma \cdot \langle R^i \cdot \lambda_v\rangle + \langle R^i \cdot \lambda_{uv}\rangle + \langle R^i \cdot \lambda_{\mathsf{g},w}\rangle$$
$$\langle \rho_{i,g,\mathsf{e},\gamma}\rangle := -\gamma \cdot \langle R^i \cdot \lambda_u\rangle + \langle R^i \cdot \lambda_{\mathsf{e},w}\rangle$$

Garbling

Multiplication Gates For each $g \in G_{\text{MUL}}$, for each $i \in [n]$, for each $\gamma \in \mathbb{F}_p$,

1. The parties compute the garbler half gate:
   – P_i sets $\widetilde{g}^{i,i}_{\mathsf{g},\gamma} := F_{\mathsf{k}^i_{u,\gamma}}(g,i) + \mathsf{k}^i_{w,\mathsf{g},0} + \rho^i_{i,g,\mathsf{g},\gamma}$
   – For every $j \neq i$, P_j sets $\widetilde{g}^{i,j}_{\mathsf{g},\gamma} := F_{\mathsf{k}^j_{u,\gamma}}(g,i) + \rho^j_{i,g,\mathsf{g},\gamma}$
2. The parties compute the evaluator half gate:
   – Party P_i sets $\widetilde{g}^{i,i}_{\mathsf{e},\gamma} := F_{\mathsf{k}^i_{v,\gamma}}(g,i) + \mathsf{k}^i_{w,\mathsf{e},0} - \gamma \cdot \mathsf{k}^i_{u,0} + \rho^i_{i,g,\mathsf{e},\gamma}$
   – Every party P_j, $j \neq i$, sets $\widetilde{g}^{i,j}_{\mathsf{e},\gamma} := F_{\mathsf{k}^j_{v,\gamma}}(g,i) + \rho^j_{i,g,\mathsf{e},\gamma}$

Fig. 7: Subprotocol Π_{Garble} for garbling a circuit.

Subprotocol Π_{Eval}

Input Wires For each wire $w \in W$ which is an input wire, the parties do the following:

1. Let $i = \mathtt{PID}(w)$: then party P_i computes and broadcasts $e_w := v_w + \lambda_w$, where $v_w \in \mathbb{F}_p$, is P_i's input.
2. For each $i \in [n]$, party P_i broadcasts k^i_{w,e_w}.

Opening For each $g \in G_{\text{MUL}}$, for each $\mathsf{x} \in \{\mathsf{g}, \mathsf{e}\}$, for each $i \in [n]$,

1. For each $j \in [n]$, for each $\gamma \in \mathbb{F}_p$, P_i broadcasts $\widetilde{g}^{j,i}_{\mathsf{x},\gamma}$.
2. All parties compute $\widetilde{g}^{i}_{\mathsf{x},\gamma} := \sum_{j=1}^{n} \widetilde{g}^{i,j}_{\mathsf{x},\gamma}$.

Circuit Evaluation Traversing the circuit in topological order, for every gate G with input wires u and v and output wire w, the parties do the following:

- If g is an addition gate, each party does the following:
  1. Set the external wire value to be $e_w := e_u + e_v$.
  2. Compute the output keys as: for each $i \in [n]$, $\mathsf{k}^i_{w,e_w} := \mathsf{k}^i_{u,e_u} + \mathsf{k}^i_{v,e_v}$.
- If g is a multiplication gate, each party does the following:
  1. For each $i \in [n]$, compute

$$\mathsf{k}^i_{w,e_w} := \underbrace{\widetilde{g}^i_{\mathsf{g},e_u} - \sum_{j=1}^{n} F_{\mathsf{k}^j_{u,\mathsf{g},e_u}}(g,i)}_{\text{Garbler half gate}} + \underbrace{\widetilde{g}^i_{\mathsf{e},e_v} - \sum_{j=1}^{n} F_{\mathsf{k}^j_{v,\mathsf{e},e_v}}(g,i)}_{\text{Evaluator half gate}} + e_v \cdot \mathsf{k}^i_{u,e_u}.$$

  2. Each party P_i determines the signal value e_w by computing $e_w := (\mathsf{k}^i_{w,e_w} - \mathsf{k}^i_{w,0}) \cdot (R^i)^{-1}$.

