MYao: Multiparty "Yao" Garbled Circuits with Row Reduction, Half Gates, and Efficient Online Computation

Aner Ben-Efraim, Lior Breitman[0009−0003−6993−4442],
Jonathan Bronshtein[0009−0009−7583−0109],
Olga Nissenbaum[0000−0001−9819−5560], and Eran Omri[0000−0001−8928−0587]

Ariel University, Ariel , Israel 4070001
anermosh@post.bgu.ac.il, liorbreitman321@gmail.com, jon9028@gmail.com,
olganis@ariel.ac.il, omrier@gmail.com

Abstract. Garbled circuits are a powerful and important cryptographic primitive, introduced by Yao [FOCS 1986] for secure two-party computation. Beaver, Micali and Rogaway (BMR) [STOCS 1990] extended the garbled circuit technique to construct the first constant-round secure multiparty computation (MPC) protocol. In the BMR protocol, the garbled circuit size grows linearly and the online computation time grows quadratically with the number of parties. Previous solutions to avoid this relied on key-homomorphic PRFs, incurring a large garbled circuit size and slow online computation time.

We present **MYao**, a new *multiparty* protocol for achieving a "Yao" garbled circuit, i.e., the garbled circuit size and online computation time are independent of the number of parties. The key innovation is that the parties collaboratively compute the PRF in MPC, which was previously believed to be inefficient. In this paper, we challenge this long-standing assumption by basing the garbled circuit construction on "MPC-friendly" PRFs. One of the highlights of our new technique is that we are able to achieve, for the first time, full row-reduction in *multiparty* garbled circuits. To achieve this optimization without increasing the number of rounds, we utilize free-XOR and half gates, presenting a new technique for choosing the keys, based on a naturally occurring relation between the 2 keys of the 2 half-gates.

MYao reduces the garbled circuit size by more than 90%, the total communication by more than 75%, and the online computation time by more than 10%, compared to all known solutions based on key-homomorphic PRFs, thus substantially improving the overall efficiency in both the offline and the online phases. Furthermore, MYao significantly improves over semi-honest BMR in online phase efficiency when the number of parties exceeds 80.

Keywords: Multiparty Garbled Circuits· Multiparty Row Reduction· Offline-Online MPC.

1 Introduction

Secure multiparty computation (MPC) protocols allow mutually distrusting parties to jointly compute a function of their private inputs, while guaranteeing various security properties, the most basic being correctness (i.e., the correct function is computed) and privacy (i.e., nothing but the output is revealed). The two mainly considered types of adversaries are semi-honest (i.e., follows the protocol, but may try to infer additional information from its view) and malicious (i.e., may arbitrarily deviate from the protocol).

In his seminal work, Yao [37] proposed a general-purpose secure two-party protocol in the semi-honest setting. In Yao's protocol, the two parties jointly evaluate the circuit that describes the function to be computed, where one party (the *garbler*) encrypts the circuit in a way that allows the second party (the *evaluator*) holding the keys (corresponding to their inputs) to decrypt (only) the output of the circuit. A key feature of Yao's protocol is that it runs in a constant number of rounds.

Yao's result was generalized to the multiparty setting by Goldreich et al. [19] and by Beaver et al.[5]. These two works followed the circuit evaluation paradigm of Yao's protocol, but differed in the manner this is performed. The GMW protocol of [19] introduced the *secret sharing paradigm*, where parties compute the circuit gate by gate, with the invariant that in coming to compute each gate the parties hold a secret sharing of the inputs for that gate. The BMR protocol of [5] extended the protocol of Yao following the *garbled circuit paradigm*, letting parties jointly garble the circuit (a task that can be done securely using a fairly simple MPC protocol). After the garbled circuit is computed, each of the parties obtains the appropriate keys and can locally evaluate the circuit on the inputs of all parties.

The shortcoming of existing solutions in the setting with high-latency and large number of parties. In the past couple of decades, MPC has shifted from a purely theoretical research area to one that also has much real-world necessity and practicality. Indeed, extremely concretely efficient protocols (i.e., protocols providing fast run-times in practice) exist for the setting of very few parties. In particular, secure two-party protocols have become very practical and are being deployed in various real-world solutions. However, real-life scenarios often require that a large number of parties perform the computation over a high-latency network (e.g., the Internet). This setting still poses significant challenges in practice as we further explain below.

Almost all known concretely efficient general-purpose MPC constructions follow either the secret sharing paradigm or the garbled circuit paradigm. Another common and useful paradigm is the *offline-online paradigm*. The idea is that in many scenarios, much of the heavy lifting computation can be done ahead of time, i.e., in the offline (preprocessing) phase, before parties know their inputs. Then, the actual (presumably, much lighter) computation is performed in the online phase, after the parties learn their inputs.

In the *secret sharing paradigm* [19], the parties compute the circuit in the online phase, layer by layer, where the inputs to the gates at each layer are secret shared. Therefore, the number of communication rounds depends on the depth of the circuit. Hence, the online phase of MPC protocols in the secret-sharing paradigm is inherently slow in high-latency networks for deep circuits.

One of the main advantages of the *garbled circuit paradigm* [37, 5] is that it allows the entire MPC protocol to be computed in a constant number of rounds. In addition, the garbling of the circuit does not depend on the inputs of the parties and can be pushed to the offline phase, leaving only key revelation to the online phase. While this seems very promising for high latency networks, it turns out that for BMR based constructions [5, 6, 9, 22, 35], the size of the garbled circuit grows linearly in the number of parties and online computation time grows quadratically in the number of parties. Thus, these constructions become impractical as the number of parties grows.

A "naïve" approach for tackling this issue is for the parties to jointly construct a Yao garbled circuit in the preprocessing phase, hence making the garbled circuit size optimal. This solution was considered impractical, as it would render a very inefficient preprocessing phase. A somewhat similar idea, based on key-homomorphic PRFs, was introduced in [10, 8, 18]. However, their constructions still fell short in providing overall efficiency.

In this paper, we suggest a way, based on "MPC-friendly" PRFs with certain properties, to advance towards a concretely efficient MPC protocol in the many-party high latency setting – a joint construction of a "Yao" garbled circuit (i.e., independent of the number of parties). Somewhat surprisingly, our protocol benefits from optimizations that were thus far only achievable in two-party garbled circuit based constructions. We implement this general idea using the "MPC-friendly" PRFs of Dinur et al. [14], which we show to possess the required properties. This results in a concretely-efficient protocol with competitive runtime.

Semi-honest Security. In this work we consider only semi-honest security. In real situations, malicious security is of course always preferable. Nevertheless, it has been shown time and again that introducing techniques to improve efficiency in the semi-honest setting, soon brings similar improvements for the more involved malicious setting. Furthermore, there are known generic semi-honest to malicious compilers, e.g., [19, 23, 13].

We believe the techniques and results presented in this paper are significant, and will prove highly beneficial in constructing maliciously secure protocols enjoying similar qualities. Below, we refer to the semi-honestly secure versions in all explained protocols.

1.1 Background on Garbled Circuits

Before explaining our results and techniques, we first give a more detailed overview of the garbled circuit paradigm and its optimizations, in the *semi-honest* setting.

Garbled circuits. Yao's garbled circuit protocol consists of two parties: a garbler and an evaluator. The garbler assigns two keys to each wire of the circuit, corresponding to its two possible values, i.e., $k_{w,0}, k_{w,1}$ for the 0 and 1 value of the wire w, respectively.

Each gate is then constructed as a table of four ciphertexts, corresponding to the gate's truth table. In each row the appropriate output wire key is encrypted by the corresponding input wire keys. To avoid revealing any information about the true values on the wires, the garbler can use the *point-and-permute* technique, where it chooses a random masking bit $\lambda_w \in \{0,1\}$ for every wire w. During evaluation, the evaluator will learn only the *external* value $e_w = v_w \oplus \lambda_w$, where v_w is the real value on the wire w. Using a 2-keyed Encryption scheme Enc, a garbled AND-gate, with input wires $in1, in2$ and output wire out, consists of the four ciphertexts

$$\tilde{g}_{\alpha,\beta} = \mathrm{Enc}_{\mathrm{k_{in1},\alpha},\mathrm{k_{in2},\beta}}\left(\mathrm{k_{out,e_{out}^{\alpha,\beta}}} || \mathrm{e_{out}^{\alpha,\beta}}\right), \tag{1}$$

for $\alpha, \beta \in \{0,1\}$, where $e_{out}^{\alpha,\beta} = \lambda_{out} \oplus (\alpha \oplus \lambda_{in1})(\beta \oplus \lambda_{in2})$ is the *conditional* external value on wire out, i.e., the external value in the case this row is decrypted. Garbling of other gate types can be similarly defined, but as explained below, XOR gates can in fact be computed "for free" using the free-XOR technique. Implementations of Yao's protocol use a variety of encryption schemes for Enc, the most common technique being XORing with the output of a pseudo-random function (PRF).

Upon receiving from the garbler the garbled circuit, the masking bits for the output wires, and the keys and external values corresponding to the input wires (for the inputs of the evaluator these are received via oblivious transfer), the evaluator can decrypt the circuit locally. This is computed gate by gate in topological order, by choosing the appropriate row of the gate according to the external values and using the corresponding keys to decrypt the output wire's key and external value. After decrypting all the gates, the evaluator can recover the true outputs by XORing with the masking bits of the output wires.

