On Multiparty Garbling of Arithmetic Circuits

Aner Ben-Efraim

Dept. of Computer Science, Ben Gurion University of the Negev, Be'er Sheva, Israel
anermosh@post.bgu.ac.il

Abstract. We initiate a study of garbled circuits that contain both Boolean and arithmetic gates in secure *multiparty* computation. In particular, we incorporate the garbling gadgets for arithmetic circuits recently presented by Ball, Malkin, and Rosulek (ACM CCS 2016) into the multiparty garbling paradigm initially introduced by Beaver, Micali, and Rogaway (STOC '90). This is the first work that studies arithmetic garbled circuits in the *multiparty* setting. Using mixed Boolean-arithmetic circuits allows more efficient secure computation of functions that naturally combine Boolean and arithmetic computations. Our garbled circuits are secure in the semi-honest model, under the same hardness assumptions as Ball et al., and can be efficiently and securely computed in constant rounds assuming an honest majority.

We first extend free addition and multiplication by a constant to the multiparty setting. We then extend to the multiparty setting efficient garbled multiplication gates. The garbled multiplication gate construction we show was previously achieved only in the two-party setting and assuming a random oracle.

We further present a new garbling technique, and show how this technique can improve efficiency in garbling selector gates. Selector gates compute a simple "if statement" in the arithmetic setting: the gate selects the output value from two input integer values, according to a Boolean selector bit; if the bit is 0 the output equals the first value, and if the bit is 1 the output equals the second value. Using our new technique, we show a new and designated garbled selector gate that reduces by approximately 33% the evaluation time, for any number of parties, from the best previously known constructions that use existing techniques and are secure based on the same hardness assumptions.

On the downside, we find that testing equality and computing exponentiation by a constant are significantly more complex to garble in the multiparty setting than in the two-party setting.

Keywords: Arithmetic Garbled Circuits, Constant Round MPC, Multiparty Garbling

1 Introduction

Garbled circuits are a fundamental cryptographic primitive, introduced by Yao in the 1980s [33]. They are used in one-time programs, key-dependent message security, homomorphic computation, verifiable computation, and more. The original motivation of garbled circuits, and to date still their main use, is for secure computation. The most practical approaches of secure two-party computation are based on garbled circuits.

Since their introduction, garbled circuits have been significantly optimized in a series of works, [29, 25, 31, 34, 26, 5, 30, 21] being a very partial list. These works reduced the size of the garbled gates and concretely improved the efficiency of garbling protocols. For example, using the free-XOR technique introduced by Kolesnikov and Schneider [25], XOR gates are "for free", meaning they incur no communication or cryptographic operations.

Due to efficiency reasons, garbled circuits were almost exclusively considered for Boolean circuits. However, there have been a few attempts to efficiently extend the ideas of garbled circuits to arithmetic circuits (in the two-party setting), e.g., [1, 28, 2]. The works of Ball et al. [2] and Malkin et al. [28] showed how to extend free-XOR to free addition and multiplication by a constant. They further showed how to efficiently garble multiplication in $\mathbb{F}_p$, for small p. Ball et al. also showed how to efficiently garble exponentiation by a constant. By combining CRT representations in a primorial modulus, Ball et al. showed that the above results extend to efficient garbling of arithmetic circuits over the *integers*.

Garbled circuits are important also for secure *multiparty* computation. The multiparty garbling paradigm was introduced by Beaver et al. [4] in the first constant round secure multiparty protocol. The first implementation of secure multiparty computation, FairplayMP [6], followed this multiparty garbling paradigm. Recently, experimental results in [8, 32] suggested that concretely efficient implementations following the multiparty garbling paradigm, such as [8] in the semi-honest model and [22, 32] in the malicious model, are more suited for secure multiparty computations over networks with high latency, such as the internet.

The adversarial model. We assume throughout that the adversary is semi-honest, i.e., follows the protocol but might try to learn private information from the messages it receives. A more realistic adversarial model is the malicious adversary, which can deviate from the protocol arbitrarily. Nevertheless, advances in semi-honest secure computation, and garbled circuits in particular, have often proved to be a significant stepping stone for later advances in the malicious model. Aside from numerous examples in the two-party setting, this was recently demonstrated also in the multiparty setting: the concretely efficient semi-honest protocols of [8] have been efficiently extended to maliciously secure protocols in [22] and [7].

We also assume an honest majority, i.e., the adversary corrupts only a strict minority of the parties. We do so in order to use the efficient constant round protocol for unbounded fan-in multiplication of Bar-Ilan and Beaver [3], which is needed in several of our constructions. Note that this is only needed for some of the gates. Thus, arithmetic circuits can also be garbled efficiently in the *dishonest majority* setting, assuming oblivious transfer. This is shown in Appendix F.

The hardness assumption we rely on is the existence of a mixed-modulus circular correlation robust (MMCCR) hash function, introduced by Ball et al. [2]; see Definition 1. The definition of MMCCR hash functions is similar to the definition of circular 2-correlation robust hash functions, introduced by Choi et al. [12] to prove the security of free-XOR [25]. Ball et al. [2] conjecture that one could use AES for a MMCCR hash function. Using AES is known to be extremely fast in practice using AES-NI instructions.

Our results and techniques. We study garbled circuits containing both a Boolean part and an arithmetic part in secure *multiparty* computation. We show both how to compute efficient arithmetic garbled circuits, and also how the Boolean and arithmetic parts of the circuit can efficiently affect each other. This allows for more efficient secure multiparty computation of functions that naturally combine Boolean and arithmetic computations, see our motivating example below.

We begin by extending known results for garbled arithmetic circuits from the two-party setting to the multiparty setting. In the *two-party* setting, Ball et al. [2] and Malkin et al. [28] showed that the free-XOR idea of Kolesnikov and Schneider [25] can be extended to free addition and multiplication by a constant in $\mathbb{F}_p$. We show that these results naturally extend to the multiparty setting. This follows similar lines to the extension of free-XOR to the multiparty setting by Ben-Efraim et al. [8].

We further efficiently extend the half-gates construction of Zahur et al. [34] (which efficiently compute AND gates in the 2-party setting) to efficient multiplication gates in the multiparty setting. Using the half-gate extension, we manage to garble multiplication gates in $\mathbb{F}_p$, in the multiparty setting, with only $2p$ garbled rows. By representing numbers in a primorial modulos and using the Chinese Remainder Theorem, as suggested by Ball et al. [2], we obtain efficient arithmetic computations over the integers. In the two-party setting, efficient garbled multiplication gates were previously suggested by Malkin et al. [28] and by Ball et al. [2], see Table 1.

We then show how the Boolean and arithmetic parts of the circuit efficiently affect each other. In the multiparty setting, this requires a simple primitive that we call a multifield-shared bit, in which the same bit is secret shared in multiple fields of different characteristics. We show an efficient protocol for constructing this primitive in the semi-honest model with an honest majority. Furthermore, we explain that this primitive can be precomputed before the circuit is known.

To show how the Boolean part can efficiently affect the arithmetic part, we look at selector gates, which compute a simple "if statement": A selector gate has 3 input wires, x, y, and w_0. The wires x, y hold values in $\mathbb{F}_p$ and the wire w_0 holds a Boolean value representing the selection bit. The output wire z should equal either x or y, according to the value of the selection bit. I.e., denote the value on wire ω by v_ω, then a selector gate computes the following simple if statement: If $(v_{w_0} == 0)$ then $v_z = v_x$ else $v_z = v_y$.

We show two constructions for garbled selector gates: the first is an extension of known techniques to the multiparty setting, using projection gates from Boolean to $\mathbb{F}_p$. To the best of our knowledge, this is the best construction of selector gates using existing techniques that relies only on MMCCR hash functions.[1] Our second construction is a designated construction, using new techniques described below. This construction reduces the evaluation time by approx. 33% from the construction using projection gates.

We give an informal overview of the main ideas in the designated selector gate construction: The gate contains two components. Using the first component, the evaluator tries to compute the multiplication between the Boolean value and the values in $\mathbb{F}_p$. But since the Boolean value seen by the evaluator is not the real value on the wire, this computation possibly inserts an error. To solve this, the second component is "corrector gates". The result from the corrector gate is (freely) added in order to correct the values from the first component. However, the (possibly) inserted error depends on the value seen by the evaluator on the Boolean wire. Thus, there are in fact 2 corrector gates, and the evaluator decrypts only one of them, according to the value it sees. This raises a question of security, as a corrupt evaluator can also decrypt the "wrong" corrector gate. This issue is solved by double partitioning of the keys and permutation elements, ensuring the decrypted keys and external values on the "wrong" corrector gate leak no information, even given the correctly decrypted keys and values. To the best of our knowledge, the technique of using a double partition of the keys and permutation elements is new in this setting.

To show how the arithmetic part can affect the Boolean part, we extend to the multiparty setting the construction of Ball et al. [2] of gates that test equality. These equality gates use free subtraction and projection gates. Unfortunately, we find that garbling general projection gates, and garbling equality gates in particular, is significantly more complicated in the multiparty setting. To explain this, we note that the equations for equality gates require exponentiation. In the multiparty setting, the values needed for the offline computation are secret shared, and so this exponentiation is computed using MPC. We optimize these computations using the constant round protocol of [3]. However, this still implies that the offline time for computing equality gates is significantly slower. On the positive side, the size of garbled projection gates and their evaluation time are not affected by this, and therefore the difference in the online phase from the two-party setting is similar to the Boolean case.

A motivating example. Many real-world applications naturally use a mixture of Boolean and arithmetic computations. To illustrate the importance of mixed Boolean-arithmetic circuits, we look at a simple natural problem, the problem of conditional summation. Of course, it is possible to encode the problem as a Boolean circuit or as an arithmetic circuit. However, notice that encoding the conditions in arithmetic $0/1$ would be very inefficient when the conditions are complex. On the other hand, the summation could be expensive in Boolean, while free in an arithmetic circuit (using free addition). Therefore, a more efficient manner to perform the computation would be to compute the conditions in a Boolean circuit, then use selector gates, and finally compute the summation in an arithmetic circuit. Possibly, the conditions (which are Boolean) could decide multiplication constants instead of only $0, 1$, (i.e., weighted conditional summation) in which case multiplication gates are also required. An important application that requires this is a collaborative intrusion detection system (CIDS). Thus, a mixed Boolean-arithmetic computation is required to compute CIDS in MPC efficiently.

Comparison with previous works and techniques. Garbled multiplication gates were previously considered in the two-party setting by Malkin et al. [28] and by Ball et al. [2]. In Table 1 we compare our garbled multiplication gates with those of [2] and [28]. We compare only with 2-party garbling protocols, because previous multiparty garbling protocols did not handle arithmetic gates.[2] For sake of comparison, we also include the values of our garbled multiplication gate in the 2-party setting. The difference is that in the 2-party setting, the number of rows in each half-gate can be reduced by one, using the row reduction

[1] In the 2-party setting there is a more efficient construction based on *stronger assumptions* by Ball et al. [2], see remark 7. Nevertheless, since these stronger assumptions are currently not needed to optimize any other garbled gate, we believe it is of interest also to optimize selector gates which are secure based on MMCCR.

[2] One could of course use an encoding of arithmetic into Boolean, e.g. the CRT encoding in [1], and then apply any Boolean multiparty garbling protocol. For a comparison between encoding into Boolean and arithmetic gates as discussed here, see [2].

technique [29]. Furthermore, in the multiparty setting, each row requires n ciphertexts, and "decrypting" a row requires n^2 decryptions (hash function calls), whereas in the two-party setting each row is a single ciphertext and requires a single decryption.

In Table 2 we compare garbled selector gates using known techniques (projection gates) and the new designated construction.

Garbled Multiplication Gate

	Parties	Rows	Size	Sec. Ass.	# Dec.
[28]	2	$2p-2$	$(2p-2)\kappa$	Random Oracle[3]	2
[2]	2	$6p-5$	$(6p-5)\kappa$	MMCCR	6
New	2	$2p-2$	$(2p-2)\kappa$	MMCCR	2
New	**n**	$2p$	$(2p)\kappa \cdot n$	MMCCR	$2 \cdot n^2$

Table 1: Comparison of our garbled multiplication gates with those of [2], [28] in number of parties, number of garbled rows, total size of garbled gate, security assumption, and number of decryptions (hash function calls needed in the online phase). For high fan-in multiplication, the construction of [2] scales differently than ours, but still seems to have more rows.

Garbled Selector Gate

	Parties	Rows	Size	Sec. Ass.	# Dec.
New – Known Techniques	**n**	$2p+2$	$(2p+2)\kappa n$	MMCCR	$3 \cdot n^2$
New Technique	**n**	$2p+2$	$(2p+2)\kappa n$	MMCCR	$\mathbf{2} \cdot n^2$

Table 2: Comparison of garbled selector gates using known techniques (projection gates) and the new designated construction.

Other related works. Most protocols for securely computing arithmetic circuits follow the secret-sharing paradigm, e.g., [11, 15, 10, 17, 16, 23] to name but a few. In the secret-sharing paradigm, the parties share their inputs. Then, for each layer of the circuit, the parties interact in order to compute shares for the next layer. Thus, the number of rounds depends on the depth of the circuit. This could potentially lead to very slow online times when the circuit is very deep and the latency is large (for example over the internet), as demonstrated in [8]. Furthermore, garbled circuits are an important primitive, and therefore justify study even outside the context of secure computation. We believe this should hold true also for multiparty garbled circuits.

Hence, these works in the secret-sharing paradigm are incomparable with our work. In addition, the recent works of Damgård et al. [18] and Keller et al. [24] in the secret-sharing paradigm use gate-scrambling, which shares many ideas with garbling. Advances in garbling techniques could potentially aid these protocols.

Apart from the works of Ball el al. [2] and Malkin et al. [28], there has been another notable work that studied arithmetic garbled circuits in the two-party setting, the work of Appelbaum et al. [1]. The main result of [1] relies on LWE and is quite complex. It is unclear if the result can be efficiently extended to the multiparty setting. Their secondary result using CRT has been surpassed by the results of Ball et al. [2].

Organization. In Section 2 we review the basics of multiparty garbling and garbling of arithmetic circuits. In Section 3 we explain how to efficiently garble multiplication gates in $\mathbb{F}_p$. In Section 4 we explain on selector gates and show how to efficiently garble them. In Section 5 we prove the security of our protocols. In Appendix A we explain how to garble equality testing gates and exponentiation by a public constant.

2 Preliminaries

We assume that the reader is familiar with the BGW protocol and its improvement [9, 19]. Sections 2.3, 4, and Appendix A also use the constant round protocol for unbounded fan-in multiplication of Bar-Ilan and Beaver [3]. This protocol is nicely explained in [13, Section 4].

2.1 Notation, Conventions, and Security Assumption

We list some of the conventions and notations that we use throughout this paper. We consider a static semi-honest adversary $\mathcal{A}$ corrupting a strict minority of the parties . The circuit of the function to be computed is denoted by C, and $g \in C$ denotes both the gate and its index. The set of all wires is denoted by W, and $\mathcal{W}$ denotes the wires that are *not* outputs of "free gates" (e.g., XOR, addition, and multiplication by a constant

[3] Possibly, this construction, which is also based on an extension of half-gates, could be proven secure also based on MMCCR, using techniques later developed in [2]. However, we note that proving the extension of half-gates to multiplication gates based on MMCCR requires an additional step that was not needed in previous proofs, see Section 5.

gates). The respective sets of wires with values in $\mathbb{F}_p$ are denoted W_p and $\mathcal{W}_p$ respectively. The number of parties in the protocol is n, and $t = \lfloor \frac{n-1}{2} \rfloor$ is the bound on the number of corrupt parties. We denote the security parameter by κ. For binary fields, the keys are therefore in $\mathbb{F}_{2^\kappa}$. Notice that for characteristic p fields, keys should be in $\mathbb{F}_{p^{\kappa_p}}$, with $\kappa_p \geq \lceil \kappa / \log p \rceil$; see also Remark 1. We often abuse notation, writing $p^\kappa \stackrel{\text{def}}{=} p^{\kappa_p}$.