Output To obtain the output of wire $w \in W_{\text{OUT}}$, call $\mathcal{F}_{\mathsf{MPC}}$ to execute the procedure **Output** to reveal the value $v_w - [\![\lambda_w]\!]$.

Fig. 8: Subprotocol Π_{Eval} for evaluating the garbled circuit.

All of the protocols in this section can be realised using protocols (with minor modifications) given in MASCOT [KOS16]; however, the two parts of the computation outlined above are most optimally performed using a mixed approach: using the Overdrive protocol [KPR18] to realise $\mathcal{F}_{\mathsf{MPC}}$, and using MASCOT-like protocols to perform the Wire Mask/Global Difference products. The reason is that Overdrive is more efficient over large prime fields, as opposed to large extension fields such as $\mathbb{F}_{2^k}$ for which MASCOT is better.

4 Selector Gate

It was argued by Ben-Efraim [Ben18] that a selector gate taking a Boolean selection bit and choosing between field elements is a desirable feature of garbling protocols as the selection bit is likely to come from the evaluation of some Boolean subcircuit. Such a construction was given in [Ben18]; in this section we give an alternative construction, which we call the alternative selector gate, which, specifically, takes one input in $\mathbb{F}_2$, held as a signal bit with a corresponding key in $\mathbb{F}_{2^\kappa}$ and viewed as output from a Boolean circuit, and two inputs in

$\mathbb{F}_p$, and outputs one of the field elements according to the selection bit. Note that if the selection bit is also a field element then the standard $2 \cdot p$ ciphertexts for general field/field multiplication is required, as is the case in Ben-Efraim's work [Ben18].

Multifield Shared Bits Rotaru and Wood [RW19] showed how to generate secret-sharings of uniformly-random bits shared in two fields with authentication in each; these were called daBits, for doubly-authenticated bits. This can be viewed as an actively-secure version of the multi-field bits discussed by Ben-Efraim, which can be used in arithmetic garbling of selector gates. The protocol for generating such bits uses authentication in a black-box way, and so any actively-secure MPC protocol can be used to generate them. In this work, we use daBits shared in $\mathbb{F}_p$ and $\mathbb{F}_{2^\kappa}$ for our selector gates.

4.1 New Selector Gate

Recall that the standard cost of multiplication in $\mathbb{F}_p$ is $p \cdot p$ ciphertexts; the garbler/evaluator half-gate approach reduces this to $p+p$ ciphertexts. The main observation driving our alternative selector gate is that the actual selection operation is a multiplication of a bit by an element in $\mathbb{F}_p$, and thus the goal is to reduce the naïve $2 \cdot p$ ciphertexts to (almost) $2 + p$.

A selection gate based on selection bit b between the values on wires u and v is computed via the standard multiplexer $u + (v - u) \cdot b$. Since linear operations are garbled without communication or additional preprocessing, we focus on the product of the wire $w := v - u \in \mathbb{F}_p$ with the bit $b \in \mathbb{F}_2$; the output wire is denoted by z.

The point is that while the previous approach by Ben-Efraim involved converting the bit to $\mathbb{F}_p$ using a so-called *projection gate* and evaluating a standard multiplication gate in $\mathbb{F}_p$, we can use daBits to perform this projection directly. We will now explain how to garble the new selector gate; this explanation is followed by a formal protocol description.