Garbled circuits were extended to the multiparty setting by Beaver, Micali and Rogaway [5]. In the BMR protocol, for each wire w of the circuit, each party P_i for $i \in [n]$ has two keys: $k_{w,0}^i$ and $k_{w,1}^i$. The wire keys (or *superseeds*, as called in [5]) are the tuple of keys belonging to all the parties, i.e., $k_{w,b} = \{k_{w,b}^i\}_{i=1}^n$ for $b \in \{0,1\}$. Furthermore, the masking bits λ_w are secret-shared, as they must remain hidden from an evaluating party, even if some of the parties are colluding. For input and output wires, the masking bits are revealed to the relevant parties.

Garbling an AND gate in the BMR protocol is achieved by the parties computing shares for the output keys for each row in MPC, and then each party encrypting its shares, by XORing it with a 2-keyed PRF F^2. These are then reshared between the parties, XORed, and reconstructed, resulting in the BMR garbled gates:

$$\tilde{g}_{\alpha,\beta} = \left\{ \bigoplus_{i=1}^{n} \left(F^2_{k_{u,\alpha}^i, k_{v,\beta}^i}(g_j) \right) \oplus k^j_{w, e_w^{\alpha,\beta}} \right\}_{j=1}^{n}, \tag{2}$$

for $\alpha, \beta \in \{0,1\}$, where g_j is a unique index corresponding to the gate g and party index j, and $e_w^{\alpha,\beta}$ is as above. An important observation is that all the garbled gates can be computed in parallel, leading to a constant round protocol.

The BMR online phase has two rounds of communication: in the first, the parties broadcast the external values on their input wires, computed using their inputs and masking bits. In the second, all the parties broadcast their keys of the input wires that correspond to the external values on those wires.

Upon receiving the garbled BMR circuit along with keys corresponding to the input wires and permutation bits of the output wires, an evaluating party can decrypt the garbled circuit locally in topological order, using its own pair of keys to decide the external value at each wire.

Notice that the length of the garbled rows (Equation (2)) grows linearly in the number of parties n. Furthermore, each key of each party is encrypted by the keys of all the parties. Thus, decrypting a single row requires n^2 decryptions (i.e., invocations of the PRF F^2). As a consequence, evaluation time in the BMR protocol grows *quadratically* in the number of parties.

Free-XOR The free-XOR technique was introduced by Kolesnikov and Schneider [25] for Yao's protocol, and later extended to the BMR protocol by Ben-Efraim et al. [9]. It is also used in several other multiparty garbled circuit protocols, e.g., [28, 35, 8, 36]. Free-XOR allows XOR gates to be garbled "for free", meaning that they require no communication and no encryption or decryption.

To achieve this, for every wire of the circuit, the key corresponding to the '1' value is set to be $k_{w,1} = k_{w,0} \oplus \Delta$, where Δ is a *global* uniformly random constant. For wires that are not output wires of XOR gates, the key $k_{w,0}$ and masking bit are chosen uniformly at random as before. For output wires of XOR gates, the key $k_{w,0}$ and the wire mask are set to be the XOR of those of the input wires, i.e., $k_{out,0} = k_{in1,0} \oplus k_{in2,0}$ and $\lambda_{out} = \lambda_{in1} \oplus \lambda_{in2}$. Then, during evaluation, output keys and external values of an XOR gate can be similarly computed via XOR operations by the evaluator.

As observed by [12], the free-XOR technique requires an additional security requirement on the PRF used for the encryption, termed *circular correlation robustness*. Intuitively, this is due to the fact that the same global Δ is used both in the encrypting keys and the encrypted keys, and therefore the PRF used must protect against exploiting this. The formal definition of circular correlation robustness can be found in Section 3.1. Furthermore, as commented in [9], free-XOR in the multiparty setting requires the secret-sharing schemes used to be *linear* over a *characteristic 2* field (for example, using XOR secret sharing). Hence, works that used schemes over fields of characteristic $\neq 2$, e.g., [6, 10], could not incorporate free-XOR.

Row Reduction. Row reduction, introduced for Yao's protocol by Naor et al. [29], allows to reduce communication and garbled circuit size by fixing the first row of each garbled gate to a public constant, usually 0. Thus, the first row in each garbled gate does not need to be sent to the evaluator, as it is always 0. This can be achieved from Equation (1) by setting $k_{out,e_{out}^{0,0}} || e_{out}^{0,0} = \mathrm{Enc}^{-1}_{\mathrm{k_{in1,0}},\mathrm{k_{in2,0}}}(0)$.

It is important to observe that this implies that the keys and masking bits are not chosen randomly, but rather "derived" from the keys of the input wires. This implies that the gates are computed sequentially and not in parallel. This is problematic for a constant round protocol in the multiparty setting, and is perhaps one of the main reasons why, prior to this work, there has been almost no advance in row reduction for the multiparty setting. In fact, to the best of our knowledge, the best known result in this regard is by Yang et al. [36], which achieves only a $2/n$-row reduction.

Half-gates. The above row-reduction technique was later extended by Zahur et al. [38], to reduce the size of the garbled gates to only 2 ciphertexts, using *half gates*. The main idea of half gates is that instead of computing a single garbled gate, the garbler computes two "half"-gates, each encrypted with a single input key, such that the XOR of the outputs of both halves equals the required output of the original gate. As each half-gate is encrypted using only a single key, it requires only 2 ciphertexts, while XOR requires no ciphertexts, assuming free-XOR. Then, using the above row-reduction technique on each of the half-gates, each half gate requires only a single ciphertext to be sent, and in total only 2 ciphertexts are sent.

Half gates have been explored in the multiparty setting in [7, 36], but the result for row-reduction has been extremely limited. In particular, [36] achieved a $2/n$-row reduction and [7] used half-gates only to reduce multiplication size in characteristic > 2 and achieved no row-reduction.

BMR Alternatives via Key-Homomorphic PRFs. A solution for averting the BMR asymptotic growth of the garbled circuit size and online computation time was recently suggested in [10, 8, 18]. The idea is that the parties hold *additive shares* of the garbled circuit keys and compute the garbled circuit using a *key-homomorphic* PRF $\mathcal{F}$. A key-homomorphic PRF intuitively means that there exist operations $+, \tilde{+}$ on the PRF key space domain and output range, respectively, such that the homomorphic property $F_{k_1}(x)\tilde{+}\cdots\tilde{+}F_{k_m}(x) = F_{k_1+\cdots+k_m}(x)$ holds for any $F_{k_1}, \ldots, F_{k_m} \in \mathcal{F}$ and input x.

To garble the gates each party *locally* computes the PRF on a public value using its share of the input key as the key, i.e., party P_i computes $\{F_{k^i_{u,\alpha}}(g)\}_{\alpha\in\{0,1\}}$ and $\{F_{k^i_{v,\beta}}(g)\}_{\beta\in\{0,1\}}$. Then, using the homomorphic property, the parties compute in MPC additive shares of

$$F_{k^1_{u,\alpha}}(g)\tilde{+}\cdots\tilde{+}F_{k^n_{u,\alpha}}(g) = F_{k^1_{u,\alpha}+\cdots+k^n_{u,\alpha}}(g) = F_{k_{u,\alpha}}(g) \tag{3}$$

for $\alpha \in \{0,1\}$, and similarly for the input keys of wire v. Additive shares of the key that needs to be encrypted are securely computed similarly to the BMR protocol, along with shares of the conditional external value, and thus the parties can locally obtain shares of the garbled gates $\tilde{g}_{\alpha,\beta} = F_{k_{u,\alpha}}(g_{\alpha,\beta}) + F_{k_{v,\beta}}(g_{\alpha,\beta}) + \left(k_{w,e_w^{\alpha,\beta}} || e_w^{\alpha,\beta}\right)$, for $\alpha, \beta \in \{0,1\}$. Notice that the parties only need to compute in MPC the homomorphic property (the $\tilde{+}$ operations in Equation (3) left), which

is usually significantly easier than securely computing the PRF on the shared key directly (Equation (3) right).

The upside of this technique is that the PRFs are computed only locally by each party, also in the offline phase. On the downside, key-homomorphic PRFs usually depend on public key encryption techniques, implying that they require relatively large key and encryption sizes and have slow computation time. Thus, the key-homomorphic PRF garbled circuit schemes presented in [10, 8, 18] have large garbled circuit size and increased online communication. Furthermore, the online computation time, although independent of the number of parties, is relatively slow.

1.2 Our Results and Techniques

The above discussion raises several interesting questions.

New Method for Multiparty Garbling. As observed in [9], the garbled circuit size and online computation time in the BMR protocol grow linearly and quadratically in the number of parties, respectively. This becomes problematic when the number of parties is very large. The known solution to avoid these asymptotics is by using key-homomorphic PRFs [10, 8, 18], but the suggested key-homomorphic PRFs have long keys and encryption sizes, and also slow decryption time.

Thus, it is natural to ask whether key-homomorphic PRFs are essential for efficiently constructing a multiparty "Yao" garbled circuit. By multiparty "Yao" garbled circuit, we mean a garbled circuit that has size and online computation time independent of the number of parties, and the keys and masking bits are secret-shared amongst the parties.

Question 1 *Can a "Yao" garbled circuit be efficiently constructed by a secure multiparty protocol without key-homomorphic PRFs?*

The main difficulty in constructing a "Yao" garbled circuit in the multiparty setting is computing the encryption on the secret-shared keys. As the keys are not known to any party, without the homomorphic property, the encryption scheme itself needs to be computed in MPC. Thus, it was widely believed to be impractical.