Throughout the paper we have computations in several fields. We often avoid mentioning the field in which the computations are carried out when this can be inferred from the equation. For example, if $\lambda \in \mathbb{F}_p$ and $\Delta_p^i \in \mathbb{F}_{p^\kappa}$ then the multiplication $\lambda \Delta_p^i$ is computed in $\mathbb{F}_{p^\kappa}$. Observe that $\mathbb{F}_p \subset \mathbb{F}_{p^\kappa}$ is a field extension, so this is well defined. We also ensure that the computation is well defined for the shares of λ and Δ_p^i; see Remark 1.

We sometimes use vector notation for the keys of the parties. For example, if each party P_i has a key $k_x^i \in \mathbb{F}_{p^\kappa}$ then we write $k_x \stackrel{\text{def}}{=} \left(k_x^1, \ldots, k_x^n\right) \in \mathbb{F}_{p^\kappa}^n$. Addition of vectors and multiplication by a constant are the standard linear algebra operations.

The hardness assumption we rely on, which we define next, is the existence of a mixed-modulus circular correlation robust hash function that we denote by H. This is the exact same assumption used by Ball et al. [2] in the two-party setting. Ball et al. conjectured that it is secure to use AES for H.

Definition 1. *Let H be a hash function, and for each p in some set of primes P let $\Delta_p \in \mathbb{F}_{p^\kappa}$. We define an oracle $\mathcal{O}_P^H$ that acts as follows:*

$$\mathcal{O}_P^H(\rho, a, b, k, \gamma, \delta) = H(k + \gamma \Delta_{p_a}, \rho) + \delta \Delta_{p_b} \tag{1}$$

where $\rho \in \mathbb{N}$, $p_a, p_b \in P$, $\gamma \in \mathbb{F}_{p_a}$, $\delta \in \mathbb{F}_{p_b}$, $k \in \mathbb{F}_{p_a{}^\kappa}$, and the output of H is interpreted as in $\mathbb{F}_{p_b{}^\kappa}$. Note that $\gamma \Delta_{p_a}$ is the inner offset and $\delta \Delta_{p_b}$ is the outer offset. Legal queries *to the oracle have inputs in the correct domains and satisfy:*

1. *The oracle is never queried with $\gamma = 0$,*
2. *For each ρ, all the queries have the same p_a, p_b, and each $\gamma \in \mathbb{F}_{p_a} \setminus 0$ is used in at most one query.*

We say that H is mixed-modulus circular correlation robust *if for all polynomial time adversaries making only legal queries to the oracle, the oracle $\mathcal{O}_P^H$, for random Δ_ps, is indistinguishable from a random function (with the same input/output domains).*

We use the shortened notation $\mathcal{F}_k(\rho) \stackrel{\text{def}}{=} H(k, \rho)$ ($\mathcal{F}$ can be thought of as a PRF). In our garbled gates, we use $\rho = g||j$ (formally, $\rho = ng + j$), where g is the index of the gate we garble and $j \in [n]$. In most gates, the key of each party is "encrypted", using $\mathcal{F}$, by all parties, see for example Equations (2) and (3).[4] We therefore use the shortened notation $Enc_{k_x}\left[\, k_z^j \,\right] \stackrel{\text{def}}{=} \left(\Sigma_{i=1}^n \mathcal{F}_{k_x^i}(g||j)\right) + k_z^j$. The outputs of the $\mathcal{F}_{k_x^i}(g||j)$'s are, in this case, assumed to be in the same field as k_z^j. "Decryption" of the above ciphertext is by subtracting $\Sigma_{i=1}^n \mathcal{F}_{k_x^i}(g||j)$.

Remark 1. In our offline protocols, the parties share both "small" field elements $\lambda \in \mathbb{F}_p$ and "large" keys/offsets $k_x^i, \Delta_p^i \in \mathbb{F}_{p^\kappa}$, with $\kappa_p \geq \lceil \kappa / \log p \rceil$. These are shared using Shamir secret-sharing scheme in fields of characteristic p (to allow linear combinations). Apart from the characteristic, there are three other requirements of the fields in which the elements, keys, and offsets are shared. The first two are always required by Shamir secret-sharing schemes.

1. The field must contain at least $n + 1$ elements.
2. The size of the field is at least the size of the domain of the secret.
3. We need to be able to multiply shares of the field element $\lambda \in \mathbb{F}_p$ with the shares of the offset $\Delta_p^i \in \mathbb{F}_{p^\kappa}$.

[4] As we explain later, it is more efficient to garble Boolean gates regularly than using half-gates in the multiparty setting. However, this requires assuming also the existence of a circular *two*-correlation robust hash function (as defined in [12]), which we denote, using shortened notation, by $\mathcal{F}_{k_1,k_2}^2$. If we garble AND gates using the half-gates construction in Section 3, this extra assumption is not needed.

In order to satisfy the first requirement, the parties share λ in a field extension $\mathbb{F}_{p^{m_p}}$ with $p^{m_p} > n$. In order to satisfy the second requirement, k_x^i and Δ_p^i are shared in $\mathbb{F}_{p^\kappa}$, as they cannot be shared in a smaller field. In order to satisfy the third requirement, it must hold also that $m_p | \kappa_p$, so that $\mathbb{F}_{p^{m_p}} \subseteq \mathbb{F}_{p^{\kappa_p}}$. One way to ensure all the requirements are met is to set $m_p = \kappa_p = \lceil \kappa / \log p \rceil$. This is not always the most efficient solution – any implementation should optimize the choice of m_p and κ_p for each p, in correspondence with the bound on the number of parties, such that they satisfy all the above requirements.

2.2 Multiparty Garbling

In the multiparty setting, the first proposal for constructing a multiparty garbled circuit was given in [4]. We extend a simplified description for the semi-honest model given in [8] to the arithmetic setting (in the field $\mathbb{F}_p$), by applying the ideas of [28, 2]. The construction of [8] allows the free-XOR ideas of [25]. In the two-party setting, Malkin et al. [28] and Ball et al. [2] showed that free-XOR extends to free addition, subtraction, and multiplication by a public constant in the field $\mathbb{F}_p$. As we shall see, this is also the case in the multiparty setting.

The multiparty garbling paradigm consists of two phases. In the first phase, often called the offline or garbling phase, the parties collaboratively construct a garbled circuit. Then, in the second phase, called the online or evaluation phase, the parties exchange masked input values and the corresponding keys. After that, each party (or a designated evaluating party) locally computes the outputs of the function. Our secure computation protocol that follows this paradigm is given in Section 2.4. We next recall the basics of the mulitparty garbling paradigm.

Boolean Circuits. For constructing the garbled circuit, each party P_i chooses, for each wire $\omega \in \mathcal{W}$, two random keys, $k_{\omega,0}^i$ and $k_{\omega,1}^i$. To enable the free-XOR technique [25], the parties need to choose the keys such that $k_{\omega,1}^i = k_{\omega,0}^i \oplus \Delta^i$ for some global offset Δ^i.

Each wire ω in the circuit is assigned a random secret permutation bit λ_ω. This bit masks the real values of the wires during the online phase. For an AND gate with input wires x, y and output wire z, the garbled gate is the encryptions $g_{\alpha,\beta} = \left(g_{\alpha,\beta}^1, \ldots, g_{\alpha,\beta}^n\right)$ for $(\alpha, \beta) \in \{0,1\}^2$, where

$$g_{\alpha,\beta}^j = \left(\bigoplus_{i=1}^{n} \mathcal{F}^2_{k^i_{x,\alpha}, k^i_{y,\beta}}(g||j) \right) \oplus k_{z,0}^j \oplus \left(\left[(\lambda_x \oplus \alpha) \cdot (\lambda_y \oplus \beta) \oplus \lambda_z \right] \Delta^j \right). \tag{2}$$

Notice that all the values are "encrypted" by *all* the parties. XOR gates are computed using the free-XOR technique of Kolesnikov and Schneider [25], which was extended to the multiparty setting in [8] – the permutation bit and keys on the output wire are set to be the XOR of those on the input wires; they require no cryptographic operations or communication. For the circuit output wires, the permutation bits are revealed. For input wires of party P_i, the corresponding permutation bits are disclosed to party P_i.

During the evaluation phase, an evaluating party learns at each wire ω a bit e_ω, called the *external* or *public* value, and the corresponding keys. The keys on the output wire of a garbled gate are recovered by decrypting the row g_{e_x,e_y} using the keys on the input wires. As was first pointed out in [27], if the evaluating party participates in the garbling (which we generally assume), the external value can be extracted from the decrypted key – an evaluating party P_i can compare the ith key with the keys it used for the garbling, and thus learn the external value. I.e., if the key is $k_{z,0}^i$ then $e_z = 0$ and if it is $k_{z,1}^i$ then $e_z = 1$.

The external value e_ω is the XOR of the real value v_ω with the random permutation bit λ_ω. Since the permutation bit is random and secret, the external value reveals nothing about the real value to the evaluating party. The evaluating party uses the external value and keys to continue the evaluation of the proceeding garbled gates. For the output wires of the circuit, the permutation bit values are revealed, and thus the output is learnt by XORing with the external values.

Extension to $\mathbb{F}_p$ Arithmetic. The above generalizes naturally to arithmetics in the field $\mathbb{F}_p$. We explain this briefly; see [2] for a detailed explanation (in the two-party setting). Instead of each wire having a

permutation bit λ, now each wire has a random secret permutation field element $\lambda \in \mathbb{F}_p$. The external value on wire ω is similarly defined $e_\omega \stackrel{\text{def}}{=} v_\omega + \lambda_\omega$. The permutation field elements are shared, using a linear secret-sharing scheme, in a field of characteristic p. Furthermore, each party P_i has a global random secret offset $\Delta_p^i \in GF(p^\kappa)$. For each wire ω, each party P_i has a random key k_ω^i. The p keys of each party P_i that relate to the p possible external values, are set to be $k_{\omega,\alpha}^i \stackrel{\text{def}}{=} k_\omega^i + \alpha\Delta_p^i$ for each $\alpha \in \mathbb{F}_p$.[5]

Thus, addition and subtraction are "free": The zero keys of the output of an addition/subtraction gate are chosen to be the sum/difference of the keys of the input wires. The permutation field element of the output wire is set to be the sum/difference of the permutation elements of the input wires. Since the keys and permutation elements are shared using a linear secret-sharing scheme in a field of characteristic p, the shares of the addition/subtraction can be computed locally by the parties (by performing local additions on their shares). Similarly, multiplication by a public constant c is also free: if $c \neq 0$, the zero keys and permutation element of the output wire are set to be the multiplication by c. Again, all the necessary computations can be performed locally by the parties, both at the garbling phase and the evaluation phase. The case of $c = 0$ is dealt using a global 0 wire.

A straightforward method for garbling multiplication gates is to extend Equation 2 from Boolean to characteristic p. I.e., for a multiplication gate with input wires x, y and output wire z, the garbled gate is the encryptions

$$g_{\alpha,\beta}^j = \left(\Sigma_{i=1}^n \mathcal{F}^2_{k^i_{x,\alpha},k^i_{y,\beta}}(g||j)\right) + k_{z,0}^j + \left(\left[(\alpha-\lambda_x)\cdot(\beta-\lambda_y)+\lambda_z\right]\Delta_p^j\right) \tag{3}$$

for every $\alpha, \beta \in \mathbb{F}_p$ and $j \in [n]$. The summations and multiplications in the above equation are carried out in $\mathbb{F}_{p^\kappa}$. Observe that for this equation to make sense, the output of $\mathcal{F}^2$ must also be in $GF(p^\kappa)$. At the online phase, the evaluator recovers the output keys by decrypting row (e_x, e_y).

The above straightforward method requires p^2 garbled rows. In Section 3 we describe a more efficient way to garble multiplication gates in the multiparty setting that requires only $2p$ garbled rows, by extending the half-gates idea of Zahur et al. [34]. Extension of half-gates to $\mathbb{F}_p$ was shown in the two-party setting by Malkin et al. [28], but their techniques are quite different from ours. Also, in the two-party setting, Ball et al. [2] suggested a different solution to garble multiplication gates in $O(p)$ garbled rows. However, their solution relies heavily on projection gates. Unfortunately, projection gates are relatively expensive to garble in the multiparty setting, as we explain in Section 2.2.2.

2.2.1 CRT Representation and Application to Arithmetic Garbled Circuits. We briefly explain the idea presented by Ball et al. [2] for constructing efficient arithmetic garbled circuits over *the integers*; see [2] for a more detailed explanation. The idea is to use the Chinese Remainder Theorem (CRT), along with efficient garbling in the field $\mathbb{F}_p$, for small p.

The computations are done in the primorial modulus $Q_k = 2 \cdot 3 \cdots p_k$, the product of the first k primes. The number of primes k is chosen such that $Q_k > Z$, where Z is the bound on the possible intemediate values of the computation. Each number is represented by a bundle of wires, one for each of the k primes. We call such a representation a CRT bundle representation. Adding two numbers is free, because the sum can be carried out in each prime separately (and addition in $\mathbb{F}_p$ is free), and similarly multiplication by a constant. Multiplication and exponentiation by a constant are also computed separately for each prime. Thus, the total number of computations and garbled rows is the sum of the computations/garbled rows in the different primes. Correctness of computing this way follows from the Chinese Remainder Theorem.

2.2.2 General Projection Gates. One of the main garbling gadgets used by Ball et al. [2] is projection gates. A projection gate is a gate which has one input wire and one output wire. For example, an exponentiation gate that computes $x \mapsto x^c$, where c is a public constant. In addition to gates $g\colon \mathbb{Z}_N \to \mathbb{Z}_N$, there are also useful projection gates in which the domain of the input wire differs from the domain of the output wire. Ball et al. [2] showed in the two party setting that any projection gate $g\colon \mathbb{Z}_{N_1} \to \mathbb{Z}_{N_2}$ can be garbled

[5] Note that $\mathbb{F}_p \subset \mathrm{GF}(p^\kappa)$ is a field extension so $\alpha \cdot \Delta_p^i$ is well defined.

using at most $N_1 - 1$ garbled rows, where the -1 comes from the row reduction technique [29]. Furthermore, they showed that, in the two-party setting, it is not difficult to compute these garbled projection gates, because the garbler knows all the information for constructing the gate. In particular, the garbler knows all the permutation bits/elements.

In contrast, in the multiparty setting, the parties only hold shares of the permutation bits/elements. Therefore, the garbled gates are computed via an MPC subprotocol with these shares. In general, garbling a projection gate might require computing a very complex equation in MPC. Projection gates in which the output domain differs from the input domain are potentially even more complex.

In Section 4.1 and Appendix A we discuss three types of projection gates: a projection identity gate from Boolean to $\mathbb{F}_p$, an equality testing gate from $\mathbb{F}_p$ to Boolean, and an exponentiation by a (public) constant gate from $\mathbb{F}_p$ to $\mathbb{F}_p$. The first gate can be computed very efficiently. On the other hand, the equality and exponentiation gates, while significantly more efficient than general projection gates, do still seem to be quite expensive. This is because there are exponentiations in the gate equations, and computing exponentiation in MPC is expensive, even using the protocols suggested in [3] or [13].

On the positive side, the number of garbled rows in our projection gates is only one row more than the respective garbled gates in the two-party construction of [2]. Thus, the size of the garbled projection gates is only slightly more than n times of the respective gate in the two-party setting. Furthermore, at the evaluation phase only a single row is decrypted. Therefore, the online computation is only about n^2 times than the two-party setting. This matches the Boolean case.

2.3 Multifield-Shared Bits

In this section we introduce a new primitive that we use in our constructions in Section 4 and Appendix A.1. Note that garbling multiplication gates does not require this primitive. The primitive is a random bit $b \in \{0,1\}$ that is shared multiple times in different fields, of different characteristics. That is, each party holds multiple shares of the same secret random bit, where each share is in a different field with a different characteristic.