Let b' be the Boolean wire, and let b be the $\mathbb{F}_p$ wire to which we wish to convert. We let the wire mask output of the Boolean wire be a daBit $\lambda_{b'} \in \{0,1\}$ and convert it to an $\mathbb{F}_p$ wire using $2n$ ciphertexts in $\mathbb{F}_{p^{\ell_\kappa}}$ as follows: for every $\beta \in \{0,1\}$, for every $j \in [n]$,

$$g^j_\beta := \sum_{i=1}^{n} F_{\mathsf{k}^i_{b',\beta}}(g,j) + \mathsf{k}^j_{b,0} + ((\beta + \lambda_{b'} - 2 \cdot \beta \cdot \lambda_{b'}) + \lambda_b) \cdot R^j,$$

where $\mathsf{k}^i_{b',\beta} \in \mathbb{F}_{2^\kappa}$ for all $i \in [n]$ and λ_b is a uniform mask in $\mathbb{F}_p$. Since the PRF used in previous sections takes keys of length at least κ bits, we may assume the same PRF is used here, with additional padding if necessary. Here, we use the fact that in any field, if a and b are in $\{0,1\}$ then their XOR is computed as

$$a \oplus b = a + b - 2 \cdot a \cdot b,$$

which means that we can remove the mask in $\mathbb{F}_p$ since the mask $\lambda_{b'}$ used in the garbling of the Boolean circuit was a daBit. The two ciphertexts (for each $i \in [n]$) are indexed by the two possible Boolean external values, which is denoted by $e_{b'}$; the external value on the output, denoted by e_b, is not needed in the next steps, but can be computed by the evaluators in the usual way (i.e., by P_i comparing the output key indexed by i to its own p keys). In doing so, the evaluators learn either $0 + \lambda_b$ or $1 + \lambda_b$, but do not learn which they hold. In fact, this external value e_b is never used by the evaluators.

The multiplication gate is then computed in two halves:

$$g^j_{\mathsf{g},\alpha} := \sum_{i=1}^{n} F_{\mathsf{k}^i_{w,\alpha}}(g,j) + \mathsf{k}^j_{\mathsf{g},z,0} - \alpha(\mathsf{k}^j_{b,0} + \lambda_b R^j)$$

$$g^j_{\mathsf{e},\beta} := \sum_{i=1}^{n} F_{\mathsf{k}^i_{b,\beta}}(g,j) + \mathsf{k}^j_{\mathsf{e},z,0} - (\beta + \lambda_{b'} - 2\beta\lambda_{b'})\lambda_w R^j + \lambda_z R^j$$

Now when evaluating, the parties will obtain e_w and $e_{b'}$, will compute $a := \mathsf{Dec}(g_{\mathsf{g},e_w})$ and $b := \mathsf{Dec}(g_{\mathsf{e},e_{b'}})$ and will compute

$$\begin{aligned}
\mathsf{k}_{z,e_z} &= a + b + e_w \mathsf{k}_{b,e_b} \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - e_w(\cancel{\mathsf{k}^j_{b,0}} + \lambda_b R^j)\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - (e_{b'} + \lambda_{b'} - 2e_{b'}\lambda_{b'})\lambda_w R^j + \lambda_z R^j\right) \\
&\qquad + e_w(\cancel{\mathsf{k}_{b,0}} + e_b R^j) \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - e_w\lambda_b R^j\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - v_b\lambda_w R^j + \lambda_z R^j\right) + e_w e_b R^j \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - (v_w + \lambda_w)\lambda_b R^j\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - v_b\lambda_w R^j + \lambda_z R^j\right) \\
&\qquad + (v_w + \lambda_w)(v_b + \lambda_b)R^j \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - \cancel{(v_w + \lambda_w)\lambda_b R^j}\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - v_b\lambda_w R^j + \lambda_z R^j\right) \\
&\qquad + ((v_w + \lambda_w)v_b + \cancel{(v_w + \lambda_w)\lambda_b})R^j \\
&= \mathsf{k}^j_{\mathsf{g},z,0} + \left(\mathsf{k}^j_{\mathsf{e},z,0} + \lambda_z R^j - \cancel{v_b\lambda_w R^j}\right) + ((v_w + \cancel{\lambda_w})v_b)R^j \\
&= \mathsf{k}^j_{z,0} + (v_w v_b + \lambda_z)R^j.
\end{aligned}$$

In total, this requires $p + 4$ ciphertexts per party: 2 for the conversion, 2 for the first half gate and p for the second.