However, recently there has been significant progress in "MPC-friendly" PRFs [20, 1, 3, 15, 2, 16]. The term "MPC-friendly" implies that these PRFs can be computed efficiently in MPC on shared keys. By using "MPC friendly" PRFs, we show that a multiparty "Yao" garbled circuit can be efficiently constructed in MPC; we call this construction **MYao**. In order to minimize the number of offline communication rounds, while also achieving free-XOR and row-reduction (explained below), we chose to concretely instantiate MYao with the "MPC-friendly" PRFs by alternating moduli of Dinur et al. [15]. Dinur et al.'s constructions can be computed extremely efficiently in MPC in the correlated randomness model, where the required correlated randomness is a simple primitive – uniformly random bits doubly shared over the fields $\mathbb{Z}_2$ and $\mathbb{Z}_3$. This primitive is a specific case

of *dBits* (or *daBits* in the malicious model), i.e., doubly-shared bits, which is a well studied primitive [7, 11, 33, 4, 27, 17, 26, 15]. There are several suggested efficient protocols for computing dBits and daBits, some of which are constant round.

We show that MYao results in approximately 90% smaller garbled circuit sizes, compared to garbled circuits from key-homomorphic PRFs [10, 8, 18]. To fully compare the online time, we implemented the online computation phase of MYao. As part of the implementation process, we made the following observation overlooked by [15]: there are some $\mathbb{Z}_3$ elements whose value is actually limited to $0, 1$. This observation allows us to significantly optimize the local wPRF computation; the details appear in Section 11. Our experiments show that MYao improves the online computation time by at least 10% compared to solutions based on key-homomorphic PRFs, and in circuits containing XOR gates the improvement is even greater. Additionally, MYao's online time is faster than BMR's online time when the number of parties exceeds 80.

We also show that MYao improves by more than 70% in offline communication compared to key-homomorphic PRFs solutions. To this end, we observe that the main bottleneck in offline communication in MYao is the preprocessing of dBits. Although efficient constant round protocols for computing dBits and daBits exist, these protocols were designed for general fields. Hence, we suggest an alternative preprocessing protocol, specifically tailored for the case of dBits over $\mathbb{Z}_2$ and $\mathbb{Z}_3$.

The protocol goes roughly as follows: each party randomly chooses its random share b^i in $\mathbb{Z}_2$ and each party P_i except P_n chooses its random share t^i in $\mathbb{Z}_3$. The parties then compute, using a simple MPC protocol based on OT, the correct share t^n for the party P_n, such that $\bigoplus_{i=1}^{n} b^i = \sum_{i=1}^{n} t^i \mod 3$. To minimize communication, the computation proceeds in a "tournament-style" manner, and each party participates only in the parts where its shares are required. The full details appear in Section 10.

Although our alternative preprocessing protocol has $O(\log n)$ rounds, it requires less communication than the constant round protocols mentioned above. We remark that it seems there is still significant room for improvement in preprocessing dBits. We leave this as an open question for future work.

Garbled Circuit Optimizations. As already mentioned, free-XOR can be incorporated into multiparty garbled circuits, if the secret-sharing scheme is linear over a field of characteristic 2 and the PRF used for encrypting is circular-correlation robust. MYao's encryption scheme has Binary keys and requires only XOR sharing for the construction. Therefore, if it is possible to construct a circular correlation-robust "MPC friendly" PRF with binary keys, which we believe is a reasonable assumption, free-XOR can be naturally incorporated into the MYao garbled circuit.

However, regarding other garbled circuit optimizations, such as row reduction and half gates, the situation is less clear. Prior to this work, it was not known whether row reduction can be incorporated into multiparty garbled circuits – below we explain why straightforward attempts did not succeed. Additionally,

as the benefit of half-gates seems to rely on row-reduction, it also raises the question of whether using half-gates can give any real benefit in multiparty garbled circuits. Thus, the following two questions are natural:

Question 2 *Can full row reduction be achieved efficiently in a constant number of rounds in multiparty garbled circuit protocols?*

Question 3 *Can half gates significantly aid in garbling Boolean circuits in the multiparty setting?*

At first glance, it may seem that row reduction should not be possible for the general multiparty setting, for two main difficulties:

1. In the row reduction technique the gates are computed sequentially, as output keys are derived from the input ones. Computing the gates in parallel is crucial for achieving a constant round protocol. Yang. et al. [36] bypassed this problem and achieved a $\frac{2}{n}$-row reduction in BMR, by letting one of the parties compute its PRFs sequentially. This is possible as in BMR the PRFs are computed locally, but this technique cannot be extended to a full row-reduction.
2. The row reduction technique implies that the size of the key used for the encryption and the size of the garbled row (i.e., the size of the ciphertext) should be the same. Therefore, it seems not possible to incorporate row-reduction in the LPN-based protocol of [8], as it requires error-correcting codes, and, hence, the garbled rows are significantly longer than the keys. This point also raises a question on the feasibility of achieving full row-reduction in BMR.

Without row-reduction, it might seem that half gates have little, if any, positive effect. Thus, it may appear that the answer to both questions should be negative.

Nevertheless, we show that row-reduction is indeed possible for multiparty garbled circuits, by using half-gates in MYao. We introduce a technique for choosing the output keys of each gate independently of all other gates. This is done by determining the output 0-key for one of the half-gates by the output of the PRF, while the 0-key on the output wire of the AND gate remains uniformly random. This allows us to construct a constant round protocol for computing the half-gates such that the first row of one of them is 0. The key shares for the output wire of the first half gate are computed using the shares of the PRF value. Then, the key shares on the output wire of the second half gate are computed using the shares of the keys on the output wires of the first half gate and the (overall) AND gate. Using this construction, we achieve row-reduction without increasing the number of rounds. We note that the reliance on free-XOR implies that this technique cannot be used for the key-homomorphic protocols of [10].

Additionally, we show that in some cases, such as when constructing the PRF using the LPN-style "MPC-friendly" PRF of [15], half-gates are beneficial even without row-reduction, as they reduce the offline computation and communication, without significantly increasing the online computation. The reason is that

when garbling using only a single-keyed PRF, a regular AND gate requires 8 computations of the PRF in the offline phase, while the half-gates AND gate requires only 4 computations of the PRF. Thus, the offline phase computation and communication are significantly reduced when using half-gates. In the online phase, both regular AND gates and half-gate based AND gates require 2 PRF invocations for decryption. As the overwhelmingly dominant factor in the online computation is computing the PRF, the online computation time remains almost the same for both constructions (despite half-gates requiring significantly more XOR operations).

Thus, we answer both questions affirmatively. We also discuss scenarios in which it is possible to achieve even a *double row reduction* for all or some of the gates, with all other gates still maintaining a single row reduction.

1.3 Other Relevant Works

We mention that many garbled circuit optimizations remained outside the scope of this paper, e.g., double-row reduction was first achieved by Pinkas et al. [31], but their technique is not compatible with free-XOR. The FleXOR technique [24] improves over free-XOR in some cases, but was superseded by half-gates [38]. There are some recent improvements over half-gates, most notably the "slicing and dicing" technique of [32] that reduces garbled AND gates to almost 1.5 rows and are compatible with free-XOR.

Regarding the reduction of BMR garbled circuit size, the so-called "Tinykeys" technique, introduced by Hazay et al. [21], showed how the BMR garbled circuit size can be significantly reduced when some portion of the parties are assumed to be honest. Although the reduction is remarkable, we show it still results in bigger garbled circuits than MYao, even though MYao does not make any assumption on the number of honest parties.

Paper Organization. In Section 2 we review some basic definitions and constructions necessary for understanding our work. In Section 4 we give the basic MYao protocol. In Section 5 we explain how to incorporate half-gates and row-reduction into the MYao protocol. In Section 9 we give the details of jointly computing the garbled gate efficiently using an "MPC friendly" PRF from [15]. In Section 11 we give the details of our implementation and compare MYao with state-of-the-art solutions based on garbled circuits.

2 Preliminaries

In this section, we briefly explain the security model, notation, and some well known definitions and protocols necessary for understanding our work.

Notation. We denote concatenation by $||$. The number and set of parties are denoted by n and $\mathcal{P} = \{\mathsf{P}_1, ..., \mathsf{P}_n\}$, respectively. We denote the computational security parameter by κ, and the length of the key for a PRF which ensures κ-bit

security by $\ell(\kappa)$. Note that, depending on the PRF, $\ell(\kappa)$ might be significantly longer than κ. Bit-string multiplication is denoted by $\cdot$, i.e. for $b \in \{0,1\}$ and $s = (s_1, ..., s_\kappa) \in \{0,1\}^\kappa$, $b \cdot s \stackrel{def}{=} (bs_1, ..., bs_\kappa)$.

2.1 Basic Functionalities and Protocols

In this section, we describe the main building blocks of our construction such as secret sharing, oblivious transfer, secure bit multiplication.

The two basic operations which parties from $\mathcal{P}$ compute jointly, are bit-bit and bit-string multiplications. In this section, following the work [9], we show two protocols for such operations that are secure in the presence of a static semi-honest adversary corrupting up to $n-1$ parties. These protocols are built on a folklore two-party bit-bit and bit-string multiplication OT-based protocols given in this section. Before bringing the multiplication functionalities and protocols, we'll give the short explanation of the 1-out-of-2 Oblivious transfer (OT) functionality and XOR additive secret sharing which are necessary for the multiparty multiplication protocols we use.