In the semi-honest model with an honest majority, it is quite simple to construct this primitive. First, each party P_i chooses a random bit b_i. The secret random bit will be $b = \bigoplus_{i=1}^{n} b_i$. Note that if there is an additional requirement that $b_z = b_x \oplus b_y$ (as needed in some of our constructions to allow free XOR), then party P_i sets $(b_z)_i := (b_x)_i \oplus (b_y)_i$ instead of randomly choosing it – permutation bits/elements are chosen only for the input wires of the circuit and for output wires of garbled gates/components. Next, the parties run protocols to share b in each field; these protocols are run in parallel.

We next explain the bit-sharing protocols. The sharing we describe is of Shamir shares, which is the type of shares used in our constructions. The sharing protocol depends on the characteristic of the field. See Remark 1 regarding the fields in which the shares should be generated.

1. In characteristic 2 fields, each party P_i shares its bit b_i amongst all the parties in a $(t+1)$-out-of-n Shamir sharing. The parties sum (XOR) their received shares to obtain shares of the bit b.
2. In characteristic $p \neq 2$ fields, each party P_i shares the value $b'_i = \begin{cases} -1, & b_i = 1 \\ 1, & b_i = 0 \end{cases}$ amongst all the parties in a t+1-out-of-n Shamir sharing. Then, the parties use an MPC protocol to compute shares of $b = \frac{1-\left(\Pi_{i=1}^{n} b'_i\right)}{2}$. This is computed in constant rounds by combining the protocol of Bar-Ilan and Beaver [3] for unbounded fan-in multiplication (to compute shares of $\Pi_{i=1}^{n} b'_i$) and then linear operations on the shares (note that 2 is invertible in $\mathbb{F}_p$ and the inverse is easily computable).

Observe that b computed in both protocols is the same: $b = 0$ if and only if an even number of b_is is 1. This happens if and only if an even number of b'_is is -1, which is if and only if $\Pi_{i=1}^{n} b'_i = 1$, so if and only if $\frac{1-\Pi_{i=1}^{n} b'_i}{2} = 0$. The case of $b = 1$ is similar.

Remark 2. In our description of the protocol, we assume the parties know the circuit when computing the multifield-shared bits. However, it is possible to compute all the multifield-shared bits even before the function is known, at almost no extra cost. This is explained in Appendix H.

Remark 3. It seems that a more natural method to construct this primitive would be to replace protocol 2 above with a share conversion protocol from a characteristic 2 field to a characteristic p field. However, to the best of our knowledge, the current existing efficient protocols for share-conversion between fields of different characteristics, e.g., those suggested in [14], are unsuitable here.

2.4 Protocol for Secure Computation

In this section we give the details of our secure multiparty computation protocol. The protocol is an extension of the semi-honest BMR protocol, e.g. [8], to the arithmetic case. The details of the garbled gates are explained in Sections 3, 4, and Appendix A. The proofs of correctness and security appear at Section 5.

The garbling phase of protocols following the multiparty garbling paradigm is often abstracted as a functionality that outputs the garbled circuit and the necessary permutation bits to the respective parties. This functionality, which we term F_{GC}, is described in Figure 1. We next sketch out a straightforward protocol for securely computing F_{GC} in constant rounds, using a combination of the BGW protocol [9, 19] and the constant round protocol for unbounded fan-in multiplication of Bar-Ilan and Beaver [3].

Step 1, Setup: For each prime p in the primorial modulus, each party P_i does the following:
- For each wire $\omega \in \mathcal{W}_p$ (i.e., input wires of the circuit and output wires of garbled gates/components), randomly chooses a random element $(\lambda_\omega)_i \in \mathbb{F}_p$ and (zero) key $k^i_\omega \in \mathbb{F}_{p^\kappa}$.[6] The random permutation element on the wire is $\lambda_\omega \stackrel{\text{def}}{=} \Sigma_{i=1}^n [(\lambda_\omega)_i]$.
- In topological order on the circuit, computes $(\lambda_\omega)_i$ and k^i_ω for each wire $\omega \notin \mathcal{W}_p$, by summing/multiplying by a constant (according to gate type), by using λ_i and k^i on the input wires – see Section 2.2 on "free" gates.
- Each party randomly chooses a random global offset $\Delta^i_p \in \mathbb{F}_{p^\kappa}$.
- For each garbled component $g \in C$, compute $\mathcal{F}_{k^i_{x,\alpha}}(g, j)$ for each $j \in [n]$ and $\alpha \in \mathbb{F}_p$, where p is according to the gate/component type.

Step 2, Sharing: Each party P_i shares all the keys, elements, and outputs of $\mathcal{F}$ in Step 1 using $(t+1)$-out-of-n Shamir secret-sharing scheme. Multifield-shared bits are also shared using Protocol 2 in Section 2.3 for each p. The parties obtain shares of λ_ω for each wire by locally summing their shares of $\{(\lambda_\omega)_i\}_{i=1}^n$.

Step 3, Computing the garbled gates: Shares of the garbled rows of each garbled gate/component are computed using their respective equation (e.g., Equations 6,8,12,13), where in each equation
- Addition and multiplication by a constant are computed locally,
- Multiplication is computed using a BGW degree-reduction round,
- Exponentiation (Appendix A) is computed using the protocol of [3].

More details can be found in the respective section.

Step 4, Reconstructing the outputs: The parties exchange the shares (of the outputs of F_{GC}) and reconstruct the outputs of F_{GC}, namely the garbled gates/components and the output permutation elements. Furthermore, each party receives the shares and reconstructs the permutation elements on its input wires.

Remark 4. The above protocol is constant round since all gates are computed in parallel and each step is constant round (Step 1 is local). However, the protocol can be considerably optimized using techniques described in [7], such as share-conversion and masking by additive shares of zeros. Due to space limitations, we give the optimized protocol in Appendix B. An alternative protocol for arithmetic garbled circuits that does not require an honest majority is given in Appendix F.

Next, in Figure 2 we give the details of our MPC protocol, in the F_{GC}-hybrid model (i.e., F_{GC} can be excuted securely as a black-box). The protocol is similar to other protocols following the multiparty garbling paradigm, e.g., [8]. The only major difference is the external values are not exclusively Boolean, and the size of the garbled gates/components varies according to the gate type. The evaluation of the various gates (Step 3b in Figure 2) is explained in the respective section. Correctness and security of the protocol are shown in Section 5.

[6] In the designated selector gates, this choice is slightly more involved – P_i randomly chooses $(\widehat{\lambda_\omega})_i, (\widehat{\widehat{\lambda_\omega}})_i, (\widetilde{\lambda_\omega})_i, (\widetilde{\widetilde{\lambda_\omega}})_i$ such that $(\widehat{\lambda_\omega})_i + (\widetilde{\lambda_\omega})_i = (\widehat{\widehat{\lambda_\omega}})_i + (\widetilde{\widetilde{\lambda_\omega}})_i$. The keys are similarly partitioned; see Section 4.

Functionality F_{GC}

Computation Course:

1. For every prime p in the mixed modulos, the functionality assigns a random global offset $\Delta_p^i \in \mathbb{F}_{p^\kappa}$ to each party P_i.
2. For each wire $\omega \in \mathcal{W}_p$, the functionality assigns
   - A random permutation element $\lambda_\omega \in \mathbb{F}_p$.
   - For each party P_i, a random zero key $k_\omega^i \in \mathbb{F}_{p^\kappa}$. The keys associated with the p external values are set to be $k_{\omega,\alpha}^i = k_\omega^i + \alpha k_\omega^i$ for $\alpha \in \mathbb{F}_p$.
3. For each addition gate or multiplication by a constant $c \neq 0$ gate, with output wire $z \in W_p$ and input wires x, y (or just x), the functionality computes
   - The permutation element of the output wire $\lambda_z = \lambda_x + \lambda_y$ or $\lambda_z = c\lambda_x$ respectively.
   - The zero keys of the output wire, $k_z^i = k_x^i + k_y^i$ or $k_z^i = ck_x^i$ respectively for each party P_i.

   The computations are in the fields $\mathbb{F}_p$ and $\mathbb{F}_{p^\kappa}$ respectively.
4. The functionality computes the garbled circuit GC, i.e., computes the garbled rows for each garbled gate/component. For non-Boolean gates, this is computed according to the gate equations in Sections 3, 4, and Appendix A.

Outputs:

1. The functionality outputs the garbled gates to the evaluating parties.
2. For output wires of the circuit, the functionality outputs the permutation bits to the evaluating parties.
3. The functionality outputs to each party its global offsets, its zero key for each wire $w \in \mathcal{W}$, and the permutation bit of each of its input wires.

Fig. 1: Functionality F_{GC} for Constructing a Multiparty Garbled Circuit

3 Multiparty Multiplication Gates

In this section we show how to extend the notion of half-gates, introduced by Zahur et al. [34], to multiplication gates in the multiparty setting. The multiplication gates are in the finite field $\mathbb{F}_p$ (note that regular half-gates, i.e. AND gates, are multiplication in $\mathbb{F}_2$). The total cost of a multiplication gate in $\mathbb{F}_p$ will be $2p$ garbled rows, in comparison with p^2 garbled rows of the naïve construction. In particular, the Boolean AND gate will cost $4 = 2 * 2 = 2^2$ garbled rows using both the half-gates and the regular construction.

Remark 5. In the two party case, row reduction allows to reduce 2 garbled rows using half-gates, while other methods either allow only a single row reduction or are not compatible with the free-XOR technique. This is the main reason to use half-gates also in the two-party Boolean case. However, no *efficient* row reduction technique is yet known for the general multiparty case. Therefore, half-gates does not seem to be suitable for the multiparty Boolean case.[7]

Remark 6. In [2], multiplication gates in $\mathbb{F}_p$ are constructed differently, mainly using projection gates. As explained, projection gates seem to be considerably more expensive in the multiparty case than in the two-party case. Therefore, multiplication using an extension of the half-gates idea, as explained here, should be preferred in the multiparty setting. In fact, the garbled multiplication gate of [2] require slightly more rows and more decryptions, so possibly using the half-gates extension should be considered also for the two-party setting.

We follow the convention of [34], describing the two half gates as the "Garbler Half Gate" and "Evaluator Half Gate", though in our scenario all parties perform the garbling collaboratively and each party can perform the evaluation.

[7] Using half gates requires double the amount of decryptions during evaluation and is therefore inferior in this case despite having the same number of garbled rows.

Protocol Π_{online}

1. **Offline phase:** The parties execute functionality F_{GC} to receive the garbled circuit, the output wires' permutation elements, and the input permutation elements on their respective input wires.
2. **Exchange garbled keys associated with inputs:** For every circuit-input wire w:
   (a) Let P_i be the party whose input is associated with wire w and let x_{i_w} be P_i's input value associated with the wire. Then, P_i sends $e_w = x_{i_w} + \lambda_w$ to all parties.
   (b) Each party P_j sends its part k^j_{w,e_w} of the garbled label on w to the evaluating parties.
   (c) At this point, the evaluating parties hold $k^1_{w,e_w}, \ldots, k^n_{w,e_w}$ for every circuit-input wire.
3. **Local circuit computation:** Each evaluating party locally evaluates the garbled circuit by traversing the circuit in a topological order, computing gate by gate. Let g be the current gate with output wire z and input wires x, y (or just x). Let e_x and e_y be the *extrenal values* on wires x and y, respectively.
   (a) If g is an addition or multiplication by $c \neq 0$ gate, then P_0 sets $e_z = e_x + e_y$ or $e_z = ce_x$ respectively. In addition, for every $j = 1, \ldots, n$, it computes $k^j_{z,e_z} = k^j_{in1,e_{in1}} + k^j_{in2,e_{in2}}$ or $k^j_{z,e_z} = c \cdot k^j_{in1,e_{in1}}$.
   (b) If g is a non-free gate, then the evaluating party recovers the output keys and external value by evaluating the gate. This is explained, according to the gate type, in Sections 3, 4, and Appendix A.
4. **Output determination:** For every output wire w, the evaluating party computes the *real* output value of wire w to be $e_w - \lambda_w$, where e_w is the external value on wire w and λ_w is as received from the output of F_{GC}.

Fig. 2: The online phase – circuit evaluation

Before going into the details of each half-gate, we give an informal overview of the idea. Assume we have a multiplication gate with input wires x, y and output wire z. During evaluation, the evaluating party learns on the input wires the external values bits $e_x = v_x + \lambda_x$ and $e_y = v_y + \lambda_y$, where v and λ are the real value and the permutation element on the wires respectively. The evaluating party also learns the keys corresponding to these external values. Using this, the evaluating party should be able to recover the output external value

$$e_z = v_z + \lambda_z = v_x v_y + \lambda_z \tag{4}$$

and corresponding keys.[8] In the naïve construction, this is done by decrypting the row (e_x, e_y), see Section 2.2.

In the half-gates construction, the computation is split into two distinct half-gates, each performing a different computation. Informally, the the first gate computes $-\lambda_y v_x$ and the second half-gate computes $v_x(v_y + \lambda_y)$. Then, adding the two outputs, which is free, results in $v_z = v_x v_y$.

To securely compute a multiplication gate using these two half gates in the *multiparty* setting, two adjustments have to be made. The necessity of these adjustments will become apparent when we discuss security. The first adjustment is that the permutation element on the output wire, λ_z, must be partitioned $\lambda_z = \widetilde{\lambda_z} + \widehat{\lambda_z}$, where $\widetilde{\lambda_z}, \widehat{\lambda_z} \in \mathbb{F}_p$ are random elements under the constraint that they sum to λ_z (which is the random permutation element of the output wire). This is because the outputs of both half gates must be hidden, otherwise information might be leaked on some of the values.[9]

The second adjustment is that the zero keys k^i_z on the output wire also need to be partitioned $k^i_z = \widehat{k_z}^i + \widetilde{k}^i_z$, where $\widehat{k_z}^i, \widetilde{k}^i_z \in \mathbb{F}_{p^\kappa}$ are random under the constraint that they sum to k^i_z. The main idea of this partition is that the output keys of an honest party P_i on both half gates do not leak information on the global offset Δ^i_p. The permutation elements and keys $\widetilde{\lambda_z}, \widetilde{k}^i_z$ are used in the Garbler half gate, and $\widehat{\lambda_z}, \widehat{k_z}^i$ are used in the Evaluator half gate.

[8] As explained in Section 2.2, it is enough to learn the keys; the evaluating party learns the external value by comparing with its local key used for the garbling.

[9] This is different than the two-party case, where the evaluator half gate can be handled differently, cf. [34].

To conclude, informally the half gates construction computes the output using the following equation:

$$e_z = v_x v_y + \lambda_z = \overbrace{\left(-\lambda_y v_x + \widetilde{\lambda_z}\right)}^{\text{“Garbler Half Gate”}} + \overbrace{\left(v_x(v_y + \lambda_y) + \widehat{\lambda_z}\right)}^{\text{“Evaluator Half-Gate”}}, \tag{5}$$

The true construction and resulting equations are more involved, and we next explain them in detail.

3.1 Garbler Half Gate

In the original description of this half gate in [34], the idea is described that the garbler can take advantage that it knows the permutation bit (or color bit in the terminology of [34]). In the multiparty case, no unauthorized subset (i.e., a subset that could be controlled by the adversary) is allowed to know the permutation element on any wire that it should not learn. However, we can use the fact that the permutation elements are secret-shared to do the necessary computations. The computed gate is slightly more complicated than in the two-party case because the garbling parties also participate in the evaluation, and thus have additional information.

As already stated, the garbler half gate should compute the value $-\lambda_y v_x + \widetilde{\lambda_z}$. Note that v_x is the real value on the wire x (in an ungarbled computation) and is therefore never known – neither during garbling nor during the evaluation phase. Thus, we cannot hope to use it directly.