We do not provide a complete proof of the security of the alternative selector gate as it follows straightforwardly from the security of the selector gate of Ben-Efraim [Ben18]. However, to give a high-level intuition, consider that the keys $\mathsf{k}^j_{\mathsf{g},z,0}$ and $\mathsf{k}^j_{\mathsf{e},z,0}$ are sampled uniformly at random, and independently of one another, and so their sum $\mathsf{k}_{z,0}$ is also uniformly random, as is required of 0 keys; furthermore, the wire mask λ_z is uniform and not known to any individual party, so the external value of the output wire perfectly hides the real value $v_w \cdot v_b$. The complete protocol for garbling these new selector gates is given in Figure 9.

The evaluation protocol is the same as the evaluation of a multiplication gate and is therefore omitted.

Subprotocol Π_{Select}

This subprotocol takes a gate with Boolean input wire b' and arithmetic inputs u and v and output wire $z = u + (v - u) \cdot b'$.

Wire Masks and Keys

Wire Mask/Global Difference Products

Selection Gates If g is a selection gate with input wires u and v, selection bit wire b, and output wire z,

1. If b' is the Boolean input wire, let $\lambda_{b'}$ be the $\mathbb{F}_p$ daBit mask stored as $[\![\lambda'_b]\!]$ in $\mathcal{F}_{\mathsf{MPC}}$.
2. Generate an $\mathbb{F}_p$ wire mask $[\![\lambda_b]\!]$:
   (a) For each $i \in [n]$, party P_i samples $\lambda_b^i \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with this value as input.
   (b) Compute $[\![\lambda_b]\!] := [\![\sum_{i=1}^n \lambda_b^i]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
3. Each party P_i samples a key $\mathsf{k}_{b,0}^i \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$.
4. Generate an $\mathbb{F}_p$ output wire mask $[\![\lambda_z]\!]$ in the same way as for $[\![\lambda_b]\!]$, above.
5. Let $[\![\lambda_u]\!]$ and $[\![\lambda_v]\!]$ be the masks stored in $\mathcal{F}_{\mathsf{MPC}}$ for wires u and v, respectively, generated when garbling an addition or multiplication gate or input wire. Set $[\![\lambda_w]\!] := [\![\lambda_v]\!] - [\![\lambda_u]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
6. Let $\mathsf{k}_{u,0}^i$ and $\mathsf{k}_{v,0}^i$ be the keys previously generated by party P_i for wires u and v. For each $i \in [n]$, party P_i sets $\mathsf{k}_{w,0}^i := \mathsf{k}_{v,0}^i - \mathsf{k}_{u,0}^i$.
7. For each $i \in [n]$, party P_i samples a wire key $\mathsf{k}_{w,\mathsf{g},0}^i \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and sets $\mathsf{k}_{w,\mathsf{e},0}^i := \mathsf{k}_{w,0}^i - \mathsf{k}_{w,\mathsf{g},0}^i$.
8. Compute $[\![\lambda_{b'w}]\!] := [\![\lambda_{b'} \cdot \lambda_w]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
9. Execute $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ to obtain
$$\Big\{\langle R^i \cdot \lambda_{b'}\rangle, \langle R^i \cdot \lambda_b\rangle, \langle R^i \cdot \lambda_w\rangle, \langle R^i \cdot \lambda_{b'w}\rangle, \langle R^i \cdot \lambda_z\rangle\Big\}.$$
10. For each $i \in [n]$, for each $\alpha \in \mathbb{F}_p$ and $\beta \in \{0,1\}$, set
$$\langle \rho_{i,g,b,\beta}\rangle := (1 - 2 \cdot \beta) \cdot \langle R^j \cdot \lambda_{b'}\rangle + \langle R^j \cdot \lambda_b\rangle$$
$$\langle \rho_{i,g,\mathsf{g},\alpha}\rangle := -\alpha \cdot \langle R^j \cdot \lambda_b\rangle$$
$$\langle \rho_{i,g,\mathsf{e},\beta}\rangle := -\beta \cdot \langle R^j \cdot \lambda_w\rangle - (1 - 2 \cdot \beta) \cdot \langle R^j \cdot \lambda_{b'w}\rangle + \langle R^j \cdot \lambda_z\rangle.$$