Oblivious Transfer. The bit 1-out-of-2 Oblivious Transfer 2-party functionality, or further bit-OT, allows two parties P_1 and P_2 to securely compute a function $f_{OT}((a_0, a_1), b) = (\cdot, a_b)$, where P_1 is called a sender with inputs $a_0, a_1 \in \{0,1\}$, and P_2 is called a receiver, with his choice bit $b \in \{0,1\}$ as the input. The sender's output is empty, and the receiver's output is only a single of sender's input bits a_b.

The κ-bit string-OT is defined in a similar way as $f^\kappa_{OT}((s_0, s_1), b) = (\cdot, s_b)$, where $s_0, s_1 \in \{0,1\}^\kappa$ are two strings.

XOR Secret Sharing A bit x is XOR-shared, if every $\mathsf{P}_i \in \mathcal{P}$ holds a random bit $x_i \in \{0,1\}$ such that $x_1 \oplus \cdots \oplus x_n = x$. For two XOR-shared bits x and y, a XOR-sharing of their sum, $x \oplus y$, can be locally computed by each party computing $x_i \oplus y_i$. Similarly, a bitstring $s \in \{0,1\}^\ell$ is XOR-shared if each party P_i holds a random string $s_i \in \{0,1\}^\ell$ such that $\bigoplus_{i=1}^n s_i = s$.

Secure two-party bit-bit multiplication. We define the two-party functionality $f_\times(b_1, b_2) = (c_1, c_2)$, where $b_1 b_2 = c_1 \oplus c_2$. The protocol computing $f_\times$ for parties P_1 with the input $b_1 \in \{0,1\}$, and P_2 with the input $b_2 \in \{0,1\}$, works as follows:

1. P_1 chooses a random $r \in \{0,1\}$, sets $x_0 = r$, $x_1 = r \oplus b_1$.
2. P_1 as a sender with input (x_0, x_1), and P_2 as a receiver with input b_2 run a bit oblivious transfer. P_2 receives x_{b_2}.
3. P_1 outputs r, and P_2 outputs $x_{b_2} = r \oplus b_1 b_2$.

The (semi-honest) security of this protocol is trivial in the OT-hybrid model, since the only communication is done via the OT, hence P_1 learns nothing, and P_2 learns only it's output. This protocol requires only a single OT call.

Secure two-party bit-string multiplication. The two-party functionality $f_{\times}^{\kappa}(s,b) = (s_1, s_2)$, where $s \cdot b = s_1 \oplus s_2$, and $s, s_1, s_2 \in \{0,1\}^{\kappa}$, $b \in \{0,1\}$ can be securely computed using a single string-OT in the same way as $f_{\times}$, where P_1 inputs $(r, r \oplus s)$, and P_2 inputs b in the f_{OT}^{κ} functionality. Here, $r \leftarrow \{0,1\}^{\kappa}$.

Secure multiparty bit-bit multiplication. Now we show how n parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ having the additive shares of two bits a and b could securely compute the shares of their product $c = ab$ in the $f_{\times}$-hybrid model. We denote this functionality as $f_{mult}((a_1, b_1), \ldots, (a_n, b_n)) = (c_1, \ldots, c_n)$. Notice, that every P_i holds two bits a_i and b_i such that $a = a_1 \oplus \ldots \oplus a_n$, $b = b_1 \oplus \ldots \oplus b_n$, and

$$c = ab = \left(\bigoplus_{i=1}^{n} a_i\right)\left(\bigoplus_{i=1}^{n} b_i\right) = \left(\bigoplus_{i=1}^{n} a_i b_i\right) \oplus \left(\bigoplus_{i \neq j} a_i b_j\right). \tag{4}$$

Each P_i can locally compute $a_i b_i$. Every P_i and P_j (where $i \neq j$) run $f_{\times}(a_i, b_j)$ and $f_{\times}(b_i, a_j)$, and obtain output shares of $a_i b_j$ and $a_j b_i$ respectively. Finally, each party P_i XORs it's local share product $a_i b_i$ together with all it's output shares from $f_{\times}$ instances. Denote the result as c_i. By (4), $c = c_1 \oplus \ldots \oplus c_n$.

This protocol requires $n(n-1)$ parallel bit-OT instances, thus its round complexity is constant and equals to the round complexity of the underlying (two-party, semi-honest) bit-OT. Each party participates in $2(n-1)$ OT instances.

Security of this protocol relies on the fact that all communications occur in $f_{\times}$ instances, and everything each party sees is a random share of terms in (4), hence is easy simulated.

Secure multiparty bit-string multiplication. The n-party bit-string multiplication functionality f_{mult}^{κ} when each $\mathsf{P}_i \in \mathcal{P}$ holds XOR-additive shares b_i and s_i of the bit b and the κ-bit string s, and obtains the XOR-additive share of $b \cdot s$, can be securely computed in the similar way as the n-party bit-bit multiplication in a $f_{\times}^{\kappa}$ hybrid model and requires $n(n-1)$ parallel string-OT calls, where each party takes part in $2(n-1)$ OT's.

We denote bit-bit and bit-string multiplication by Π_{mult} and Π_{mult}^{ℓ}, respectively.

Pseudorandom Functions (PRF) and weak Pseudorandom Functions (wPRF) Informally, a PRF is a function family $\mathcal{F}$, such that for a randomly chosen function $f \leftarrow \mathcal{F}$ and any input x, the value $f(x)$ is indistinguishable from a random string. A wPRF is similarly defined, except that the input x must also be random, i.e., for a randomly chosen function $f \leftarrow \mathcal{F}$ and a random known input x, the output $f(x)$ is indistinguishable from a random string. A formal definition for wPRFs is given in Section 3.2.

LPN-style "MPC-friendly" wPRF from alternating moduli [15]. We implemented MYao using the LPN-style wPRF of [15], which is defined as follows: Let $B \in$

$\mathbb{Z}_2^{t\times m}$ be a public matrix. For a key $k \in \mathbb{Z}_2^\ell$, let $K \in \mathbb{Z}_2^{m\times\ell}$ be the circulant matrix[1] defined by setting its first row equal to k. On an input $x \in \mathbb{Z}_2^\ell$, the wPRF $f_k \in \mathcal{F}$ is defined by

$$f_k(x) = B \cdot [K \cdot x \oplus (K \cdot x \mod 3) \mod 2], \tag{5}$$

where in $(K \cdot x \mod 3)$, both K and x are first reinterpreted over $\mathbb{Z}_3$. According to [15], by setting $\ell = m = 2\kappa$ and $t = \kappa$, this results in a weak PRF candidate with approximately κ-bit security that withstands known attacks. See [15] for more details and Section 9 for how we use this wPRF in the MYao garbled circuit construction.

3 Security Definitions

3.1 Circular Correlation Robustness

As was shown by Choi et al. [12], a useful cryptographic primitive for proving the security of a garbled circuit type construction with free-XOR is a *circular 2-correlation robust* encryption scheme. This property ensures security even if the adversary has multiple AND-gates where the pairs of input and output keys have the same difference Δ. For the half-gates optimization, each half-gate has only a single key. The required security definition for an underlying cryptographic primitive was revisited by Zahur, et al. in [38], who replaced it with *circular correlation robustness*. In **MYao**, as we show below, the property of circular correlation robustness for the wPRF F is sufficient, not only for the half-gates version, but even for the basic **MYao**. We next give the definition of *circular correlation robustness* for a single-keyed PRF.

For a function family $\mathcal{F} = \left\{F_k : \{0,1\}^{2\ell(\kappa)} \to \{0,1\}^{(\ell(\kappa)+1)} \,|k \in \{0,1\}^{\ell(\kappa)}\right\}$, for $F \in \mathcal{F}$, and $\tilde{\Delta} \in \{0,1\}^{\ell(\kappa)}$, we define the oracle $Circ^F_{\tilde{\Delta}}$ as follows. For $b \in \{0,1\}$, the output of $Circ^F_{\tilde{\Delta}}(k,g,b)$ is $F_{(k\oplus\tilde{\Delta})}(g) \oplus b \cdot (\tilde{\Delta}||1)$.[2] The oracle *Rand* has the same input and output domain as $Circ^F_{\tilde{\Delta}}$, and returns fresh random responses from $\{0,1\}^{\ell(\kappa)+1}$ for new queries, while on previously queried values, it returns the same answer. We say that a distinguisher $\mathcal{D}$ makes *legal* queries to the oracle $\mathcal{O} \in \left\{Circ^F_{\tilde{\Delta}}, Rand\right\}$, if for any values k, g, it doesn't make both queries $(k,g,0)$ and $(k,g,1)$.

Definition 1 (Circular Correlation Robustness). *A function family* $\left\{F_k : \{0,1\}^{2\ell(\kappa)} \to \{0,1\}^{(\ell(\kappa)+1)} \,|k \in \{0,1\}^{\ell(\kappa)}\right\}$ *is circular correlation robust if,*

[1] A circulant matrix is a matrix in which each row after the first is a cyclic rotation of the first row by one place. Note that a circulant matrix is a Toeplitz matrix, and is fully defined by its first row and its dimensions.

[2] The definition of Zahur et al. was given for hash-functions, where $Circ^H_{\tilde{\Delta}}(k,g,b)$ outputs $H(k \oplus \tilde{\Delta}||g) \oplus b \cdot (\tilde{\Delta}||1)$.

for every non-uniform polynomial-time distinguisher $\mathcal{D}$, making only legal queries to its oracle,

$$\left|\Pr[\mathcal{D}^{Circ_{\tilde{\Delta}}^{F}}(1^{\kappa})=1]-\Pr[\mathcal{D}^{Rand}(1^{\kappa})=1]\right| \tag{6}$$

is negligible in κ when $\tilde{\Delta} \leftarrow \{0,1\}^{\ell(\kappa)}$.