To overcome this, the value is computed using the equation $-\lambda_y v_x + \widetilde{\lambda_z} = -\lambda_y(v_x + \lambda_x) + \lambda_y\lambda_x + \widetilde{\lambda_z}$. The value $v_x + \lambda_x$ is the external value on wire x and thus revealed during evaluation. For garbling, the rows are computed for all p values, using the BGW protocol with the shares of permutation bits, and with $v_x + \lambda_x$ treated as a constant (as α in the αth row). The final garbled garbler half gate is the set of encryptions

$$\tilde{g}^i_\alpha = Enc_{k_{x,\alpha}}\left[\ \widetilde{k}^i_z + \left(-\alpha\lambda_y + \lambda_y\lambda_x + \widetilde{\lambda_z}\right)\Delta^i_p\ \right] \tag{6}$$

for every $\alpha \in \mathbb{F}_p$ and $i \in [n]$. Note that α is a constant and all other values are secret-shared. Since the multiplicative depth of this equation is 2, computing this half gate (Equation (6)) requires two BGW degree-reduction rounds.

To verify that the correct output key is recovered, we observe that if the input external value is e_x, then the encryptions $\tilde{g}^i_{e_x}$ are decrypted for all i. Thus, the recovered output keys are

$$\begin{aligned} Dec_{k_{x,e_x}}[\tilde{g}^i_{e_x}] &= \widetilde{k}^i_z + \left(-e_x\lambda_y + \lambda_y\lambda_x + \widetilde{\lambda_z}\right)\Delta^i_p \\ &= \widetilde{k}^i_z + \left(-(v_x + \lambda_x)\lambda_y + \lambda_y\lambda_x + \widetilde{\lambda_z}\right)\Delta^i_p \\ &= \widetilde{k}^i_z + \left(-\lambda_y v_x + \widetilde{\lambda_z}\right)\Delta^i_p = \widetilde{k}^i_{z,-\lambda_y v_x + \widetilde{\lambda_z}} \end{aligned}$$

matching the expected value of the keys corresponding to $-\lambda_y v_x + \widetilde{\lambda_z}$.

3.2 Evaluator Half Gate

As in the two-party case, the main idea of this half gate is that the evaluating party learns at the evaluation phase the external values of the wires, and can use this information for the computation. As we shall see more clearly when we extend the half gates to $\mathbb{F}_p$, the operation done by the evaluating party is to multiply by this external value.

The evaluator half gate should compute the value $v_x(v_y + \lambda_y) + \widehat{\lambda_z}$. The value $v_y + \lambda_y$ is the external value e_y on input wire y, and therefore known at evaluation time. On the other hand, the value v_x is the true value on wire x, and thus generally should never be learnt by any subset of parties. Therefore, to compute the gate we use the equation:

$$v_x(v_y + \lambda_y) + \widehat{\lambda_z} = (v_x + \lambda_x)(v_y + \lambda_y) + \lambda_x(v_y + \lambda_y) + \widehat{\lambda_z}. \tag{7}$$

The computation of the value $\lambda_x(v_y + \lambda_y) + \widehat{\lambda_z}$ is similar to the computations in the Garbler Half Gate. Thus, the main addition in this half gate is the computation of the value $(v_x + \lambda_x)(v_y + \lambda_y)$. Naïvely, it would seem that this requires p^2 rows in order to garble for each combination of $(v_x + \lambda_x, v_y + \lambda_y) \in (\mathbb{F}_p)^2$. However, in the two party Boolean case, [34] observed that this computation can be obtained practically for free. We first explain the observation of [34], and then extend it to the multiparty $\mathbb{F}_p$ case.

In the Boolean case, the external values are $(v_x \oplus \lambda_x)$ and $(v_y \oplus \lambda_y)$. Note first that $(v_x \oplus \lambda_x)(v_y \oplus \lambda_y)$ can be computed at evaluation time as both external values are known. This is still insufficient, because the evaluating party needs to recover some key that corresponds to this value. The "trick" performed by [34] is to XOR with the key on the wire x if $v_y \oplus \lambda_y = 1$ and to ignore it if $v_y \oplus \lambda_y = 0$. We next describe this slightly differently for the $\mathbb{F}_p$ case, but the descriptions in fact coincide for $p = 2$.

To extend the technique of [34], during evaluation, each evaluating party multiplies the key on wire x by the external value $v_y + \lambda_y$ and adds it to the decrypted key. Notice that this completely coincides with the Boolean case when $p = 2$ (since multiplying the key by 0 is the same as ignoring the key). The only subtlety is that now the corresponding multiplication of the zero key must be subtracted from the encrypted key during the garbling. However, the proof for this extended technique is slightly trickier, as we shall see in Section 5.

The garbled evaluator half gate is the set of encryptions

$$\widehat{g}^j_\beta = Enc_{k_{y,\beta}}\left[\ \widehat{k}^i_z - \beta k^i_x + \left(-\beta\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p\ \right] \tag{8}$$

for every $\beta \in \mathbb{F}_p$ and $i \in [n]$. Since the multiplicative depth is 1, computing this half gate (Equation 8) requires one BGW degree-reduction round.

Now during evaluation, the evaluating party multiplies the key on the x wire by the external value $e_y = v_y + \lambda_y$. This is then added to the key decrypted at row e_y. We next verify that the recovered output keys indeed corresponds to the correct value: the recovered output keys are the sum (for each $i \in [n]$) of $e_y \cdot k^i_{x,e_x}$ with the decryption of $\widehat{g}^i_{e_y}$. Simplifying,

$$\begin{aligned} e_y \cdot k^i_{x,e_x} + Dec_{k_{y,e_y}}[\widehat{g}^i_{e_y}] &= e_y(k^i_x + e_x\Delta^i_p) + \widehat{k}^i_z - e_y k^i_x + \left(-e_y\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p \\ &= e_y k^i_x + e_y(v_x + \lambda_x)\Delta^i_p + \widehat{k}^i_z - e_y k^i_x + \left(-e_y\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p \\ &= \widehat{k}^i_z + \left(e_y v_x + \widehat{\lambda_z}\Delta^i_p\right)\Delta^i_p = \widehat{k}^i_{z,e_y v_x + \lambda_z} \end{aligned}$$

matching the expected key value of $v_x\,(v_y + \lambda_y) + \widehat{\lambda_z}$.

4 Selector Gates

One of the more challenging tasks of performing an arbitrary computation using arithmetic circuits is to perform conditional statements. In this section, we discuss a gate computing a simple if statement. Namely, we build a "selector" gate, which chooses between two input wires in $\mathbb{F}_p$, according to a Boolean "selection bit". I.e., the gate has three input wires x, y, and w_0, and an output wire z. The values on the input wires x, y are from $\mathbb{F}_p$ and the value on w_0 is the selection bit. The selector gate computes the following if statement: If $(v_{w_0} == 0)$ then $v_z = v_x$ else $v_z = v_y$. Note that by applying this to each wire in the CRT representation, we get a selector gate for integers.

We show two constructions for a selector gate. The first construction is using known techniques. The gate is constructed by first projecting the value of w_0 into $\mathbb{F}_p$ using a projection gate, and then using a multiplication gate. That is, the gate is computed using the equation:

$$\begin{aligned} v_z = \varphi(v_{w_0}) \cdot v_x + \varphi(v_{w_0} \oplus 1) \cdot v_y &= \varphi(v_{w_0}) \cdot v_x + (1 - \varphi(v_{w_0})) \cdot v_y \\ &= \varphi(v_{w_0}) \cdot (v_x - v_y) + v_y \end{aligned} \tag{9}$$

where φ denotes the projection of the bit into $\mathbb{F}_p$. There is one projection and one $\mathbb{F}_p$ multiplication in Equation 9, costing 2 and $2p$ garbled rows respectively. Thus, a selector gate using the above construction has $2p+2$ garbled rows. However, note that the evaluator has to decrypt 3 rows using this method: 1 for the projection gate, and 2 for the multiplication gate (1 in each half gate). To the best of our knowledge, this is the best selector gate construction using existing techniques and relying only on the existence of MMCCR PRFs; see Remark 7.

Our second construction will be a new and designated construction of a garbled selector gate. The cost of the designated garbled selector gate will be also $2p + 2$ garbled rows. However, the number of rows the evaluator will have to decrypt will be only 2. Thus, we expect evaluation of this designated selector gate to be approx. 33% faster.

Remark 7. If w_0 is in $\mathbb{F}_p$ then a selector gate can be garbled with $2p$ rows and only 2 decryptions at evaluation (since projection is not needed). However, we argue that it is important to consider the case of Boolean w_0 for two reasons: the first is that when computing over the integers using CRT, we would like the same bit to select in all the characteristics. The second is that w_0 could be determined by a complex set of conditions, so it would make sense that w_0 is the output or intermediate value of a Boolean sub-circuit.

If we do not restrict the security assumption to only MMCCR hash functions, then in the 2-party setting, Ball et al. [2] showed a direct construction of a selector gate that has $2p - 1$ rows and requires only 1 decryption, which can be proved secure based on a random oracle (or possibly also on some extension of definition 1 to allow correlation and circularity on two input keys). Note that in the 2-party setting, also the new designated construction and the construction using a projection gate have $2p - 1$ rows (this is because row-reduction [29] reduces 1 row from every garbled *component*). However, they require 2 and 3 decryptions, respectively. Nevertheless, since in the two-party setting this is the only garbled gate which an optimization is known using a stronger assumption, we feel it is important also to optimize constructions that are based only on MMCCR.

4.1 Charateristic 2 to Characteristic p Projection Gates

In this section we explain how to construct a projection gate that maps a bit value on a Boolean wire to the same value on a wire in $\mathbb{F}_p$. The projection gate has a single input wire w_0 containing a Boolean value, and an output wire z, containing the same value in $\mathbb{F}_p$. I.e., if $v_{w_0} = 0$ then $v_z = 0$ and if $v_{w_0} = 1$ then $v_z = 1$ (note that $v_{w_0} \in \mathbb{F}_2$ and $v_z \in \mathbb{F}_p$). This projection gate is needed if one wishes to multiply the bit value by a value in $\mathbb{F}_p$, as in the first selector gate construction described above.

The projection gate takes advantage of the following observation: suppose that $v_{w_0}, \lambda_{w_0} \in \{0, 1\}$. Then,

$$v_{w_0} \oplus \lambda_{w_0} = \begin{cases} v_{w_0} - \lambda_{w_0}, & v_{w_0} \oplus \lambda_{w_0} = 0 \\ v_{w_0} + \lambda_{w_0}, & v_{w_0} \oplus \lambda_{w_0} = 1, \end{cases} \tag{10}$$

where the computations on the left and right are in $\mathbb{F}_2$, the computation in middle is in $\mathbb{F}_p$, and equality signifies that the value is the same value in $\{0, 1\}$ (whether in $\mathbb{F}_2$ or $\mathbb{F}_p$). I.e., if $v_{w_0} = \lambda_{w_0}$ then $v_{w_0} - \lambda_{w_0} = 0 = v_{w_0} \oplus \lambda_{w_0}$ and if $v_{w_0} \neq \lambda_{w_0}$ then $v_{w_0} + \lambda_{w_0} = 1 = v_{w_0} \oplus \lambda_{w_0}$.

To use Equation (10), we will assume that λ_{w_0} is a multifield-shared bit, shared in both a field of characteristic 2 and a field of characteristic p. Note that although the output value is known to be a bit, it is masked using a random permutation element in $\mathbb{F}_p$ to avoid leaking information. Thus, the equation of the gate will be

$$e_z = v_z + \lambda_z = v_{w_0} + \lambda_z = \begin{cases} (v_{w_0} \oplus \lambda_{w_0}) + \lambda_{w_0} + \lambda_z, & v_{w_0} \oplus \lambda_{w_0} = 0 \\ (v_{w_0} \oplus \lambda_{w_0}) - \lambda_{w_0} + \lambda_z, & v_{w_0} \oplus \lambda_{w_0} = 1. \end{cases} \tag{11}$$

Hence, the garbled projection gate is the following encryptions for every $i \in [n]$:

$$Enc_{k_{w_0,0}} \left[\widehat{k}_z^i + (\lambda_{w_0} + \lambda_z)\, \Delta_p^i \right], \tag{12}$$

$$Enc_{k_{w_0,1}} \left[\widehat{k}_z^i + (1 - \lambda_{w_0} + \lambda_z)\, \Delta_p^i \right]. \tag{13}$$

As explained in Section 2, although $k_{w_0,0}, k_{w_0,1} \in \mathbb{F}_{2^\kappa}$, the output of the PRF in this case is in $\mathbb{F}_{p^\kappa}$. Assuming we have λ_{w_0} as a multifield-shared bit, i.e., the parties already posses Shamir shares of the the bit λ_{w_0} in the correct field of characteristic p, Equations (12),(13) can be computed using one additional BGW degree-reduction round. Using Equation (11), it is not difficult to verify that for both values of e_{w_0} the decrypted key corresponds to e_z.

4.2 Designated Selector Gate Construction

In this section we explain a designated construction for a selector gate. The gate contains three components. The first component, which we call the chooser partial gate, has 2 garbled rows. The other two components, which we call the corrector partial gates, contain p garbled rows each. Thus, this construction of a selector gate will require $2p+2$ garbled rows, same as the previous construction. However, this construction requires less decryptions at the evaluation phase, as we explain next.

The main idea we use in our construction can be seen as an extension of the half-gate technique – the evaluating party uses the key of one of the input wires, according to the external value on the selection wire. Furthermore, the evaluating party decodes only one of the two corrector partial gates according to the external value on the selection wire. Therefore, only two rows are decrypted when evaluating the gate (one in the chooser gate and one in the corrector gate), 1 less than the previous construction.

Note that since the external values are known only at the evaluation phase, we cannot prevent a corrupt evaluating party from decrypting also the other corrector partial gate. Thus, we must ensure that the decrypted key from this does not leak any extra information. This is achieved using a double partitioning of the output zero keys and permutation bit. I.e.,

$$\lambda_z = \widehat{\lambda_z} + \widetilde{\lambda_z} = \widehat{\widehat{\lambda_z}} + \widetilde{\widetilde{\lambda_z}}, \tag{14}$$

$$k_z^i = \widehat{k}_z^i + \widetilde{k}_z^i = \widehat{\widehat{k}}_z^i + \widetilde{\widetilde{k}}_z^i, \tag{15}$$

where $\widehat{\lambda_z}, \widetilde{\lambda_z}, \widehat{\widehat{\lambda_z}}, \widetilde{\widetilde{\lambda_z}} \in \mathbb{F}_p$ are random such that they satisfy Equation (14) and likewise $\widehat{k}_z^i, \widetilde{k}_z^i, \widehat{\widehat{k}}_z^i, \widetilde{\widetilde{k}}_z^i \in \mathbb{F}_{p^\kappa}$ are random such that they satisfy Equation (15). Note for example that $\widetilde{\widetilde{\lambda_z}}$ is random even given $\widehat{\lambda_z}, \widetilde{\lambda_z}$. Such observations are crucial for security, as we later explain. Otherwise, a corrupt evaluator could learn secret information by decrypting the "wrong" corrector gate. This idea of double partition of the keys and permutation elements appears to have not been used before in garbled circuits.

4.2.1 Half-Selector Gate.

We now show the construction of a half selector gate that receives only two input wires, x and w_0, and outputs either x or 0 according to w_0. This easily extends to a full selector gate, using the equation

$$v_z = v_{w_0} \cdot v_x + (v_{w_0} \oplus 1) \cdot v_y = v_{w_0} \cdot (v_x - v_y) + v_y. \tag{16}$$

I.e., computing the value of $x - y$ using free subtraction, then using a half-selector, and then freely adding the value of y. It is also possible to construct a full selector gate directly. This is explained in Appendix D. The construction of the half-selector gate is significantly simpler, but contains most of the main ideas.