Garbling

Selection Gates If g is a selection gate with input wires u and v and selection bit wire b,

1. The parties generate ciphertexts for converting the Boolean input wire b' to an $\mathbb{F}_p$ wire b: for every $i \in [n]$,
   – P_i sets $\widetilde{g}_{b,\beta}^{i,i} := F_{\mathsf{k}_{b,\beta}^i}(g,i) + \mathsf{k}_{b,0}^i + \beta \cdot R^i + \rho_{i,g,b,\beta}^i$
   – For every $j \neq i$, P_j sets $\widetilde{g}_{b,\beta}^{i,j} := F_{\mathsf{k}_{b,\beta}^i}(g,i) + \rho_{i,g,b,\beta}^j$

2. The parties compute the gates for the product of wire $w := (v - u)$ with wire b:
   (a) The parties compute the garbler half gate:
      – P_i sets $\widetilde{g}^{i,i}_{\mathsf{g},\alpha} := F_{\mathsf{k}^i_{\mathsf{g},z,\alpha}}(g,i) + \mathsf{k}^i_{\mathsf{g},z,0} - \alpha \cdot \mathsf{k}^i_{b,0} + \rho^i_{i,g,\mathsf{g},\alpha}$
      – For every $j \neq i$, P_j sets $\widetilde{g}^{i,j}_{\mathsf{g},\alpha} := F_{\mathsf{k}^j_{\mathsf{g},z,\alpha}}(g,i) + \rho^j_{i,g,\mathsf{g},\alpha}$
   (b) The parties compute the evaluator half gate:
      – P_i sets $\widetilde{g}^{i,i}_{\mathsf{e},\beta} := F_{\mathsf{k}^i_{\mathsf{e},b,\beta}}(g,i) + \mathsf{k}^i_{\mathsf{e},z,0} + \rho^i_{i,g,\mathsf{e},\beta}$
      – For every $j \neq i$, P_j sets $\widetilde{g}^{i,j}_{\mathsf{e},\beta} := F_{\mathsf{k}^j_{\mathsf{e},b,\beta}}(g,i) + \rho^j_{i,g,\mathsf{e},\beta}$

Fig. 9: Subprotocol Π_{Select} for garbling a selector gate.

5 Evaluation in comparison to previous work

We evaluate our work in comparison to all previous works in the field of arithmetic garbling; both in the two-party, and in the multiparty paradigm. As shown in Table 1, we are the only work providing full-threshold active security, and proving our garbling techniques UC-secure under the named assumptions. Previous work provided either two-party, passively secure constructions [BMR16, BCM+19], or multiparty, passively secure constructions in the honest majority setting [Ben18].

Recall that in the multiparty setting projection gates (significantly increasing the efficiency of previous work [BMR16, BCM+19]) are non-trivial to construct, and they are not universal (i.e., tailored techniques per gate are required). In general, given that in the multiparty setting all parties play the role of the garbler, we cannot exploit the asymmetry between garbler and evaluator that two-party solutions enjoy. In addition, as already pointed out by Ben-Efraim [Ben18], each garbled table row in the multiparty setting requires n ciphertexts, versus a single ciphertext in the two-party setting, and each row decryption requires n^2 PRF calls (indicated as #Decryptions in Table 1) in the multiparty setting, versus a single PRF call in the two-party setting. These values are reflected in our cost description provided in Table 1.