As we prove in Theorem 2, the property of circular correlation robustness for a single-keyed PRF F is enough even for the security of basic **MYao** (without the half-gates optimization), when $F^2_{k_1,k_2}(g) = F_{k_1}(g) \oplus F_{k_2}(g)$, and we never allow the duplicated input wire to any AND gate in the circuit, such that the case $k_1 = k_2$ could appear only at random with the negligible probability.

3.2 Weak Pseudorandom Functions (wPRF)

Definition 2 (Weak Pseudorandom Function (wPRF), [15]). *Let $\mathcal{K} = \{\mathcal{K}_\kappa\}_{\kappa\in\mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\kappa\}_{\kappa\in\mathbb{N}}$, and $\mathcal{Y} = \{\mathcal{Y}_\kappa\}_{\kappa\in\mathbb{N}}$ be ensembles of finite sets indexed by a security parameter κ. Consider an efficiently computable function family $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$ where each function is given by $\mathcal{F}_\kappa : \mathcal{K}_\kappa \times \mathcal{X}_\kappa \to \mathcal{Y}_\kappa$. We say that $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$ is an (l, t, ε)-weak pseudorandom function, if for infinitely many $\kappa \in \mathbb{N}$ and all adversaries $\mathcal{A}$ running in time at most $t(\kappa)$, the following holds:*

Let f_κ, k, and $x_1, \ldots, x_l$ be sampled uniformly at random from $\boldsymbol{Funcs}[\mathcal{X}_\kappa, \mathcal{Y}_\kappa]$, $\mathcal{K}_\kappa$, and $\mathcal{X}_\kappa$ correspondingly, where $\boldsymbol{Funcs}[\mathcal{X}_\kappa, \mathcal{Y}_\kappa]$ denotes the set of all function from $\mathcal{X}_\kappa$ to $\mathcal{Y}_\kappa$. Then

$$\left|\Pr\left[\mathcal{A}(1^\kappa, \{x_i, \mathcal{F}_\kappa(k, x_i)\}_{i\in[l]})\right] - \Pr\left[\mathcal{A}(1^\kappa, \{x_i, f_\kappa(x_i)\}_{i\in[l]})\right]\right| \le \varepsilon(\kappa).$$

3.3 Definitions of Secure Multiparty Computation

We first give the basic definitions of secure multiparty computation.

Definition 3 (computational indistinguishability). *Let $\mathcal{X} = \{X_w\}_{w\in\{0,1\}^*}$ and $\mathcal{Y} = \{Y_w\}_{w\in\{0,1\}^*}$ be two ensembles. We say that $\mathcal{X}$ and $\mathcal{Y}$ are* computationally indistinguishable*, denoted $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$, if the following holds: For every polynomial-size circuit family, $\{C_n\}_{n\in N}$, every positive polynomial p, every sufficiently large n, and every $w \in \{0,1\}^n$,*

$$|\Pr[C_n(X_w) = 1] - Pr[C_n(Y_w) = 1]| < \frac{1}{p(n)} \tag{7}$$

Definition 4 (outputs; views). *Let Π be a multiparty protocol for computing a functionality f. The* view *of party P_i during an execution of Π on inputs $(x_1, \ldots, x_n)$, denoted view_i^Π is $(x_i, r_i, m_{i,1}, \ldots, m_{i,t})$, where r_i represents the outcome of P_i's internal coin tosses, and $m_{i,j}$ represents the j^{th} message it has received.*

The output of the party P_i after an execution of Π on $(x_1, \ldots, x_n)$ denoted by $\text{output}_i^\Pi(x_1, \ldots, x_n)$ is implicit in the party's own view of the execution, and

$$\text{output}^\Pi(x_1, \ldots, x_n) \overset{def}{=} (\text{output}_1^\Pi(x_1, \ldots, x_n), \ldots, \text{output}_n^\Pi(x_1, \ldots, x_n))$$

For a subset $A = \{i_1, \ldots, i_n\} \subset [n]$ of parties, we define their joint view,

$$\text{view}_A^{\Pi} \stackrel{def}{=} (A, \text{view}_{i_1}^{\Pi}, \ldots, \text{view}_{i_n}^{\Pi}).$$

Definition 5 (semi-honest multiparty security). *Let $f = (f_1, ..., f_n)$ be a functionality. We say that π securely computes a deterministic function f in the presence of static semi-honest adversaries if there exists probabilistic polynomial-time algorithms $\{S_A\}_{A \subset [n]}$ such that*

$$\{\text{output}^{\pi}(x_1, \ldots, x_n)\}_{x_1,\ldots,x_n \in \{0,1\}^*} \stackrel{c}{\approx} \{f(x_1, \ldots, x_n)\}_{x_1,\ldots,x_n \in \{0,1\}^*}, \quad (8)$$

and for each $A \subset [n]$

$$\{S_A((x_i)_{i \in A}, (f_i(x_1, \ldots, x_n))_{i \in A})\}_{x_1,\ldots,x_n \in \{0,1\}^*} \stackrel{c}{\approx} \{\text{view}_A^{\pi}(x_1, \ldots, x_n))\}_{x_1,\ldots,x_n \in \{0,1\}^*}. \quad (9)$$

4 The Basic MYao Protocol

In this section we explain the basic MYao protocol, without the row-reduction and half-gates optimizations: In Section 4.1 we give the details of MYao's offline functionality $\mathcal{F}_{\text{garble}}$ that creates a MYao garbled circuit. In Section 4.2 we describe MYao's offline phase, an efficient MPC protocol that implements functionality $\mathcal{F}_{\text{garble}}$. In Section 4.3 we explain the MYao online phase. Security is explained in Section 6.

In Section 5 we explain the changes required for applying the row-reduction and half-gates optimizations. The details of the LPN-style wPRF of [15], which allows MYao to be computed efficiently, and the protocol for computing this wPRF, are given in Section 9.

4.1 The Garbled Circuit Functionality

The garbling functionality of MYao essentially creates a Yao garbled circuit, and outputs it to each of the parties along with the masking bits for this party's respective input wires, the masking bits for the output wires of the circuit, and XOR shares of the keys for all input wires of the circuit. We next explain this in detail. The formal description is given in Figure 1. For simplicity, we consider a circuit C containing only AND and XOR gates, with no duplicated inputs (see [30] for security issues from duplicated inputs).

For each wire w of the circuit, the functionality associates:

- 2 keys, $k_{w,0}, k_{w,1} \in \{0,1\}^{\ell(\kappa)}$, corresponding to the two values 0 and 1, respectively, and
- A *masking bit* $\lambda_w \in \{0,1\}$, used to mask the real value of the wire during the evaluation phase.

Functionality $\mathcal{F}_{\text{garble}}$ Constructing a MYao Garbled Circuit

Input: The circuit C to be garbled.

1. Sample a global offset $\Delta \leftarrow \{0,1\}^{\ell(\kappa)}$.
2. For every wire w that is *not* an output wire of a XOR-gate, sample $k_{w,0} \leftarrow \{0,1\}^{\ell(\kappa)}$ and $\lambda_w \leftarrow \{0,1\}$, and compute $k_{w,1} = k_{w,0} \oplus \Delta$.
3. In topological order, for every XOR-gate with output wire w and input wires u, v, compute $k_{w,0} = k_{u,0} \oplus k_{v,0}$, $k_{w,1} = k_{w,0} \oplus \Delta$, and $\lambda_w = \lambda_u \oplus \lambda_v$.
4. For every AND-gate with input wires u, v and output wire w, compute the garbled gate $\{\tilde{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ as follows: For every α, $\beta \in \{0,1\}$, compute the conditional external value

$$e_w^{\alpha,\beta} = \lambda_w \oplus (\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \tag{10}$$

and the garbled row:

$$\tilde{g}_{\alpha,\beta} = F_{k_{u,\alpha}}(g_{\alpha,\beta}) \oplus F_{k_{v,\beta}}(g_{\alpha,\beta}) \oplus \left(k_{w,e_w^{\alpha,\beta}} || e_w^{\alpha,\beta}\right), \tag{11}$$

where $g_{\alpha,\beta}$ is a unique public index corresponding to the gate index and the row, and F is a circular-correlation robust PRF.[a]

Output:

- To all the parties, the garbled gates $\{\tilde{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ for all AND gates.[b]
- To all the parties, on each output wire *out* of the circuit, the masking bit λ_{out},
- To each party P_i, a random share Δ^i such that $\Delta = \bigoplus_{i=1}^n \Delta^i$, and, for every input wire w of the circuit, a random share $k_{w,0}^i$, such that $k_{w,0} = \bigoplus_{i=1}^n k_{w,0}^i$.
- To each party P_i, on each input wire *in* of the party, the masking bit λ_{in}.

[a] If F is a wPRF, $g_{\alpha,\beta}$ should be random; See Section 9 for more details.
[b] See Remark 1 on page 31.