Informally, the half-selector gate is computed using the following equation:

$$v_x v_{w_0} + \lambda_z = \begin{cases} \overbrace{v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\lambda_z}}^{\text{"Chooser Gate"}} \quad \overbrace{+\lambda_{w_0} v_x + \widetilde{\lambda_z}}^{\text{"Corrector Gate"}} & v_{w_0} \oplus \lambda_{w_0} = 0 \\ v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\widehat{\lambda_z}} \quad -\lambda_{w_0} v_x + \widetilde{\widetilde{\lambda_z}} & v_{w_0} \oplus \lambda_{w_0} = 1. \end{cases} \tag{17}$$

This equation works because $v_x(v_{w_0} \oplus \lambda_{w_0}) = \begin{cases} v_x v_{w_0} - \lambda_{w_0} v_x & v_{w_0} \oplus \lambda_{w_0} = 0 \\ v_x v_{w_0} + \lambda_{w_0} v_x & v_{w_0} \oplus \lambda_{w_0} = 1, \end{cases}$ as one can readily verify for the 4 combinations of $v_{w_0}, \lambda_{w_0} \in \{0,1\}$. Note also that the equations of the chooser gate in the first and

second row simplify to $\widehat{\lambda_z}$ and $v_x + \widehat{\widehat{\lambda_z}}$ respectively, since the value of $v_{w_0} \oplus \lambda_{w_0}$ is already fixed. The reason why we need to use different partitions of λ_z in the two rows will become clear when we discuss the corrector partial gates in detail. In short, the reason is to ensure that decrypting the "wrong" corrector gate does not leak any information.

Chooser Partial Gate for Half Selector. The chooser partial gate is somewhat similar to the evaluator half gate. The first garbled row, which is decrypted when $v_{w_0} \oplus \lambda_{w_0} = 0$, should output a key corresponding to $v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\lambda_z} = \widehat{\lambda_z}$ if decrypted. The $\widehat{\lambda_z}$ is secret-shared, so this computation is done in a straightforward manner.

The second garbled row is decrypted when $v_{w_0} \oplus \lambda_{w_0} = 1$, and the output keys should correspond to the value $v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\widehat{\lambda_z}} = v_x + \widehat{\widehat{\lambda_z}}$. Here we use a similar trick as in the evaluator half-gate, i.e., the equation $v_x + \widehat{\widehat{\lambda_z}} = (v_x + \lambda_x) - \lambda_x + \widehat{\widehat{\lambda_z}}$, where for the value $v_x + \lambda_x$ the evaluator will add the key on the input wire x, as in the evaluator half gate. To conclude, the chooser partial gate for a half selector gate has the following encryptions for every $i \in [n]$:

$$Enc_{k_{w_0,0}}\left[\ \widehat{k}_z^i + \widehat{\lambda_z}\Delta_p^i\ \right], \tag{18}$$

$$Enc_{k_{w_0,1}}\left[\ \widehat{\widehat{k}}_z^i - k_x^i + \left(-\lambda_x + \widehat{\widehat{\lambda_z}}\right)\Delta_p^i\ \right]. \tag{19}$$

At the garbling phase, these equations require one BGW degree-reduction round. At the evaluation phase, if the external value e_{w_0} is 1, the evaluating party also adds the key on wire x after decryption.

We verify that the decrypted keys indeed correspond to the values $v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\lambda_z}$ and $v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\widehat{\lambda_z}}$:

1. If $e_{w_0} = 0$ the decrypted keys are $\widehat{k}_z^i + \widehat{\lambda_z}\Delta_p^i = \widehat{k}^i_{z,\widehat{\lambda_z}} = \widehat{k}^i_{z,v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\lambda_z}}$,
2. If $e_{w_0} = 1$ the output is the sum of $k_x^i + e_x\Delta_p^i$ and $\widehat{\widehat{k}}_z^i - k_x^i + \left(-\lambda_x + \widehat{\widehat{\lambda_z}}\right)\Delta_p^i$. Simplifying:

$$\begin{aligned}\left[k_x^i + e_x\Delta_p^i\right] + \left[\widehat{\widehat{k}}_z^i - k_x^i + \left(-\lambda_x + \widehat{\widehat{\lambda_z}}\right)\Delta_p^i\right] &= \widehat{\widehat{k}}_z^i + \left(v_x + \widehat{\widehat{\lambda_z}}\right)\Delta_p^i \\ &= \widehat{\widehat{k}}^i_{z,v_x + \widehat{\widehat{\lambda_z}}} = \widehat{\widehat{k}}^i_{z,v_x(v_{w_0} \oplus \lambda_{w_0}) + \widehat{\widehat{\lambda_z}}}.\end{aligned}$$

Corrector Partial Gate for Half Selector. The computation of each corrector partial gate is similar to the garbler half gate. The interesting point is that there are two corrector gates for every selector gate, and only one value is used at evaluation. However, since which of the two is used is known only at the evaluation phase, both corrector gates need to be computed at the garbling phase.

The garbled rows of the first corrector gate, which correspond to the value $\lambda_{w_0} v_x + \widetilde{\lambda_z} = \lambda_{w_0}(v_x + \lambda_x) - \lambda_{w_0}\lambda_x + \widetilde{\lambda_z}$, are the following encryptions for each $\alpha \in \mathbb{F}_p$ and $i \in [n]$:

$$Enc_{k_{x,\alpha}}\left[\ \widetilde{k}_z^i + \left(\alpha\lambda_{w_0} - \lambda_{w_0}\lambda_x + \widetilde{\lambda_z}\right)\Delta_p^i\ \right]. \tag{20}$$

The garbled rows of the second corrector gate, which correspond to the value $-\lambda_{w_0} v_x + \widetilde{\widetilde{\lambda_z}} = -\lambda_{w_0}(v_x + \lambda_x) + \lambda_{w_0}\lambda_x + \widetilde{\widetilde{\lambda_z}}$, are the following encryptions for each $\alpha \in \mathbb{F}_p$ and $i \in [n]$:

$$Enc_{k_{x,\alpha}}\left[\ \widetilde{\widetilde{k}}_z^i + \left(-\alpha\lambda_{w_0} + \lambda_{w_0}\lambda_x + \widetilde{\widetilde{\lambda_z}}\right)\Delta_p^i\ \right]. \tag{21}$$

Assuming λ_{w_0} is a multifield-shared bit, computing these gates requires two BGW degree-reduction rounds. Verification is slightly tedious and hence omitted.

Combining the above components results in the half-selector gate: At the evaluation phase, an honest evaluating party decrypts the chooser partial gate and only one of the corrector gates, according to the external value on the selector wire w_0. By summing the values, the evaluating party recovers the key corresponding to $v_x v_{w_0} + \lambda_z$.

Observe that the same key is used to decrypt both corrector gates. Thus, a corrupt evaluating party can recover the decrypted keys on both corrector gates, regardless of the external value on wire w_0. Therefore, we must ensure that the unused decrypted value does not leak any information. We explain the intuition for the case $e_{w_0} = 0$; the case of $e_{w_0} = 1$ is similar. Notice that the keys decrypted from the inactive corrector gate are $\widetilde{\widetilde{k}}_z^i + \left(-e_x\lambda_{w_0} + \lambda_{w_0}\lambda_x + \widetilde{\widetilde{\lambda_z}}\right)\Delta_p^i$ for $i \in [n]$. There are 2 key observations:

- Clearly, a corrupt evaluating party P_i can learn the value $-e_x\lambda_{w_0} + \lambda_{w_0}\lambda_x + \widetilde{\widetilde{\lambda_z}}$ by subtracting $\widetilde{\widetilde{k}}_z^i$ and dividing by Δ_p^i. Furthermore, e_x, e_{w_0}, $\widehat{\lambda_z}$, and $\lambda_{w_0}v_x + \widetilde{\lambda_z}$ are known to the evaluator from the protocol.[10] Nevertheless, $\widetilde{\widetilde{\lambda_z}} \in \mathbb{F}_p$ is random even given these values. Thus, the value $-e_x\lambda_{w_0} + \lambda_{w_0}\lambda_x + \widetilde{\widetilde{\lambda_z}}$ leaks no information on λ_{w_0} and λ_x.
- A corrupt evaluating party learns $\widetilde{\widetilde{k}}_z^j + \left(-e_x\lambda_{w_0} + \lambda_{w_0}\lambda_x + \widetilde{\widetilde{\lambda_z}}\right)\Delta_p^j$ also for every honest party P_j. However, $\widetilde{\widetilde{k}}_z^j \in \mathbb{F}_{p^\kappa}$ is random even given the keys party P_i recovers from following the protocol. Thus, this does not leak any information on Δ_p^j.

The proof of security in Section 5 formalizes the above intuition.

5 Correctness and Security

In this section we state the correctness and security of our protocol. Due to lack of space, we only give sketches of the proofs. Some more details are given in Appendix G. The proof of Claim 1 is given in Appendix E.

Correctness. We briefly explain the correctness of the protocol. To show that the outputs received by the parties in Π_{online} (Figure 2) corresponds to the correct output, we show the following statement: for each wire, the evaluating parties recover at evaluation the correct external value $e_z = v_z + \lambda_z$, and the corresponding keys. For input wires, this statement follows from Step 2. The statement is then proved by induction on the topological ordering of the gates. For output wires of each gate type, this is shown in the respective section. Using the induction argument, the statement holds also for the output wires of the circuit. Thus, in Step 4, the value recovered by the parties at wire z is $e_z - \lambda_z = v_z$.

Security. We now show the security of our protocol. We assume a semi-honest adversary corrupting a strict minority of the parties. We begin with the following lemma:

Lemma 1. *Protocol Π_{GC} securely computes F_{GC} in the presence of a static semi-honest adversary controlling a strict minority of the parties.*

Proof Sketch. Protocol Π_{GC} computes F_{GC} using only Shamir secret sharing, the BGW protocol, and the constant round protocol for unbounded fan-in multiplication of [3]. These are secure and composable with each other (the protocol of [3] can be based on BGW) in the semi-honest model with an honest majority. The intermediate messages the adversary sees throughout the protocol (Steps 2 and 3) are only Shamir shares, which appear random in the information theoretic sense. Thus, they are easily simulated. The messages of the last round (Step 4) are computed by the simulator using the output (given from the trusted party) and the messages already given to the adversary in previous rounds. □

Before stating our main security theorem, we state the following claim that follows from Definition 1:

[10] Usually, the permutation bits must remain secret as they hide the value on the wire. However, in this specific case, the value on the wire corresponding to $v_x e_{w_0} = 0$ is publicly known. Thus, there is no need to hide $\widehat{\lambda_z}$.

Claim 1 *Let $B \subset [n]$. If H is mixed-modulus circular correlation robust, then for all polynomial time adversaries making only legal queries to the oracle, the oracle*

$$\mathcal{O}_P^{H,B}(\rho, a, b, \left(k^i\right)_{i\in B}, \gamma, (\delta_i)_{i\in B}) \stackrel{def}{=} \Sigma_{i\in B}\left[\mathcal{O}_P^{H,i}(\rho, a, b, k^i, \gamma, \delta_i)\right], \tag{22}$$

where each $\mathcal{O}_P^{H,i}$ is equal to $\mathcal{O}_P^H$ with random and independent Δ_p^is for each $p \in P$ and $i \in B$, is indistinguishable from a random function (with the same input/output domains).

Claim 1 is proved from Definition 1 by a reduction. The proof is given in Appendix E. Informally, the importance of Claim 1 is to use the claim with B as the set of honest parties, so $\mathcal{O}_P^{H,B}$ mimics "encryption" by all the honest parties. Further, the oracle adds offsets, corresponding to $\delta_i\Delta_p^i$s, to the encrypted keys of the honest parties.

To give some intuition, this will allow the distinguisher to change the values to which the encrypted keys correspond to, without knowing the Δ_p^is. For example, let e_x and e_z be the external values on the input and output wires of the gate. If the distinguisher will want to encrypt row $e_x + 1$ with the key corresponding to $e_z + 2$, then for the jth part it will use $\gamma = 1$ and $\delta_i = \begin{cases} 2 & i = j \\ 0 & i \neq j \end{cases}$ for each $i \in B$ (the computation of δ_j for the evaluator half gate and designated selector gates is slightly more complex, as explained in the proof). This way, the distinguisher only uses the keys k^i_{x,e_x}, k^i_{z,e_z} of the honest parties. Next, we state our main security theorem:

Theorem 2. *If H is a mixed-modulus correlation robust hash function then Protocol $\mathbf{\Pi_{online}}$ in Figure 2 securely computes f_C in the F_{GC}-hybrid model, in the presence of a static semi-honest adversary.*

The proof follows the general ideas used in [12], with the extended assumption. The main difficulty of the proof, on which we focus, is to show how the simulator simulates the output of F_{GC}, and in particular a fake garbled circuit, such that no polynomial time distinguisher can distinguish this fake garbled circuit from a real garbled circuit. To show this, we describe a distinguisher that uses H and legal queries to an oracle $\mathcal{O} \in \left\{\mathcal{O}_P^{H,B}, Rand\right\}$ in order to construct a circuit that distributes either as a real garbled circuit or as a fake garbled circuit, according to the oracle. Thus, distinguishing between the two types of circuits breaks the mixed-modulus correlation robustness of H. See Appendix G for more details.

There are two main differences from similar proofs: the first appears in multiplication gates, and specifically in the evaluator half gate. The second appears in the designated selector gates. Therefore, we split the proof sketch into two parts. In the first, we give an overview of the general proof structure and ideas, i.e., the construction of the fake circuit by the simulator, and the construction of the circuit by the distinguisher (which distributes as a real or fake circuit according to the oracle). In the second part, we explain the difficulties and necessary changes for evaluator half gates, and give a more detailed explanation on the subtleties of selector gates.

Proof Sketch. **Simulator:** The simulator chooses a random path on the circuit, i.e., for each wire $\omega \in \mathcal{W}$ selects a random *external value*. For each wire $\omega \in \mathcal{W}$ and for each honest party, the simulator chooses random keys corresponding to these external values. Then, the simulator computes the external values and corresponding keys of free gates. Using these values, the simulator computes a single encrypted row for each non-free gate/component – this row corresponds to the external value on the input wire. The other rows are sent as completely random strings (or more precisely as a random vector in $\mathbb{F}_{p^\kappa}^n$ for the appropriate p). There are slight differences in the designated selector gates, and these are explained later in the proof.

Distinguisher: The distinguisher starts by following the simulator construction for computing the first encrypted row. The other rows are computed differently, by using the oracle. The key observation is that the distinguisher can compute the γ's it needs to supply the oracle in order to, in the case $\mathcal{O} = \mathcal{O}_P^{H,B}$, encrypt the rows correctly *and* can compute the δ_i's in order to, if necessary, change the keys of the honest parties that are encrypted in that case.[11]

[11] All the keys of the corrupt parties are known to the distinguisher. For the honest parties, the distinguisher knows the keys corresponding to the external values (chose them randomly), but does not know the Δ_p^i's. Therefore, in order to change which value the honest parties' encrypted keys correspond to, it must use the oracle $\mathcal{O}_P^{H,B}$.

Computing γ is simply by the difference in the rows – this part is unchanged in the different gate types. Note that this ensures that $\gamma \neq 0$ and that each $\gamma \in \mathbb{F}_{p_a}$ is used only once for each gate and party index. Thus, the distinguisher makes only legal queries to the oracle.

To compute the δ_i's, the distinguisher uses the inputs to compute the real values on the wires. Using the real and external values, the distinguisher extracts the permutation elements, which are used to compute δ_i for each row and each $i \in B$. In the computation of the δ_i's there are differences and subtleties from similar proofs in both the evaluator half-gate and the designated selector gate, and we address these next.

Evaluator Half Gates: The simulator computes the evaluator half gates exactly the same. I.e., the simulator chooses an "external value" $\widehat{e}_z$ and corresponding random key $\widehat{k}^i_{z,\widehat{e}_z}$. However, note that the "external value" of output wire of the evaluator half gate represents $\widehat{e}_z = v_x(v_y + \lambda_y) + \widehat{\lambda_z}$, but the key $\widehat{k}^i_{z,\widehat{e}_z}$ represents $\widehat{k}^i_z - e_y k^i_x + \left(-e_y\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p$. This is because the evaluator should add $e_y k^i_{x,e_x}$ after decrypting row e_y.