For the works that did not suggest an improved version of a specific garbled gate (e.g., multiplication gates in both our work, and the work of Ball et al. [BCM+19]), we assume the same cost as the cost of the best previous technique of which they make use. Our work almost halves the communication cost of the selector gate, compared to the previous work in the multiparty setting [Ben18], at the cost of losing the $\sim 33\%$ improvement of computation cost that Ben-Efraim's approach enjoys (in addition to the generation of daBits). This is an overall improvement, given that the main bottleneck is the communication cost, and that the computation cost is dominated by hash function calls, which are efficient. Garbling is a technique suitable for secure computation over unreliable networks, where continuous connectivity cannot be guaranteed. Although most of the communication happens during the preprocessing phase,

the communication cost remains the main bottleneck of garbling. Performing one additional PRF call during the online phase, given that it comes at such a significant efficiency increase of the offline phase, is less of a concern, since PRF are a symmetric primitive, with significant hardware optimisations on modern processors. Our selector gate remains competitive even with the related work in the two party setting [BCM+19], where we consider a selector gate to be the so-called cross-modulus multiplication for $q = 2$. We require $p + 4$ ciphertexts per party, while Ball et al. [BCM+19] require $p + 1$. This minor difference comes mainly from the fact that we cannot deploy the row reduction techniques in the multiparty setting.

Protocol	Model	Parties	Multiplication		Selection	
			#Ciphertexts	#Decryptions	#Ciphertexts	#Decryptions
[BMR16]	passive	2	$6p - 5$	6	$2p - 1$	2
[Ben18]	passive	n	$2p \cdot n$	$2n^2$	$(2p + 2) \cdot n$	$2n^2$
[BCM+19]	passive	2	$6p - 5$	6	$p + 1$	2
Ours	active	n	$2p \cdot n$	$2n^2$	$(p + 4) \cdot n$	$3n^2$

Table 1: Comparison of our garbling techniques with the garbling of [BMR16], [Ben18], and [BCM+19], in terms of security model supported, number of parties supported, number of ciphertexts required per multiplication and selection gate, and number of decryptions required per multiplication and selection gate.

6 Conclusion

Our work continues the study of multiparty arithmetic garbling initiated by Ben-Efraim [Ben18]. Specifically, we extend the previous work from the semi-honest, honest majority setting, to the full-threshold actively-secure setting. Given the practical importance of circuits, which combine Boolean and arithmetic gates, we follow this paradigm, also considered in the work of Ben-Efraim [Ben18]. In this work we consider a selector gate as suggested by Ben-Efraim [Ben18] (essentially a multiplexer); we extend it to the full-threshold actively-secure equivalent, and show how to garble such a gate, while almost halving the communication cost it incurs.

Representations of Boolean circuits have clear advantages over arithmetic circuits when it comes to non-linear operations. On the other hand, appropriate representations of arithmetic circuits are orders of magnitude more efficient than Boolean circuits for linear operations on arithmetic values. Garbling techniques that enable the construction of circuits, which integrate both Boolean and arithmetic gates, are essential to treat numerous real-world application scenarios, and allow computation of arbitrary circuits in constant rounds. This is the reason why the design of such garbling schemes is on the rise. It remains an interesting open problem to devise techniques that allow a seamless and efficient conversion between the two representations with active security in the multiparty setting.

References

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

AOR+19. Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P Smart, and Tim Wood. Zaphod: Efficiently combining lsss and garbled circuits in scale. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 33–44, 2019.

BCM+19. Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. Cryptology ePrint Archive, Report 2019/338, 2019. https://eprint.iacr.org/2019/338.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

Ben18. Aner Ben-Efraim. On multiparty garbling of arithmetic circuits. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 3–33, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

BLN+15. Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/2015/472.