Fig. 1. Functionality $\mathcal{F}_{\text{garble}}$

As the MYao protocol uses only XOR secret-sharing, it can utilize the free-XOR optimization. To enable free-XOR, the functionality sets $k_{w,1} = k_{w,0} \oplus \Delta$ on all the wires, where $\Delta \in \{0,1\}^{\ell(\kappa)}$ is a uniformly random *global* offset. Notice that this implies that for every $b \in \{0,1\}$, on all wires it holds that

$$k_{w,b} = k_{w,0} \oplus b \cdot \Delta. \tag{12}$$

Additionally, for all wires that are not output wires of XOR gates, $k_{w,0} \in \{0,1\}^{\ell(\kappa)}$ and $\lambda_w \in \{0,1\}$ are chosen uniformly at random. Then, in topological order, for all XOR gates with output wire w and input wires u, v, the functionality sets the 0-key and masking bit of wire *out* to be the XOR of the 0-keys and masking bits, respectively, of the input wires. I.e.,

$$k_{w,0} = k_{u,0} \oplus k_{v,0} \tag{13}$$

$$\lambda_w = \lambda_u \oplus \lambda_v. \tag{14}$$

The functionality then computes the garbled AND gates as follows:

$$\tilde{g}_{\alpha,\beta} = F_{k_{u,\alpha}}(g_{\alpha,\beta}) \oplus F_{k_{v,\beta}}(g_{\alpha,\beta}) \oplus \left(k_{w,e_w^{\alpha,\beta}} || e_w^{\alpha,\beta}\right), \tag{15}$$

where F is a circular-correlation robust PRF (see Section 3.1 for the definition of circular-correlation robustness), $g_{\alpha,\beta}$ is a unique index corresponding to the gate and row,[3] and

$$e_w^{\alpha,\beta} = \lambda_w \oplus (\alpha \oplus \lambda_u)(\beta \oplus \lambda_v) \tag{16}$$

is the conditional external value (i.e., the external value the evaluator will see on the wire if this row is decrypted).

The functionality then outputs to the parties the garbled gates and the masking bits on the output wires. Additionally, it distributes to the parties XOR shares of the keys on all the input wires of the circuit, and to each party P_i the masking bits on its input wires.

4.2 The MYao Offline Phase Protocol

In this section, we describe the garbling protocol that creates the MYao garbled circuit. This protocol is independent of the actual inputs of parties, and can therefore be run beforehand, in an *offline phase*. Since all the gates can be computed in parallel, the entire protocol has a constant number of rounds. The MYao garbling protocol Π_{offline} realizes the functionality $\mathcal{F}_{\text{garble}}$, and is explained below. The formal description is given in Figure 2.

In the first local step of Π_{offline}, the parties choose random shares Δ^i of the global offset Δ, and choose random shares $k^i_{w,0}$ and λ^i_w for the wire key $k_{w,0}$ and masking bit λ_w, respectively, for every wire w that is not the output of a XOR-gate. For every input wire in, attributed to Party P_j, every other party P_i $(i \neq j)$ sets its share of the mask λ^i_{in} to 0, resulting in $\lambda_w = \lambda^j_w$. Then, for every XOR-gate of the circuit in topological order, the parties locally compute the shares of the output keys and masks, by XORing the input ones. I.e., for an XOR-gate with the input wires u, v and output wire w, each party P_i computes $k^i_{w,0} = k^i_{u,0} \oplus k^i_{v,0}$, and $\lambda^i_w = \lambda^i_u \oplus \lambda^i_v$. Notice that this step does not require any interaction.

In the second step, the parties compute the garbling of all the AND-gates in parallel. In the basic MYao protocol (without row-reduction and half-gates), this is computed as follows: The parties compute secret-shares of the PRFs $F_{k_{u,\alpha}}(g_{\alpha,\beta})$ and $F_{k_{v,\beta}}(g_{\alpha,\beta})$ used for the encryption. As the parties only hold shares of the keys $k_{u,\alpha}$ and $k_{v,\beta}$, for a general PRF this would be inefficient. Therefore, we build F using the LPN-style "MPC-friendly" wPRF of [15]. The exact definition of F and the protocol for securely computing this wPRF are given in Section 9.

In parallel to the PRF computation, the parties compute shares of the key and external value that should be encrypted

[3] As we use a wPRF, $g_{\alpha,\beta}$ will need to be a (pseudo)-random value; see Section 9.

- The parties compute shares of $\lambda_u\lambda_v$ by invoking Π_{mult} on their shares of λ_u and λ_v. Then, using the linearity property, the parties locally compute shares of $e_w^{\alpha,\beta}$ for $\alpha, \beta \in \{0,1\}$ by the equation $e_w^{\alpha,\beta} = \lambda_w \oplus \lambda_u\lambda_v \oplus \alpha\lambda_v \oplus \beta\lambda_u \oplus \alpha\beta$.
- The parties compute shares of $e_w^{\alpha,\beta} \cdot \Delta$ for $\alpha, \beta \in \{0,1\}$ by invoking $\Pi_{mult}^{\ell(\kappa)}$ on the shares computed above and their share of Δ.[4] By locally summing these with their shares of $k_{w,0}$ they obtain shares of $k_{w,e_w^{\alpha,\beta}}$.

Following Equation (15), shares of the garbled gates are then computed by XORing the shares of the PRF with the shares of the keys and masking bits to be encrypted. The garbled gates can thus be obtained by a reconstruction round. The formal description of Protocol Π_{AND} for garbling a single AND-gate appears in Figure 3.

In the final step of Π_{offline}, parties output the garbled gates and reconstruct the output wire masks λ_{out} for every output wire *out*.

Protocol for Constructing a MYao Garbled Circuit

Input: The circuit C to be garbled.

1. Each party $\mathsf{P}_i \in \mathcal{P}$ chooses its share $\Delta^i \leftarrow \{0,1\}^{\ell(\kappa)}$ of the offset Δ. For every wire w in topological order:
   (a) if w is not an output wire of an XOR-gate, P_i samples $k_{w,0}^i \leftarrow \{0,1\}^{\ell(\kappa)}$, $\lambda_w^i \leftarrow \{0,1\}$, and computes $k_{w,1}^i = k_{w,0}^i \oplus \Delta^i$;
   (b) if w is an input wire of P_j, $j \neq i$, then P_i sets $\lambda_w^i = 0$. Thus, $\lambda_w = \lambda_w^j$.
   (c) if w is an output wire of a XOR-gate with input wires u, v, it computes $k_{w,0}^i = k_{u,0}^i \oplus k_{v,0}^i$, and $k_{w,1}^i = k_{w,0}^i \oplus \Delta^i$, $\lambda_w^i = \lambda_u^i \oplus \lambda_v^i$.
2. The parties run the protocol Π_{AND} in Fig. 3 on all the AND gates, in parallel, to compute and output all the garbled gates.[a]
3. For every output wire w of the circuit, the parties reconstruct $\lambda_w = \oplus_{i=1}^n \lambda_w^i$.

Remark: Steps 2 and 3 can be run in parallel.

[a] See remark 1 on page 31.

Fig. 2. Protocol Π_{offline}

4.3 The MYao Online Protocol

In this section, we explain the online phase protocol for evaluating a MYao garbled circuit. A formal description of the MYao online phase protocol Π_{online} is given in Figure 4.

Similarly to the BMR protocol, the first step of the online protocol involves two communication rounds, in which the parties learn the external values and corresponding keys on all the circuit input wires. This is achieved by each party

[4] As observed in [9], this part can in fact be computed using only 3 invocations of $\Pi_{mult}^{\ell(\kappa)}$.

Protocol Π_{AND} for computing a garbled AND gate (Equation (15))

Inputs: The parties hold secret-shares of the global offset Δ. For input wires u, v and output wire w of the AND gate, the parties hold secret-shares of the 0 keys $k_{u,0}, k_{v,0}$, and $k_{w,0}$, and of the masking bits λ_u, λ_v, and λ_w.
Computation: The parties compute the following 2 steps in parallel:

1. **PRF computation:** For each $\alpha, \beta \in \{0,1\}$, the parties call Π_F, using their shares of the keys $k_{u,\alpha}$ and $k_{v,\beta}$ as inputs, to obtain shares of $F_{k_{u,\alpha}}(g_{\alpha,\beta})$ and $F_{k_{v,\beta}}(g_{\alpha,\beta})$, respectively, where $g_{\alpha,\beta}$ is a unique index corresponding to the gate and the row; The details of Π_F are given in Section 9. Then, the parties locally compute shares of the value $F^2_{\alpha,\beta} = F_{k_{u,\alpha}}(g_{\alpha,\beta}) \oplus F_{k_{v,\beta}}(g_{\alpha,\beta})$.
2. **Computation of the plaintext shares:**
   (a) Parties call Π_{mult} with their shares of λ_u and λ_v to obtain shares of $\lambda_u \cdot \lambda_v$. The parties can then locally compute shares of $e_w^{\alpha,\beta} = \lambda_w \oplus (\lambda_u \oplus \alpha)(\lambda_v \oplus \beta)$.
   (b) The parties use the computed shares above and their shares of Δ as inputs to $\Pi^{\ell(\kappa)}_{mult}$ to compute shares of $e_w^{\alpha,\beta} \cdot \Delta$, then locally compute shares of $k_{w,e_w^{\alpha,\beta}} = k_{w,0} \oplus e_w^{\alpha,\beta} \cdot \Delta$ by XORing the result with their share of $k^i_{w,0}$.

Output Step: Using the above results, the parties can locally obtain shares $\tilde{g}^i_{\alpha,\beta}$ of the garbled gate $\tilde{g}_{\alpha,\beta} = F^2_{\alpha,\beta} \oplus \left(k_{w,e_w^{\alpha,\beta}} || e_w^{\alpha,\beta}\right)$. By broadcasting their shares and XORing it with the received shares, they reconstruct the garbled gate.

Fig. 3. Protocol Π_{AND}

P_i broadcasting $e_w = v_w \oplus \lambda_w$ on each wire w of its input wires, where v_w is P_i's input. Recall that the masking bit λ_w is known to P_i on its input wires. The parties then reconstruct k_{w,e_w} by broadcasting their shares and XORing.