This poses an extra challenge to the distinguisher when trying to compute the other rows, because they require deducting different multiplications of k^i_x, but the distinguisher does not know k^i_x.[12] However, the distinguisher does know $k^i_{x,e_x} = k^i_x + e_x\Delta^i_p$. Therefore, to deduct βk^i_x, the distinguisher computes $\beta(k^i_x + e_x\Delta^i_p) = \beta k^i_x + \beta e_x \Delta^i_p$. Then, this is deducted, and the βe_x is aggregated to the computation of the δ_i of that row. Thus, the simulator calls the oracle with these aggregated δ_i's. A detailed explanation is given in Appendix G.

Designated Selector Gates: First note that in the designate selector gates the simulator chooses three random external values and corresponding keys, although one of the corrector gates should not be decrypted. Furthermore, the simulator knows which corrector gate should not be decrypted. Nevertheless, the simulator constructs this gate as usual (one row correctly encrypted, and the other rows are random).

As for the distinguisher, the construction of the two corrector gates is similar to regular gates. The distinguisher builds both corrector gates, despite knowing which one should be decrypted. For the unused row in the chooser gate, the distinguisher uses the technique described for the evaluator half gate.

Conclusion: The proof concludes with the following key observation: If $\mathcal{O} = \mathit{Rand}$ then the circuit created by the distinguisher distributes as a fake garbled circuit created by the simulator, while if $\mathcal{O} = \mathcal{O}^{H,B}_P$ the circuit distributes as a real garbled circuit, created by a real execution of the protocol. An example computation is given in Appendix G. Thus, distinguishing between the two cases breaks the mixed-modulus circular correlation robustness. □

At first sight, it might not be obvious where in the proof we required the double partition of the keys and permutation bits. However, a closer inspection shows that by the simulator and distinguisher choosing the external values and keys of the two corrector gates randomly and independently, this fact is implicitly used. Otherwise (without the double partition), in a real garbled circuit the two external values are dependent and similarly the two keys, and would not match the distinguisher's construction. Furthermore, in a real execution of the protocol, if the λ's are not double partitioned, by subtraction of the two external values, a corrupt evaluator learns $2\lambda_{w_0}v_x$ (here λ_{w_0} is treated as an $\mathbb{F}_p$ element), violating security. If the keys are not double partitioned, then a corrupt evaluator can subtract the decrypted keys of an honest party P_i and recover a multiplication of Δ^i_p. Thus, this double partition is crucial.

Acknowledgements. I would like to thank Amos Beimel, Eran Omri, and Yehuda Lindell for the many ideas and helpful discussions.

[12] The random Δ^i_ps of $i \in B$ are an internal part of $\mathcal{O}^{H,B}_P$. The Δ^i_ps of the adversary ($i \notin B$) are known to both the simulator and the distinguisher.

Bibliography

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz. How to garble arithmetic circuits. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 120–129. IEEE Computer Society, 2011.

[2] M. Ball, T. Malkin, and M. Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM CCS, pages 565–577, 2016.

[3] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC*, 1989.

[4] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC, pages 503–513, 1990.

[5] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 478–492, 2013.

[6] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266, 2008.

[7] A. Ben-Efraim and E. Omri. Concrete efficiency improvements for multiparty garbling with an honest majority. In *Proceedings of the 5th International Conference on Progress in Cryptology – LATINCRYPT*, 2017. to appear.

[8] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing semi-honest secure multiparty computation for the internet. In *23rd ACM Conference on Computer and Communications Security (ACM CCS) 2016*, 2016. To appear.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 1–10, 1988.

[10] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 192–206, 2008.

[11] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. *Network*, 1:101101, 2010.

[12] S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the "free-XOR" technique. In *Theory of Cryptography*, pages 39–53. Springer, 2012.

[13] R. Cramer and I. Damgård. Secure distributed linear algebra in a constant number of rounds. In *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference*, pages 119–136, 2001.

[14] I. Damgard and R. Thorbek. Efficient conversion of secret-shared values between different fields. Cryptology ePrint Archive, Report 2008/221, 2008.

[15] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography–PKC 2009*, pages 160–179. Springer, 2009.

[16] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[17] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 1–18, 2013.

[18] I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Proceedings, Part I*, pages 167–187, 2017.

[19] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 101–111. ACM, 1998.

[20] O. Goldreich. *Foundations of Cryptography, Voume II Basic Applications*. Cambridge University Press, 2004.

[21] S. Gueron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 567–578. ACM, 2015.

[22] C. Hazay, P. Scholl, and E. Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Proceedings of the 23rd International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT, pages 598–628, 2017.

[23] M. Keller, E. Orsini, and P. Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.

[24] M. Keller, E. Orsini, D. Rotaru, P. Scholl, E. Soria-Vazquez, and S. Vivek. Faster secure multi-party computation of AES and DES using lookup tables. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, 2017*, 2017.

[25] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[26] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, pages 285–300, 2012.

[27] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 319–338, 2015.

[28] T. Malkin, V. Pastero, and a. shelat. An algebraic approach to garbling. Unpublished manuscript.

[29] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139. ACM, 1999.

[30] J. B. Nielsen and C. Orlandi. Lego for two-party secure computation. In *Theory of Cryptography Conference*, pages 368–386. Springer, 2009.

[31] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT, pages 250–267, 2009.

[32] X. Wang, S. Ranellucci, and J. Katz. Global-scale secure multiparty computation. In *24rd ACM Conference on Computer and Communications Security*, ACM CCS, pages 39–56, 2017.

[33] A. C. Yao. Protocols for secure computations. In *Proc. of the 23th IEEE Symp. on Foundations of Computer Science*, pages 160–164, 1982.

[34] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 220–250, 2015.

Supplementary Material

A Multiparty Garbling of Equality Testing and Exponentiation by a Constant

In this section we explain how to garble in the multiparty setting two important types of projection gates. The first is equality testing gate, which checks if a value is equal to 0, or more generally if two values are equal. The inputs of this gate are in $\mathbb{F}_p$ and the output is Boolean. The second type of gate we discuss is exponentiation by a (public) constant. The two constructions are similar, with the exponentiation by a constant being slightly simpler. The offline computations required for garbling these gates are significantly more complex than those shown in Sections 3 and 4.

A.1 Equality Gates

In this section we explain how to garble an important type of projection gate, the equality gates. An equality gate checks if the value on the input wire $v \in \mathbb{F}_p$ is 0. The gate outputs 0 if $v = 0$ and 1 otherwise. This gate can also be used to check equality mod p of 2 values on wires x, y, by comparing $v_x - v_y$ to 0 (recall that subtraction is free in our construction).

Before proceeding to explain how to compute the gates, we remark that although these gates are computed in a constant number of rounds, the computations are significantly more expensive than for the gates described in previous sections. Furthermore, in the two-party case, Ball et al. [2] showed how to reduce by a factor of 2 the cost of equality gates for integers, i.e., for a bundle of wires in CRT representation, by clever use of projection gates. Unfortunately, as projection gates are relatively expensive in the multiparty case, this optimization seems incompatible in the multiparty case. Thus, checking equality between two values in CRT representation is best done using AND gates, where the number of AND gates necessary is linear in the number of primes in the CRT representation.

The main idea of our mod p equality gate is to use Fermat's little theorem, which states that for every non-zero $v \in \mathbb{F}_p$, $v^{p-1} = 1$. Clearly, if $v = 0$ then $v^{p-1} = 0$. This can be efficiently computed in constant rounds using the unbounded fan-in protocol of [3] or the protocol of Cramer and Damgård [13, Section 7.1]. Then, the result must also be masked using the output permutation bit. The output permutation bit is used also in computations in $\mathbb{F}_{p^\kappa}$, and therefore we require it to be a multifield-shared bit.

The resulting computation is the following:

$$e_z = \text{“}v_z \oplus \lambda_z\text{”} = \left(((v_x + \lambda_x) - \lambda_x)^{p-1} - \lambda_z)\right)^2 . \tag{23}$$

Since $(v_x + \lambda_x)$ is known only at the evaluation phase, the parties compute $\left((\alpha - \lambda_x)^{p-1} - \lambda_z)\right)^2$ for every $\alpha \in \mathbb{F}_p$ during the garbling phase. Thus, the projection gate contains the following set of encryptions

$$Enc_{k_{x,\alpha}}\left[\, k_z^i + \left((\alpha - \lambda_x)^{p-1} - \lambda_z\right)^2 \cdot \Delta_p^i \,\right] \tag{24}$$

for every $\alpha \in \mathbb{F}_p$ and $i \in [n]$. Shares of $(\alpha - \lambda_x)^{p-1}$ are computed using the protocol of [3] or the protocol of [13], and squaring and multiplication by Δ_p^i each require a BGW degree reduction round. Note that the decrypted key, while corresponding to the correct external value, is in $\mathbb{F}_{p^\kappa}$. Therefore, the above projection gate is currently incompatible with our previous gadgets, e.g., free-XOR. We explain how to solve this next.

Converting to characteristic 2. Notice that the key recovered from the above garbled gate is in characteristic p, while the value is Boolean. If the wire is used only as input to AND gates or selector gates this does not pose a problem, and the key can be used in the normal way.[13] However, because the key is not in characteristic 2, the wire cannot be used in a free-XOR gate. Luckily, if the output permutation bit is a multifield-shared bit then there is a simple and cheap way to fix this.

[13] But also in this case we need the output permutation bit to be a multifield-shared bit.

Each party P_i computes and broadcasts the following two encryptions:

$$\mathcal{F}_{k^i_{z_{old},\alpha}}(g,i) \oplus \left(k^i_{z_{new}} \oplus \alpha\Delta^i\right) \tag{25}$$

for $\alpha \in \{0,1\}$, where $k^i_{z_{old}} \in \mathbb{F}_{p^\kappa}$ is the former zero output key and $k^i_{z_{new}} \in \mathbb{F}_{2^\kappa}$ is the new zero output key. We make several important observations. These observations follow because the external value that is supposed to be encrypted for each row is known already at the garbling phase (0 maps to 0 and 1 to 1). Thus, it is secure to let each party encrypt only its own key.

1. The computation of this conversion gate is done locally by each party (assuming the multifield-shared bit was precomputed).
2. The number of decryptions necessary in order to recover the output key is only linear in the number of parties. In contrast, the number of decryptions necessary in a regular gate (AND/multiplication/projection) is quadratic in the number of parties, as observed for example by [8].
3. Following the above two observations, in this specific gate we can use the standard row reduction technique, thus reducing the number of garbled rows to 1.
4. The output keys of the gate represent the same external values. Hence, if the multifield-shared bit was correctly generated they also represent the same real value. Furthermore, the new output keys are in a characteristic 2 field. Thus, they can be used as input keys for a free-XOR gate.

A.2 Exponentiation Gates

In this section we briefly explain how to garble exponentiation gates in the multiparty setting. An exponentiation gate is a projection gate, i.e., has a single input wire x and output wire z, that computes $z = x^c$ for a public constant c. The garbling of an exponentiation gate is very similar to equality gates, and even slightly simpler. In particular, there is no need for a multifield-shared bit, and λ_z can be shared only in $\mathbb{F}_p$.

The computation is done via the equation $e_z = v_z + \lambda_z = v_x^c + \lambda_z = ((v_x + \lambda_x) - \lambda_x)^c + \lambda_z$, where again $v_x + \lambda_x$ is replaced by a constant during garbling, and is known during evaluation.

Thus, the garbled rows are the set of encryptions

$$Enc_{k_{x,\alpha}}\left[\ k^i_z + (((v_x + \lambda_x) - \lambda_x)^c + \lambda_z)\,\Delta^i_p\ \right] \tag{26}$$

for every $\alpha \in \mathbb{F}_p$ and $i \in [n]$. Computing these encryptions is via the protocol of [3] or the protocol of [13]. Verification is similar to previously explained gates.

B Optimized Offline Protocol for Honest Majority

In this section we describe an optimized offline protocol that incorporates the ideas presented by Ben-Efraim and Omri [7]. The main idea is that the outputs of the hash function (as well as the zero keys) are not shared. Instead, they are locally added by the parties after performing a share conversion from Shamir shares to additive shares. The share conversion is by local multiplication by the reconstruction constant by each party. However, it is important to note that the converted shares are not fully random. Therefore, an additional zero-sharing must be added to rerandomize these shares. In the Boolean case, [7] showed that using these optimizations reduces the offline assymptotic complexity and concrete running time. [7] also suggest other optimizations, which seem to be applicable here as well, but we did not verify.

The optimized protocol is as follows:

Step 1, Setup: For each prime p in the primorial modulus, each party P_i does the following:
- For each wire $\omega \in \mathcal{W}_p$ randomly chooses a random element $(\lambda_\omega)_i \in \mathbb{F}_p$ and key $k^i_\omega \in \mathbb{F}_{p^\kappa}$. The random permutation element on the wire is $\lambda_\omega \stackrel{\text{def}}{=} \Sigma_{i=1}^n [(\lambda_\omega)_i]$.
- In topological order on the circuit, computes k^i_ω for each wire $\omega \notin \mathcal{W}_p$, by summing/multiplying by a constant (according to gate type).

- Randomly chooses a random global offset $\Delta_p^i \in \mathbb{F}_{p^\kappa}$.
- For each garbled component $g \in C$ with input wire x, compute $\mathcal{F}_{k^i_{x,\alpha}}(g||j)$ for each $j \in [n]$ and $\alpha \in \mathbb{F}_p$, where p is according to the gate/component type.

Step 2, Sharing: Each party P_i shares the permutation elements and offsets in Step 1 using $(t+1)$-out-of-n Shamir secret-sharing scheme. Multifield-shared bits are also shared using Protocol 2 in Section 2.3 for each p. The parties obtain shares of λ_ω for each wire by locally summing their shares of $\{(\lambda_\omega)_i\}_{i=1}^n$. For free gates, in topological order, the parties locally compute the shares for the output wires by locally summing/multiplying by a constant (according to gate type). Additionally, for each wire garbled component with output wire in $\mathcal{W}_p$, each party generates and sends *additive shares* of the zero vector $(0,\ldots,0) \in \mathbb{F}_{p^\kappa}$.

Step 3, Computing the garbled gates: Shares of the garbled rows of each garbled gate/component are computed using their respective Equation as in the original offline protocol, with the following important differences:

- The parties first compute shares for the part of the equation with the multiplication by the Δ's as usual (BGW/ [3]), except the last degree-reduction round is omitted.
- The parties perform a share-conversion of the above shares to *additive shares* by locally multiplying with their respective reconstruction constant.
- The parties locally add their zero keys of the output wire[14] and the outputs of the hash function.
- The parties mask the above *additive shares* by adding the zero-shares received from all the parties for that gate.

Step 4, Reconstructing the outputs: The parties reconstruct the outputs of F_{GC}, namely the garbled gates/components and the output permutation elements. Note that the garbled gates/components are now additively shared (and not Shamir shares as in the original protocol). Furthermore, each party receives the shares and reconstructs the permutation bits on its input wires.

C Definitions for Secure Computation

We follow the standard definition of secure multiparty computation for semi-honest adversaries, as it appears in the book "Foundations of Cryptography, Voume II Basic Applications" by Oded Goldreich. In brief, an n-party protocol π is defined by n interactive probabilistic polynomial-time Turing machines $\mathrm{P}_1,\ldots,\mathrm{P}_n$, called parties. The parties hold the security parameter 1^κ as their joint input and each party P_i holds a private input x_i. The computation proceeds in rounds. In each round j of the protocol, each party sends a message to each of the other parties (and receives messages from all other parties). The number of rounds in the protocol is expressed as some function $r(\kappa)$ in the security parameter.