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

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

BMR16. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press.

Can00. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/2000/067.

CDE+18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer*

Science, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

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

GNN17. Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 629–659, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

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

IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

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

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

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

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

MPs16. Tal Malkin, Valerio Pastro, and abhi shelat. An algebraic approach to garbling. *Unpublished manuscript*, 2016.

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

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conference on Electronic Commerce, EC '99*, pages 129–139. Citeseer, 1999.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

RW19. Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. Cryptology ePrint Archive, Report 2019/207, 2019. https://eprint.iacr.org/2019/207.

WRK17. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 39–56, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

A Proof of Lemma 1

Proof. The proof follows the same line as that of Hazay et al. [HSS17, Lem 3.1] and Cramer et al. [CDE+18, Thm 3]. There are two possible ways to cheat in the protocol: (a) one or more parties use different inputs to the initialisation of $\mathcal{F}_{\mathsf{COPE}}$ with different honest parties; (b) one or more parties use different shares x_k^i with different honest parties. The idea is to show that if these errors are non-zero then the protocol aborts with overwhelming probability. To do this, we will show that the probability that the checks pass but a non-zero error has been introduced is negligible.

Fix an honest party P_{i^*} arbitrarily[2] and for each $i \neq i^*$ let R^i be the global difference provided by P_i into the instance $\mathcal{F}_{\mathsf{COPE}}^{i^*,i}$ and let x_k^i be the input of P_i

[2] It can be shown that if there are multiple honest parties then the sum of the errors subsequently defined with respect to P_{i^*} is well-defined (i.e., the sum is independent of the choice of "reference" honest party).

into the instance $\mathcal{F}^{i,i^*}_{\mathsf{COPE}}$. Then for each $i \neq i^*$, for each $j \neq i$ let $\widetilde{R}^{j,i}$ be the input of P_j into the instance $\mathcal{F}^{i,j}_{\mathsf{COPE}}$, or the local contribution to $z^{j,j}_k$ when $i = j$, and let $\varepsilon^{j,i} := \widetilde{R}^{j,i} - R^j$. Similarly, for each $j \neq i^*$, for each $i \neq j$ let $\widetilde{x}^{j,i}_k$ be the input of P_j into the instance $\mathcal{F}^{j,i}_{\mathsf{COPE}}$, or the local contribution to $z^{j,j}_k$ when $i = j$, and let $\delta^{j,i} := \widetilde{x}^{j,i}_k - x^j_k$.

Let ε^j be the error introduced when creating $c^{j,j}$ and ε_l be the sum of the errors introduced just before the l^{th} commitment. Now since the checks pass, for all $j \in [n]$ and all $l \in [\ell_\sigma]$ it holds that