After this step, each party can locally evaluate the circuit, similarly to the evaluator in Yao's protocol. The evaluation is gate by gate, in topological order:

- For XOR-gates, the external value and corresponding key on the output wire are computed by XORing the external values and keys on the input wires, i.e., $e_{out} = e_{in1} \oplus e_{in2}$ and $k_{out} = k_{in1} \oplus k_{in2}$.
- For AND gates, the output wire's external value and corresponding key are computed by choosing the row in the garbled gate according to the external values, and decrypting it using the keys it has of the input wires, i.e., $k_{out,e_{out}} || e_{out} = \tilde{g}_{e_{in1},e_{in2}} \oplus F_{k_{u,e_{in1}}}(g_{e_{in1},e_{in2}}) \oplus F_{k_{v,e_{in2}}}(g_{e_{in1},e_{in2}})$.

Correctness. The correctness of the MYao protocol follows from the following observation: During evaluation, at each wire w of the circuit, the external value seen by an evaluating party corresponds to $e_w = v_w \oplus \lambda_w$, where v_w is the real value on the wire (in an ungarbled circuit computation). From this it follows that for each output wire *out* of the circuit, an evaluating party correctly outputs $v_{out} = e_{out} \oplus \lambda_{out}$ at the output step of the online protocol. To show that the observation holds we proceed as follows:

1. On input wires the observation follows from Step 1a.

2. We proceed by induction on the output wires of the gates. Let g be a gate with input wires u, v and output wire w.
   - If g is an XOR gate, we observe that $e_w = v_w \oplus \lambda_w = (v_u \oplus v_v) \oplus (\lambda_u \oplus \lambda_v) = (v_u \oplus \lambda_u) \oplus (v_v \oplus \lambda_v) = e_u \oplus e_v$
   - If g is an AND gate, the observations follows by the definition of the conditional external value in Equation (16), replacing α, β with e_u, e_v, respectively (note that this is the decrypted conditional external value).

Online Protocol for Evaluating a MYao Garbled Circuit

Input: The parties receive as input the output of Π_{offline}. Additionally, each party receives its inputs x_w on each of its input wires of the circuit.

1. **Exchanging input wires' external values and corresponding keys:** for every input wire w associated with input x_w of Party P_i:
   (a) P_i sends $e_w = x_w \oplus \lambda_w$ to all the parties.
   (b) Each P_j sends its share k^j_{w,e_w} to all other parties.
   (c) Each party computes $k_{w,e_w} = \oplus_{j=1}^{n} k^j_{w,e_w}$.
2. **Local evaluation of the circuit:** The circuit is evaluated gate by gate in topological order. For a gate g with input wires u and v and output wire w, the output external value and key are computed as follows:
   (a) If g is an XOR-gate, then $e_w = e_u \oplus e_v$, and $k_{w,e_w} = k_{u,e_u} \oplus k_{v,e_v}$.
   (b) If g is an AND-gate, then $k_{w,e_w}||e_w = \tilde{g}_{e_u,e_v} \oplus F_{k_{u,e_u}}(g_{e_u,e_v}) \oplus F_{k_{v,e_v}}(g_{e_u,e_v})$

Output: For each circuit output wire *out*, output $y_{out} = e_{out} \oplus \lambda_{out}$.

Fig. 4. Protocol Π_{online}

5 Row Reduction and Half-Gates

In this section we describe how we incorporate *Half-Gates* [38] and *Row Reduction* [29] into the MYao protocol to reduce the size of the garbled circuit and optimize the offline phase. We first explain the half-gates construction, and then the required changes to the offline and online protocols. Additionally, we explain specific situations where it is possible to achieve even a double row-reduction for some or all of the gates.

Half-gates. Using half-gates requires Free-XOR. Thus, to incorporate the half-gates optimization into MYao, all that is changed is how AND gates are garbled. For an AND gate with output wire w and input wires u and v, instead of computing one four-row garbled table for an AND gate as in Equation (15), two two-row garbled tables are computed corresponding to two (non-symmetric) "half-gates".

We formally denote the two half gates by $\tilde{G}$ and $\hat{G}$, which correspond to the "garbler half gate" and "evaluator half gate" in two party terminology; note that in the multiparty setting there are no "different roles" in the computation of the

garbled half gates, but the two half-gates are not symmetric, i.e., each of the two half gates is computed differently. We denote the corresponding garbled half gates by $\tilde{g}$ and $\hat{g}$, respectively. The half-gate $\tilde{G}$ will be encrypted (and decrypted) only with the keys of input wire u (i.e., $k_{u,0}$, $k_{u,1}$), while the half-gate $\hat{G}$ will be encrypted only with the keys of input wire v (i.e., $k_{v,0}$, $k_{v,1}$). At evaluation, an evaluating party will decrypt both half gates, and use the results to obtain the external value e_w and corresponding key k_{w,e_w} on the output wire w, matching the regular protocol.

To compute the garbled gates $\tilde{g}$ and $\hat{g}$, the 0-key $k_{w,0}$ and the masking bit λ_w on the output wire w are *partitioned* to random $\tilde{k}_{w,0}, \hat{k}_{w,0} \in \{0,1\}^{\ell(\kappa)}$ and $\tilde{\lambda}_w, \hat{\lambda}_w \in \{0,1\}$, respectively, such that $k_{w,0} = \tilde{k}_{w,0} \oplus \hat{k}_{w,0}$ and $\lambda_w = \tilde{\lambda}_w \oplus \hat{\lambda}_w$. The garbled gates are then computed by the equations:

$$\tilde{g}_\alpha = F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus \left(\tilde{k}_{w,\tilde{e}_w^\alpha} || \tilde{e}_w^\alpha\right) \tag{17}$$

$$\hat{g}_\beta = F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(\left[\hat{k}_{w,\hat{e}_w^\beta} \oplus \beta \cdot k_{u,0}\right] || \hat{e}_w^\beta\right) \tag{18}$$

where $\tilde{e}_w^\alpha = \tilde{\lambda}_w \oplus (\lambda_u \oplus \alpha)\lambda_v$ and $\hat{e}_w^\beta = \hat{\lambda}_w \oplus \beta\lambda_u$. It can now be observed from Equation (16) that for the conditional external values $e_w^{\alpha,\beta}$ (in the regular garbled AND gate) it holds that

$$e_w^{\alpha,\beta} = \tilde{e}_w^\alpha \oplus \hat{e}_w^\beta \oplus \alpha\beta. \tag{19}$$

To understand why the above construction works, first observe that by Equations (17), (18), and by repeatedly using Equations (12), (19) and rearrangement:

$$\begin{aligned}
\tilde{g}_\alpha \oplus \hat{g}_\beta &= \left(F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus \left(\tilde{k}_{w,\tilde{e}_w^\alpha} || \tilde{e}_w^\alpha\right)\right) \oplus \left(F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(\left[\hat{k}_{w,\hat{e}_w^\beta} \oplus \beta \cdot k_{u,0}\right] || \hat{e}_w^\beta\right)\right) \\
&= F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus F_{k_{v,\beta}}(g_{1,\beta}) \oplus \\
&\qquad \oplus \left(\left[\left(\tilde{k}_{w,0} \oplus \hat{k}_{w,0}\right) \oplus (\tilde{e}_w^\alpha \oplus \hat{e}_w^\beta) \cdot \Delta \oplus \beta \cdot k_{u,0}\right] || \left[\tilde{e}_w^\alpha \oplus \hat{e}_w^\beta\right]\right) \\
&= F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(\left[k_{w,0} \oplus \beta \cdot k_{u,0} \oplus (e_w^{\alpha,\beta} \oplus \alpha\beta) \cdot \Delta\right] || \left[e_w^{\alpha,\beta} \oplus \alpha\beta\right]\right) \\
&= F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(k_{w,0} \oplus e_w^{\alpha,\beta} \cdot \Delta || e_w^{\alpha,\beta}\right) \oplus \beta \cdot (k_{u,0} \oplus \alpha \cdot \Delta || \alpha) \\
&= F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(k_{w,e_w^{\alpha,\beta}} || e_w^{\alpha,\beta}\right) \oplus \beta \cdot (k_{u,\alpha} || \alpha).
\end{aligned}$$

Thus, for every $\alpha, \beta \in \{0,1\}$, it holds that

$$(k_{w,e_w^{\alpha,\beta}} || e_w^{\alpha,\beta}) = \tilde{g}_\alpha \oplus \hat{g}_\beta \oplus F_{k_{u,\alpha}}(g_{0,\alpha}) \oplus F_{k_{v,\beta}}(g_{1,\beta}) \oplus \beta \cdot (k_{u,\alpha} || \alpha). \tag{20}$$

Hence, at the online phase, an evaluating party holding external values e_u, e_v and corresponding keys k_{u,e_u}, k_{v,e_v}, decrypts $\tilde{g}_{e_u}$ and $\hat{g}_{e_v}$ by XORing with $F_{k_{u,e_u}}(g_{0,e_u})$ and $F_{k_{v,e_v}}(g_{1,e_v})$, respectively. Then, the evaluating party XORs the decrypted results along with $e_v \cdot (k_{u,e_u} || e_u)$. By Equation (20), this results in $k_{w,e_w} || e_w$ for any $e_v, e_u \in \{0,1\}$, matching the expected result.