The view of a party in an execution of the protocol contains its private input, its random string, and the messages it received throughout this execution. The random variable $\mathrm{view}^\pi_{\mathrm{P}_i}(\boldsymbol{x}, 1^\kappa)$ describes the view of P_i when executing π on inputs $\boldsymbol{x} = (x_1,\ldots,x_n)$ (with security parameter κ). Here, x_i denotes the input of party P_i. The output an execution of π on $\boldsymbol{x}$ (with security parameter κ) is described by the random variable $\mathrm{Output}^\pi(\boldsymbol{x}, 1^\kappa) = \left(\mathrm{Output}^\pi_{\mathrm{P}_1}(\boldsymbol{x}, 1^\kappa),\ldots,\mathrm{Output}^\pi_{\mathrm{P}_n}(\boldsymbol{x}, 1^\kappa)\right)$, where $\mathrm{Output}^\pi_{\mathrm{P}}(\boldsymbol{x}, 1^\kappa)$ is the output of party P in this execution, and is implicit in the view of P.

Similarly, for a set of parties with indices $I \subseteq [n]$, we denote by $\boldsymbol{x}_I$ the set of their inputs, by $\mathrm{view}^\pi_I(\boldsymbol{x}, 1^\kappa)$ their joint view, and by $\mathrm{Output}^\pi_I(\boldsymbol{x}, 1^\kappa)$ their joint output. In the setting of this work, it suffices to consider deterministic functionalities. We therefore provide the definition of security only for deterministic functionalities.

Definition 2 (security for deterministic functionalities). *A protocol π t-securely computes a deterministic functionality $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ in the presence of semi-honest adversaries if the following hold:*

[14] Some gates, e.g., Evaluator half gate, require also adding/subtracting some other key locally

Correctness: *For every $\kappa \in \mathbb{N}$ and every n-tuple of inputs $\boldsymbol{x} = x_1, \ldots, x_n$, it holds that*

$$\Pr\left[\text{Output}^{\pi}(\boldsymbol{x}, 1^{\kappa}) = f(\boldsymbol{x})\right] = 1, \tag{27}$$

where the probability is taken over the random coins of the parties.

Privacy: *There exists a probabilistic polynomial-time (in the security parameter) algorithm $\mathcal{S}$ (called "simulator"), such that for every subset $I \subseteq [n]$ of size at most t:*

$$\left\{\mathcal{S}_A\left(\boldsymbol{x}_I, f_I(\boldsymbol{x}), 1^{\kappa}\right)\right\}_{\boldsymbol{x}\in(\{0,1\}^*)^n;\kappa\in\mathbb{N}} \overset{C}{\equiv} \left\{\text{view}_I^{\pi}\left(\boldsymbol{x}, 1^{\kappa}\right)\right\}_{\boldsymbol{x}\in(\{0,1\}^*)^n;\kappa\in\mathbb{N}}. \tag{28}$$

D Direct Construction of Full Selector Gate

In this section we show how to exploit an asymmetry in the half selector gate construction, in order to construct a full selector gate without additional cost. Recall that a full selector gate is a gate that receives as input two wires in $\mathbb{F}_p$ and a Boolean selection wire, and the output corresponds to one of the input wires according to the selection bit. Notice that a full selector can be seen as the addition of 2 half selectors, with the selection bit of the second being the negation of the selection bit of the first. Also note that the chooser gate is treated differently by the evaluator according to the external value – if the selection bit is 0 then the key is only the decrypted key while if the selection bit is 1 then the key of the input wire is added to the decrypted key. Furthermore, the behaviour on the other half selector gate is exactly the opposite, as the selector bit of the second half selector gate is the negation of the selector bit.

To give an informal overview of the full selector gate, it is computed via the equation

$$v_x v_{w_0} + v_y \overline{v_{w_0}} + \lambda_z = \begin{cases} \overbrace{v_y + \widehat{\lambda_z}}^{\text{“Chooser Gate”}} & \overbrace{+\lambda_{w_0}(v_x - v_y) + \widehat{\lambda_z}}^{\text{“Corrector Gate”}}, \quad v_{w_0} \oplus \lambda_{w_0} = 0 \\ v_x + \widehat{\widehat{\lambda_z}} & -\lambda_{w_0}(v_x - v_y) + \widetilde{\widehat{\lambda_z}}, \quad v_{w_0} \oplus \lambda_{w_0} = 1, \end{cases} \tag{29}$$

with $\overline{v_{w_0}}$ being the negation of v_{w_0}. This equation is reached by summing two instances of Equation 17, once with x, w_0 and once with $y, \overline{w_0}$, and simplifying. We next explain the details of the chooser and corrector gates for the full selector gate.

Chooser Partial Gate for Full Selector. The chooser partial gate is constructed very similarly to the half selector case. The main difference is that if the external value of the selection wire is 0, the evaluator adds the key on the input wire y to the decrypted value (instead of just taking the decrypted value as in the half-selector gate).

Similarly to the chooser in the half selector, we use the equations $v_x + \widehat{\lambda_z} = (v_x + \lambda_x) - \lambda_x + \widehat{\lambda_z}$ and $v_y + \widehat{\widehat{\lambda_z}} = (v_y + \lambda_y) - \lambda_y + \widehat{\widehat{\lambda_z}}$. Concretely, the chooser partial gate is the following encryptions for every $i \in [n]$:

$$Enc_{k_{w_0,0}}\left[\ \widehat{k_z}^{\,i} - k_y^i + \left(-\lambda_y + \widehat{\lambda_z}\right)\Delta_p^i\ \right], \tag{30}$$

$$Enc_{k_{w_0,1}}\left[\ \widehat{\widehat{k_z}}^{\,i} - k_x^i + \left(-\lambda_x + \widehat{\widehat{\lambda_z}}\right)\Delta_p^i\ \right]. \tag{31}$$

Verification: if $e_{w_0} = 0$ then the key on the y wire is added to the decrypted value, and the recovered key is $[k_y + e_y\Delta_p] + \left[\widehat{k_z} - k_y + \left(-\lambda_y + \widehat{\lambda_z}\right)\Delta_p\right] = \widehat{k_z} + \left(v_y + \widehat{\lambda_z}\right)\Delta_p = \widehat{k}_{z,v_y+\widehat{\lambda_z}}$. Verification for $e_{w_0} = 1$ is symmetric.

Corrector Partial Gate for Full Selector. The main difference of the corrector from the half selector case, is that instead of decrypting according to the external value and key on the x wire, we decrypt according to the external value and key of $x - y$. We recall that subtraction in $\mathbb{F}_p$ is free.

To be more concrete, we compute the value of the first corrector gate using the equation $\lambda_{w_0}(v_x - v_y) + \widetilde{\lambda_z} = \lambda_{w_0}\left((v_x + \lambda_x) - (v_y + \lambda_y)\right) + \lambda_{w_0}(\lambda_y - \lambda_x) + \widetilde{\lambda_z}$. The value $(v_x + \lambda_x) - (v_y + \lambda_y)$ is known to the evaluator at the evaluation phase. At the garbling phase, this value is treated as a constant. Therefore, the garbled rows are

$$Enc_{k_{x,\alpha}}\left[\ \widetilde{k_z^i} + \left(\alpha\lambda_{w_0} + \lambda_{w_0}(\lambda_y - \lambda_x) + \widetilde{\lambda_z}\right)\Delta_p^i\ \right]. \tag{32}$$

for $\alpha \in \mathbb{F}_p$ and $i \in [n]$, where $k^i_{x-y,\alpha} \stackrel{\text{def}}{=} k^i_x - k^i_y + \alpha\Delta^i_p$. At evaluation phase, if $e_{w_0} = 0$, the evaluating party first computes $e_{x-y} = (v_x + \lambda_x) - (v_y + \lambda_y)$ and the corresponding key using free subtraction, and then decrypts the row e_{x-y}.

The construction of the second corrector gate is similar and omitted. The main security ideas (protecting against decryption of the "wrong" corrector) are as in the half-selector gate.

E A proof of Claim 1

In this section we prove Claim 1. The proof follows by a reduction from Definition 1. For the convenience of the readers, we repeat the definition and the claim here.

Definition 3 (Definition 1). *Let H be a hash function, and for each p in some set of primes P let $\Delta_p \in \mathbb{F}_{p^\kappa}$. We define an oracle $\mathcal{O}_P^H$ that acts as follows:*

$$\mathcal{O}_P^H(\rho, a, b, k, \gamma, \delta) = H(k + \gamma\Delta_{p_a}, \rho) + \delta\Delta_{p_b} \tag{33}$$

where $\rho \in \mathbb{N}$, $p_a, p_b \in P$, $\gamma \in \mathbb{F}_{p_a}$, $\delta \in \mathbb{F}_{p_b}$, $k \in \mathbb{F}_{{p_a}^\kappa}$, and the output of H is interpreted as in $\mathbb{F}_{{p_b}^\kappa}$. Note that $\gamma\Delta_{p_a}$ is the inner offset and $\delta\Delta_{p_b}$ is the outer offset. Legal queries *to the oracle have inputs in the correct domains and satisfy:*

1. *The oracle is never queried with $\gamma = 0$,*
2. *For each ρ, all the queries have the same p_a, p_b, and each $\gamma \in \mathbb{F}_{p_a} \setminus 0$ is used in at most one query.*

We say that H is mixed-modulus circular correlation robust *if for all polynomial time adversaries making only legal queries to the oracle, the oracle $\mathcal{O}_P^H$, for random Δ_ps, is indistinguishable from a random function (with the same input/output domains).*

Claim 3 (Claim 1) *Let $B \subset [n]$. If H is mixed-modulus circular correlation robust, then for all polynomial time adversaries making only legal queries to the oracle, the oracle*

$$\mathcal{O}_P^{H,B}(\rho, a, b, \left(k^i\right)_{i\in B}, \gamma, (\delta_i)_{i\in B}) \stackrel{def}{=} \Sigma_{i\in B}\left[\mathcal{O}_P^{H,i}(\rho, a, b, k^i, \gamma, \delta_i)\right], \tag{34}$$

where each $\mathcal{O}_P^{H,i}$ is equal to $\mathcal{O}_P^H$ with random and independent Δ_ps for each $p \in P$ and $i \in B$, is indistinguishable from a random function (with the same input/output domains).

Proof Sketch. Assume by contradiction that there exists a distinguisher D that uses legal queries to an oracle $\mathcal{O}$ and can distinguish if it is $\mathcal{O}_P^{H,B}$ or a random function. We construct a distinguisher $\mathcal{D}$ that breaks the MMCCR assumption. The oracle $\mathcal{D}$, which can make legal queries to an oracle $\mathcal{O}' \in \left\{\mathcal{O}_P^H, Rand\right\}$, proceeds as follows:

The distinguisher $\mathcal{D}$ first chooses an index $j \in B$.[15] Then, $\mathcal{D}$ randomly chooses $\tilde{\Delta}^i_p \in \mathbb{F}_{p^\kappa}$ for each $p \in P$ and $i \in B \setminus \{j\}$. The distinguisher $\mathcal{D}$ then runs the distinguisher D, while answering the (legal) queries of D (to its oracle) using the equation:

$$\mathcal{O}(\rho, a, b, \left(k^i\right)_{i\in B}, \gamma, (\delta_i)_{i\in B}) \stackrel{\text{def}}{=} \Sigma_{i\in B\setminus\{j\}}\left(H(k^i + \gamma\tilde{\Delta}^i_{p_a}, \rho) + \delta_i\tilde{\Delta}^i_{p_b}\right) + \mathcal{O}'(\rho, a, b, k^j, \gamma, \delta_j). \tag{35}$$

The proof now concludes with the following 3 observations:

[15] We assume that B is hardcoded in $\mathcal{D}$.

1. The distinguisher $\mathcal{D}$ is polynomial and makes only legal calls to the oracle $\mathcal{O}'$. This follows because D is polynomial and makes only legal calls to $\mathcal{O}$. The added computation made by $\mathcal{D}$ is only a polynomial number of computations of H.
2. If $\mathcal{O}' = \mathit{Rand}$ then $\mathcal{O}$ distributes as a random function. This is because every new legal oracle call to $\mathcal{O}$ by D results in a new oracle call to $\mathcal{O}'$ by $\mathcal{D}$, and summing with random is random.
3. If $\mathcal{O}' = \mathcal{O}_P^H$ then

$$\begin{aligned}
&\mathcal{O}(\rho, a, b, \left(k^i\right)_{i\in B}, \gamma, (\delta_i)_{i\in B}) \\
&= \Sigma_{j\in B\setminus\{i\}} \left(H(k^i + \gamma\tilde{\Delta}^i_{p_a}, \rho) + \delta_i\tilde{\Delta}^i_{p_b}\right) + \mathcal{O}'(\rho, a, b, k^j, \gamma, \delta_j) \\
&= \Sigma_{j\in B\setminus\{i\}} \left(H(k^i + \gamma\tilde{\Delta}^i_{p_a}, \rho) + \delta_i\tilde{\Delta}^i_{p_b}\right) + H(k^i + \gamma\Delta_{p_a}, \rho) + \delta_i\Delta_{p_b}.
\end{aligned}$$

   Since the $\tilde{\Delta}^i_p$'s were chosen randomly, they distribute the same as the Δ_p's of the $\mathcal{O}_P^{H,i}$'s. Thus, $\mathcal{O}$ distributes as $\mathcal{O}_P^{H,B}$.

Following the above, we conclude that since D can distinguish between cases 2 and 3 with non-negligible probability, so does $\mathcal{D}$, breaking the MMCCR assumption on H. □

F OT Protocol

In this section we present an offline protocol for arithmetic circuits that is secure for any number of semi-honestly corrupt parties. The main difference from Protocol Π_{GC} in Section 2.4 is that we change the sharing of the permutation elements from Shamir shares to additive shares in $\mathbb{F}_p$, and multiplication of shared elements is done using OT.

We first present the two basic protocols, which show how to multiply two secret-shared $\mathbb{F}_p$ elements and how to multiply a secret-shared $\mathbb{F}_p$ element with an offset $\Delta^i_p \in \mathbb{F}_{p^\kappa}$ of a single party. Let λ_x, λ_y be secret-shared $\mathbb{F}_p$ elements, and denote the share of party i by $(\lambda_x)_i$. Further let $\Delta^i \in \mathbb{F}_{p^\kappa}$ be known to party i.

1. To compute (additive shares of) $\lambda_x\lambda_y$, we first observe that

$$\lambda_x\lambda_y = (\Sigma^n_{i=1}[(\lambda_x)_i])(\Sigma^n_{i=1}[(\lambda_x)_i]) = \Sigma^n_{i=1}[(\lambda_x)_i(\lambda_y)_i] + \Sigma_{i\neq j}[(\lambda_x)_i(\lambda_y)_j]).$$

   We now observe that $(\lambda_x)_i(\lambda_y)_i$ can be computed locally by party i and a secret-sharing of $(\lambda_x)_i(\lambda_y)_j$ can be computed by parties i and j in the standard way by running a 1-out-of-p OT[16]: party i is plays the receiver with chosen element $(\lambda_x)_i$ and party j plays the sender with the elements $\{r + \alpha(\lambda_{inpb})_j\}_{\alpha\in\mathbb{F}_p}$ for a random $r \in \mathbb{F}_p$; the share of party i is the received message and the share of party j is r. Summing over all the parties results in an additive sharing of $\lambda_x\lambda_y$.
2. To compute (additive shares of) $\lambda\Delta^j_p$, we first observe that

$$\lambda_x\Delta^j_p = (\Sigma^n_{i=1}[(\lambda_x)_i])\Delta^j_p = \Sigma^n_{i=1}[(\lambda_x)_i\Delta^j_p].$$

   Now a secret-sharing of $(\lambda_x)_j\Delta^i_p$ can be computed by parties i and j in the standard way by running a 1-out-of-p *string*-OT[16]: party i is plays the receiver with chosen element $(\lambda_x)_i$ and party j plays the sender with the strings $\left\{r + \alpha\Delta^i_p\right\}_{\alpha\in\mathbb{F}_p}$ for a random $r \in \mathbb{F}_{p^\kappa}$; the share of party i is the received message and the share of party j is r. Summing over all the parties results in an additive sharing of $\lambda_x\Delta^i_p$ – note that these shares are not random, and therefore must be rerandomized; see Step 3 below.