$$
\begin{aligned}
0 &= \sum_{i=1}^{n} c^{i,j}_l = c^{j,j}_l + \sum_{i \neq j} c^{i,j}_l \\
&= \left(\varepsilon_l - c_l R^j + \left(z^{j,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \cdot z^{j,j}_k\right)\right) + \sum_{i \neq j} \left(z^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \cdot z^{i,j}_k\right) \\
&= \varepsilon_l - c_l R^j + \sum_{i=1}^{n} \left(z^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \cdot z^{i,j}_k\right) \\
&= \varepsilon_l - c_l R^j + \sum_{i=1}^{n} z^{i,j}_{m+l} + \sum_{i=1}^{n}\sum_{k=1}^{m} \chi_{l,k} \cdot z^{i,j}_k \\
&= \varepsilon_l - c_l R^j + \left(\left(\widetilde{x}^{j,j}_{m+l}\widetilde{R}^{j,j} + \sum_{i \neq j} q^{j,i}_{m+l}\right) + \sum_{i \neq j} t^{i,j}_{m+l}\right) + \\
&\qquad \sum_{k=1}^{m} \chi_{l,k} \cdot \left(\left(\widetilde{x}^{j,j}_{k}\widetilde{R}^{j,j} + \sum_{i \neq j} q^{j,i}_{k}\right) + \sum_{i \neq j} t^{i,j}_{k}\right) \\
&= \varepsilon_l - c_l R^j + \left(\left(\sum_{i=1}^{n} \widetilde{x}^{i,j}_{m+l}\widetilde{R}^{j,i}\right) + \left(\sum_{k=1}^{m} \chi_{l,k} \cdot \sum_{i=1}^{n} \widetilde{x}^{i,j}_{k}\widetilde{R}^{j,i}\right)\right) \\
&= \varepsilon_l - c_l R^j + \sum_{i=1}^{n} \left(\widetilde{x}^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k}\widetilde{x}^{i,j}_k\right) \cdot \widetilde{R}^{j,i} \\
&= \varepsilon_l - \cancel{c_l R^j} + \cancel{\sum_{i=1}^{n} \left(x^i_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \cdot \sum_{i=1}^{n} x^i_k\right) R^j} + \sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k}\delta^{i,j}_k\right) R^j \\
&\qquad + \sum_{i=1}^{n} \left(x^i_{m+l} + \sum_{k=1}^{m} \chi_{l,k} x^i_k\right) \varepsilon^{j,i} + \sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k}\delta^{i,j}_k\right) \varepsilon^{j,i} \\
&= \varepsilon_l + \underbrace{\sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k}\delta^{i,j}_k\right)}_{=:\delta^j_l} R^j + \sum_{i=1}^{n} \left(x^i_{m+l} + \sum_{k=1}^{m} \chi_{l,k} x^i_k\right) \varepsilon^{j,i} \\
&\qquad + \sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k}\delta^{i,j}_k\right) \varepsilon^{j,i}
\end{aligned}
$$

Suppose P_j is an honest party; then the goal is to show that both $\delta_k^{i,j} = 0$ for all $k \in [m]$ and $i \in [n]$ and that $\varepsilon^{j,i} = 0$ for all $i \in [n]$, except with negligible probability in σ.

If $\delta_l^j \neq 0$ then since R^j is generated by the honest party, the second summand is uniform in $\mathbb{F}_{p^{\ell_\kappa}}$ and unknown to the adversary, so the probability that the adversary can choose the other errors so that the equation above holds is at most $p^{-\ell_\kappa}$.

If $\delta_l^j = 0$ then $\delta_k^{i,j} = 0$ for all $k \in [m]$ except with probability p^{-1}, since the coefficients are unknown before the errors are introduced; thus the fourth summand is also zero. (Note that these errors can be in $\mathbb{F}_{p^{\ell_\kappa}}$ but the coefficients lie in $\mathbb{F}_p$, so the greatest upper bound on the probability is only p^{-1} and not $p^{-\ell_\kappa}$.) Now again since the matrix H is sampled after the errors $\varepsilon^{j,i}$ are introduced on the global differences – i.e., the values $\chi_{l,k}$ are unknown to the adversary when introducing these errors – if one or more $\varepsilon^{j,i}$ is non-zero then the third summand is uniformly random in $\mathbb{F}_p$, so the adversary must choose each ε_l to correct the error, which can be done for a given l with probability at most p^{-1}. Thus the probability that some $\delta_k^{i,j}$ or some $\varepsilon^{j,i}$ is not zero but the equation above holds for all $l \in [\ell_\sigma]$ is at most $p^{-\ell_\sigma}$.

Thus for every honest party $j \in [n] \setminus A$, all errors $\{\delta_k^{i,j}\}$ and $\{\varepsilon^{j,i}\}$ are 0 except with probability at most $\max\{p^{-\ell_\kappa}, p^{-\ell_\sigma}\} < 2^{-\sigma}$. ■