Using these two basic protocols, the OT based garbling protocol proceeds as follows:

Step 1, Setup: For each prime p in the primorial modulus, each party P_i does the following:

[16] Or alternatively using $\log p$ 1-out-of-2 OTs

- For each wire $\omega \in \mathcal{W}_p$ randomly chooses a random element $(\lambda_\omega)_i \in \mathbb{F}_p$ and key $k^i_\omega \in \mathbb{F}_{p^\kappa}$. The random permutation element on the wire is $\lambda_\omega \stackrel{\text{def}}{=} \Sigma_{i=1}^n [(\lambda_\omega)_i]$.
- In topological order on the circuit, computes k^i_ω for each wire $\omega \notin \mathcal{W}_p$, by summing/multiplying by a constant (according to gate type).
- Randomly chooses a random global offset $\Delta^i_p \in \mathbb{F}_{p^\kappa}$.
- For each garbled component $g \in C$ with input wire x, compute $\mathcal{F}_{k^i_{x,\alpha}}(g||j)$ for each $j \in [n]$ and $\alpha \in \mathbb{F}_p$, where p is according to the gate/component type.

Step 2, Computing the garbled gates: Shares of the garbled rows of each garbled gate/component are computed using their respective Equation as in the original offline protocol, with the following important differences:
- The multiplication of the permutation elements is done using protocol 1.
- The multiplication of the permutation elements with the offsets is done using protocol 2.
- The parties locally add their zero keys of the output wire[17] and the outputs of the hash function. Note that now the parties hold a *non-random* additive "sharing" of the gates (i.e., the "shares" sum to the output, but are not uniformly random).

Step 3, Rerandomization: The parties reshare the result from the last step to receive *random* additive shares of the garlbed gates.[18]

Step 4, Reconstructing the outputs: The parties reconstruct the outputs of F_{GC}, namely the garbled gates/components and the output permutation elements. Furthermore, each party receives the shares and reconstructs the permutation bits on its input wires.

Note that mixed Boolean-arithmetic circuits and exponenetiation by a constant gates also require an efficient protocol for unbounded fan-in multiplication. It does not seem straightforward to use the protocol of [3] *efficiently* in this case. Possibly, a more direct $\log n$ round protocol should be considered here.

G Additional Details on Proof of Theorem 2

Note that from the definition of secure computation, we need to show that there exists a simulator which can simulate the view of the adversary, and this should be computationally indistinguishable from the view of the adversary in a real protocol execution. The view of the adversary, in the F_{GC}-hybrid model, consists of the following:

- The output of F_{GC}, i.e., the garbled circuit, the output permutation elements, the permutation elements on the input wires of the adversary, the random global offsets of the corrupt parties, and the zero keys of the corrupt parties on all the wires.
- The online rounds, i.e., the external values on (all) the input wires and the corresponding keys (of all the parties).

We first note that although the external values are chosen differently by the simulator (chosen at random) and in the real protocol (correspond to $v_\omega + \lambda_\omega = v_\omega + \Sigma_{i=1}^n[(\lambda_\omega)_i]$), they distribute identically (since λ_ω is random in the real protocol). Similarly, for the corresponding keys k^i_{ω,e_ω} of the honest parties $i \in B$, which are chosen randomly by the simulator, and in the real protocol correspond to $k^i_\omega + e_\omega \Delta^i_p$. But again the distribution is the same since k^i_ω is random. For the keys of the corrupt parties, the simulator follows the real protocol, i.e., chooses random zero keys and global offsets, and computes the keys corresponding to the external values as usual – notice that this is possible because the zero keys and offsets of the corrupt parties are part of the view of the adversary.

Once the external values and corresponding keys for the honest parties are chosen on all the wires, simulating the online rounds poses no additional difficulty. Additionally, simulating the output permutation elements is done using the chosen external values and real outputs (given from the trusted party). Similarly,

[17] Some gates, e.g., Evaluator half gate, require also adding/subtracting some other key locally

[18] This step can be optimized using secret-sharings of zero vectors, as in the Appendix B.

supplying the permutation elements on the input wires of the adversary is done using the external values and the inputs of the adversary.

Thus, the proof boils down to showing how the simulator can simulate the garbled circuit. It is important to note that the simulator *cannot* build a circuit that distributes the same as the real circuit, because this would imply the simulator can compute the real values on all the wires. However, the simulator can build a fake garbled circuit in which one row is correctly encrypted, and the other rows are random $\mathbb{F}_{p^\kappa}$ vectors. Then, it suffices to show that being able to distinguish between this fake garbled circuit and a real garbled circuit, for some inputs, breaks the MMCCR assumption. This is explained in Section 5. As explained there, the distinguisher uses the oracle to compute the garbled gates, such that they distribute as a real or fake circuit according to the oracle type.

The complete computations of the distinguisher, and proving that the circuit distributes as a fake or real garbled circuit according to the oracle, is quite tedious. We next give example computations for the evaluator half gate (the computations for this gate are the most complex).

Example computations for evaluator half gate. Recall first that the multiplication gate has two input wires x, y and the output wire z. The external values on the input wires are e_x and e_y respectively. For the evaluator half gate, the output external value is $\widehat{e_z} = v_x(v_y + \lambda_y) + \widehat{\lambda_z}$. The garbled gate is the set of encryptions

$$\widehat{g}^j_\beta = Enc_{k_{y,\beta}}\left[\ \widehat{k}^i_z - \beta k^i_x + \left(-\beta\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p\ \right] \tag{36}$$

for every $\beta \in \mathbb{F}_p$ and $i \in [n]$.

Now notice that for row e_y (this is the row that is decrypted) the equations can be rewritten as

$$\begin{aligned}\widehat{g}^j_{e_y} &= Enc_{k_{y,e_y}}\left[\ \widehat{k}^i_z - e_y k^i_x + \left(-e_y\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p\ \right]\\ &= Enc_{k_{y,e_y}}\left[\ \widehat{k}^i_z - e_y k^i_{x,e_x} + \left(e_y v_x + \widehat{\lambda_z}\right)\Delta^i_p\ \right]\\ &= Enc_{k_{y,e_y}}\left[\ \widehat{k}^i_{z,\widehat{e_z}} - e_y k^i_{x,e_x}\ \right].\end{aligned}$$

Now we can clearly see that the simulator can compute this row, since it knows $e_x, e_y, \widehat{e}_z$ and the corresponding keys for all the parties. For the other rows, the simulator sends random $\mathbb{F}_{p^\kappa}$ elements.

The distinguisher's job is clearly harder – it can compute the row e_y, i.e., the $\widehat{g}^j_{e_y}$'s, as the simulator, but it must also compute the other rows, using the oracle, such that they distribute as the fake or real rows. We next give the details of this computation.

For computing $\widehat{g}^j_\beta$, the simulator first computes $\gamma = e_y - \beta$. Next, the distinguisher uses the inputs to compute the real values on the input wires v_x, v_y and the value $\widehat{v_z} = v_x e_y$. Using this, the distinguisher computes $b_\beta = -\beta(e_x - v_x) + (\widehat{e_z} - \widehat{v_z}) = -\beta\lambda_x + \widehat{\lambda_z}$ (this value is the value the encrypted keys in this row correspond to in a real execution of the protocol with the same real and external values).

The computation now splits for the parts of the honest parties and the parts of the corrupt parties, i.e., for $j \in B$ and $j \notin B$. Denote $p = p_a$ (i.e., p is the ath prime in P).

– For $j \in B$ the distinguisher queries the oracle for

$$\mathcal{O}_{g,\beta,j} = \mathcal{O}^{H,B}_P(g||j, a, a, \left(k^i_{y,e_y}\right)_{i\in B}, \gamma, (\delta_i)_{i\in B})$$

where $\delta_i = \begin{cases} b_\beta - \widehat{e}_z + \beta e_x & i = j \\ 0 & i \neq j \end{cases}$, and computes

$$\hat{g}^j_\beta = \left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_{x,e_x} + \widehat{k}^j_{z,\widehat{e_z}} + \mathcal{O}_{g,\beta,j}. \tag{37}$$

Note that for $j \in B$, the distinguisher knows k^j_{x,e_x} and $\widehat{k}^j_{z,\widehat{e_z}}$, but does not know Δ^j_p.

– For $j \notin B$, the distinguisher queries the oracle for

$$\mathcal{O}_{g,\beta,j} = \mathcal{O}_P^{H,B}(g||j, a, a, \left(k^i_{y,e_y}\right)_{i\in B}, \gamma, (\delta_i)_{i\in B})$$

where $\delta_i = 0$ for all i, and computes

$$\hat{g}^j_\beta = \left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_x + k^j_{z,b_\beta} + \mathcal{O}_{g,\beta,j}. \tag{38}$$

Note that for $j \notin B$, the distinguisher knows k^j_x and can compute $\widehat{k}^j_{z,b_\beta}$ from $\widehat{k}^j_z$ and Δ^j_p.

We first observe that the distinguisher makes only legal queries to the oracle, since for each fixing of g and j, the oracle queries only once for each $0 \neq \gamma \in \mathbb{F}_p$.

We next observe that if the oracle is a random function, then $\widehat{g}^j_\beta$ results in an independent random $\mathbb{F}_{p^\kappa}$ element from both Equations (37) and (38). This is because each computation uses a different oracle call, and summing with random is random.

The final observation is that when the oracle is $\mathcal{O}_P^{H,B}$, the result distributes as in the real garbled circuit. We again split for $j \in B$ and $j \notin B$.

– If $j \in B$, then

$$\begin{aligned}
\hat{g}^j_\beta &= \left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_{x,e_x} + \widehat{k}^j_{z,\widehat{e_z}} + \mathcal{O}_{g,\beta} \\
&= \left[\left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_{x,e_x} + \widehat{k}^j_{z,\widehat{e_z}}\right] + \left[\Sigma_{i\in B}\left(H(k^i_{y,e_y} + \gamma\Delta^i_p, g||j) + \delta_i\Delta^i_p\right)\right] \\
&= \left[\left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_{x,e_x} + \widehat{k}^j_{z,\widehat{e_z}}\right] + \left[\Sigma_{i\in B}\left(\mathcal{F}_{k^i_{y,e_y+\gamma}}(g||j)\right) + (b_\beta - \widehat{e}_z + \beta e_x)\Delta^j_p\right] \\
&= \left(\Sigma_{i\in[n]}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \left(\beta k^j_{x,e_x} - \beta e_x\Delta^j_p\right) + \left(\widehat{k}^j_{z,\widehat{e_z}} + (b_\beta - \widehat{e}_z)\Delta^j_p\right) \\
&= \left(\Sigma_{i\in[n]}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_x + \widehat{k}^j_{z,b_\beta} = Enc_{k_{y,\beta}}\left[\ \widehat{k}^j_{z,b_\beta} - \beta k^j_x\ \right] \\
&= Enc_{k_{y,\beta}}\left[\ \widehat{k}^i_z - \beta k^i_x + \left(-\beta\lambda_x + \widehat{\lambda_z}\right)\Delta^i_p\ \right].
\end{aligned}$$

– If $j \notin B$, then

$$\begin{aligned}
\hat{g}^j_\beta &= \left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_x + \widehat{k}^j_{z,b_\beta} + \mathcal{O}_{g,\beta} \\
&= \left[\left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_x + \widehat{k}^j_{z,b_\beta}\right] + \left[\Sigma_{i\in B}\left(H(k^i_{y,e_y} + \gamma\Delta^i_p, g||j) + \delta_i\Delta^i_p\right)\right] \\
&= \left[\left(\Sigma_{i\notin B}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_x + \widehat{k}^j_{z,b_\beta}\right] + \left[\Sigma_{i\in B}\left(\mathcal{F}_{k^i_{y,e_y+\gamma}}(g||j)\right)\right] \\
&= \left(\Sigma_{i\in[n]}\mathcal{F}_{k^i_{y,\beta}}(g||j)\right) - \beta k^j_x + \widehat{k}^j_{z,b_\beta} = Enc_{k_{y,\beta}}\left[\ \widehat{k}^j_{z,b_\beta} - \beta k^j_x\ \right] \\
&= Enc_{k_{y,\beta}}\left[\ \widehat{k}^j_z - \beta k^j_x + \left(-\beta\lambda_x + \widehat{\lambda_z}\right)\Delta^j_p\ \right].
\end{aligned}$$

H Precomputing Multi-Field Shared Bits

The garbling phase is often computed "offline", meaning before the parties choose their inputs. However, it does require knowledge of the circuit. Whenever possible, it is desirable to preprocess as much as possible even before the function the parties wish to compute is known. This is especially important when the computations are relatively heavy, as in Protocol 2 in Section 2.3. That is, the offline phase is split into two: the function independent preprocessing, in which primitives are computed without knowledge of the circuit, and function dependent preprocessing, in which computations depend on the circuit, but do not require the inputs.[19]

[19] The function ind. preproc. is often also split to two, and computations independent of the number of gates are called the setup phase. The required number of multifield-shared bits depends on the number of projection, equality, and selector gates. Thus, precomputing multifield-shared bits is *not* part of the setup phase.

The main problem of precomputing multi-field shared bits is that it is not compatible with the free-XOR technique. For instance, assume we want the permutation bit λ_{w_0} of wire w_0 to be a multi-field shared bit, but w_0 is the output of an XOR gate. Thus, λ_{w_0} should be computed via $\lambda_{w_0} = \lambda_x \oplus \lambda_y$, and therefore we cannot use our precomputed λ_{w_0}. One way to solve this issue is to use buffer gates, but this adds additional garbled rows and decryptions. We next explain a more efficient solution, based on the key allignment idea from [30].

To solve this issue the parties will precompute a multi-field shared bit $\tilde{\lambda}_{w_0}$ and choose random keys $\tilde{k}^i_{w_0}$. These are independent of the values λ_{w_0} and $k^i_{w_0}$, which are computed by the standard construction (i.e., by XORing the permutation bits/keys of the input wires). Then, for each *garbled* gate that w_0 is an input wire to, the computations are done using $\tilde{\lambda}_{w_0}$ and $\tilde{k}^i_{w_0}$.

In order to be able to use $\tilde{\lambda}_{w_0}$ and $\tilde{k}^i_{w_0}$ in the evaluation phase, in the offline protocol the parties broadcast the following two messages:

1. Each party i broadcasts $(\tilde{\lambda}_{w_0})_i \oplus (\lambda_{w_0})_i$. Summing these messages the parties recover $\tilde{\lambda}_{w_0} \oplus \lambda_{w_0}$.
2. Each party i broadcasts the difference $k_{w_0} \oplus \tilde{k}_{w_0} \oplus (\tilde{\lambda}_{w_0} \oplus \lambda_{w_0})\Delta^i_p$.

Notice that $e_{w_0} \oplus (\tilde{\lambda}_{w_0} \oplus \lambda_{w_0}) = \tilde{e}_{w_0}$ and $k_{w_0.e_{w_0}} \oplus (k_{w_0} \oplus \tilde{k}_{w_0} \oplus (\tilde{\lambda}_{w_0} \oplus \lambda_{w_0})\Delta^i_p) = \tilde{k}_{w_0,\tilde{e}_{w_0}}$. This implies that the evaluating parties can compute $\tilde{e}_{w_0}$ and $\tilde{k}_{w_0,\tilde{e}_{w_0}}$ at the evaluation phase. By rearranging the equations, this also implies that the simulator can simulate the two messages above (the first message requires more work because the simulator has to simulate $(\tilde{\lambda}_{w_0})_i \oplus (\lambda_{w_0})_i$ for each party i; the details are ommited).