So far, we have not made any saving to the garbled circuit size, as the two half-gates require 2 garbled rows each, so 4 garbled rows in total for an AND

gate. However, we first observe that in MYao, using half-gates is beneficial even without row reduction: the number of PRF invocations required for computing a garbled half-gate based AND gate is 4, whereas computing the standard garbled AND gate requires 8 invocations of the PRF. The difference from standard Yao is that we are not using a 2-keyed PRF/Encryption. Furthermore, in MYao, the online time for half-gate based AND is only marginally more than standard AND, as in both cases decrypting an AND gate requires 2 local PRF invocations, which is the overwhelmingly dominant factor of the online computation.

We next explain how the number of rows is reduced to 3 using row reduction.

Row reduction via half gates. To achieve row reduction, the first row of the garbled half-gate $\tilde{g}$ is set to be the default value 0, i.e., $\tilde{g}_0 = 0$. By Equation (17) it follows that

$$F_{k_{u,0}}(g_{0,0}) = \tilde{k}_{w,\tilde{e}^0_w} || \tilde{e}^0_w. \tag{21}$$

Hence, $\tilde{e}^0_w$ and $\tilde{k}_{w,0}$ are now decided by $F_{k_{u,0}}(g_{0,0})$ and the global offset Δ. Similarly, other parts of the partitioned output key and conditional external value are decided by the above and the 0-key $k_{w,0}$ and masking bit λ_w on output wire w of the AND gate:

$$\hat{e}^0_w = \lambda_w \oplus \lambda_u\lambda_v \oplus \tilde{e}^0_w; \quad \tilde{e}^1_w = \tilde{e}^0_w \oplus \lambda_v; \quad \hat{e}^1_w = \hat{e}^0_w \oplus \lambda_u; \tag{22}$$

$$\hat{k}_{w,0} = k_{w,0} \oplus \tilde{k}_{w,0}. \tag{23}$$

It might be tempting to try to get a double row reduction in MYao, by setting the first row of the garbled gate $\hat{g}$ to the default value as well. However, this turns out to create complications for constructing a constant round MPC protocol; We discuss this at the end of this section and in Section 8.

The offline functionality for MYao with row reduction via half-gates, $\mathcal{F}^{HG}_{garble}$, is identical to the basic MYao functionality $\mathcal{F}_{\text{garble}}$ apart from Step 4, i.e., computing the garbled gates. Thus, we give only the alternative Step 4 of $\mathcal{F}^{HG}_{garble}$ in Figure 5. Observe that in Figure 5, $\tilde{g}_1$ appears instead of $\tilde{g}_\alpha$ – this is because for $\alpha = 0$ we have $\tilde{g}_\alpha = \tilde{g}_0 = 0$ due to the row reduction, and so we omit it.

The offline protocol with half gates and row reduction. The offline protocol computing a MYao garbled circuit with the half gates and row reduction optimizations differs from the basic MYao offline protocol only in Step 2, i.e., securely computing the garbled gates.

To securely compute the garbled gates, parties first compute in parallel:

1. Shares of the outputs of PRF, i.e., of the values $F_{k_{u,\alpha}}(g_{0,\alpha})$ and $F_{k_{v,\beta}}(g_{1,\beta})$ for $\alpha, \beta \in \{0,1\}$
2. Shares of $\lambda_u\lambda_v$ using their shares of λ_u and λ_v on Π_{mult}. Then, using additionally their shares of Δ, the parties compute by invoking $\Pi^{\ell(\kappa)}_{mult}$ shares of the multiplications: $\lambda_w \cdot \Delta$, $\lambda_u \cdot \Delta$, $\lambda_v \cdot \Delta$, and $(\lambda_u\lambda_v) \cdot \Delta$

From the first computation above, the parties parse the result as $(\tilde{k}^i_{w,\tilde{e}^0_w} || \tilde{e}^{0,i}_w)$, where $\tilde{k}^i_{w,\tilde{e}^0_w} \in \{0,1\}^{\ell(\kappa)}$, $\tilde{e}^{0,i}_w \in \{0,1\}$, i.e., their shares of $\tilde{e}^0_w$ and $\tilde{k}_{w,\tilde{e}^0_w}$ are

Functionality for MYao Garbled Circuit with Half-Gates

4. For every AND-gate with input wires u, v and output wire w, compute the garbled half-gates $\tilde{g}_1$ and $\hat{g}_\beta$ for $\beta \in \{0,1\}$ as follows.
   (a) Compute $F_{k_{u,0}}(g_{0,0})$ and parse it as $(\tilde{k}_{w,\tilde{e}^0_w}||\tilde{e}^0_w)$, where $\tilde{k}_{w,\tilde{e}^0_w} \in \{0,1\}^{\ell(\kappa)}$, $\tilde{e}^0_w \in \{0,1\}$.
   (b) Compute the partition of the external values $\tilde{e}^1_w = \tilde{e}^0_w \oplus \lambda_v$, $\hat{e}^0_w = \tilde{e}^0_w \oplus \lambda_w \oplus \lambda_u\lambda_v$, $\hat{e}^1_w = \hat{e}^0_w \oplus \lambda_u$, and of the keys $\tilde{k}_{w,0} = \tilde{k}_{w,\tilde{e}^0_w} \oplus \tilde{e}^0_w \cdot \Delta$, $\tilde{k}_{w,1} = \tilde{k}_{w,0} \oplus \Delta$, $\hat{k}_{w,0} = k_{w,0} \oplus \tilde{k}_{w,0}$, $\hat{k}_{w,1} = \hat{k}_{w,0} \oplus \Delta$.
   (c) Compute and output to the parties the reduced half gates

$$\begin{aligned} \tilde{g}_1 &= F_{k_{u,1}}(g_{0,1}) \oplus \left(\tilde{k}_{w,\tilde{e}^1_w}||\tilde{e}^1_w\right); \\ \hat{g}_\beta &= F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(\hat{k}_{w,\hat{e}^\beta_w} \oplus \beta \cdot k_{u,0}||\hat{e}^\beta_w\right), \end{aligned} \tag{24}$$

   where $\beta \in \{0,1\}$.

Remark: All other steps are identical to Functionality $\mathcal{F}_{\text{garble}}$. The change to the output is that the garbled gates are different (garbled half-gate AND gates instead of regular garbled AND gates).

Fig. 5. Functionality $\mathcal{F}^{HG}_{garble}$

determined by this computation. Notice that by Equation (21), this implies that $(\tilde{k}_{w,\tilde{e}^0_w}||\tilde{e}^0_w) = F_{k_{u,0}}(g_{0,0})$.

The rest of the required shares for the garbled gates, i.e., the shares of $\tilde{e}^1_w$, $\hat{e}^0_w$, $\hat{e}^1_w$, $\tilde{k}_{w,\tilde{e}^1_w}$, $\hat{k}_{w,\hat{e}^0_w}$, and $\hat{k}_{w,\hat{e}^1_w}$ can now be computed by linear combinations of the shares they already have, using Equations (22), (23), and (12). E.g.,

$$\tilde{k}_{w,\tilde{e}^1_w} = \tilde{k}_{w,0} \oplus \tilde{e}^1_w \cdot \Delta = \tilde{k}_{w,0} \oplus (\tilde{e}^0_w \oplus \lambda_v) \cdot \Delta = \tilde{k}_{w,\tilde{e}^0_w} \oplus \lambda_v \cdot \Delta \tag{25}$$

$$\begin{aligned} \hat{k}_{w,\hat{e}^0_w} = \hat{k}_{w,0} \oplus \hat{e}^0_w \cdot \Delta &= k_{w,0} \oplus \tilde{k}_{w,0} \oplus (\lambda_w \oplus \lambda_u\lambda_v \oplus \tilde{e}^0_w) \cdot \Delta \\ &= k_{w,0} \oplus \tilde{k}_{w,\tilde{e}^0_w} \oplus (\lambda_w \oplus \lambda_u\lambda_v) \cdot \Delta \end{aligned} \tag{26}$$

and

$$\begin{aligned} \hat{k}_{w,\hat{e}^1_w} = \hat{k}_{w,0} \oplus k_{u,0} \oplus \hat{e}^1_w \cdot \Delta &= \hat{k}_{w,0} \oplus k_{u,0} \oplus (\hat{e}^0_w \oplus \lambda_u) \cdot \Delta \\ &= \hat{k}_{w,\hat{e}^0_w} \oplus k_{u,0} \oplus \lambda_u \cdot \Delta. \end{aligned} \tag{27}$$

Notice that this avoids multiplying $\tilde{e}^1_w$, $\hat{e}^0_w$, $\hat{e}^1_w$ with Δ directly, which would require adding another multiplication round, as it cannot be computed in parallel with Step 1a. The parties can now recover the garbled gates $\tilde{g}_1$ and $\hat{g}_\beta$ for $\beta \in \{0,1\}$ by a reconstruction round.

The online protocol. The online phase of MYao with half-gates and row-reduction is the same as the online phase for basic MYao, except for Step 2b, i.e., the evaluation of the garbled half-gate based AND gates replaces the evaluation of regular garbled AND gates.

Protocol Π_{AND}^{HG} for computing the garbled half-gate with a row reduction.

Inputs: The private inputs of each party P_i are same as for Π_{AND}.
Computation:

1. The parties perform the following two steps in parallel:
   (a) The parties call the multiparty protocol Π_F four times in parallel to compute shares $\tilde{F}_\alpha^i$, $\hat{F}_\beta^i$ of four values $\tilde{F}_\alpha = F_{k_{u,\alpha}}(g_{0,\alpha})$, $\hat{F}_\beta = F_{k_{v,\beta}}(g_{1,\beta})$ as PRF's on the secret-shared keys $k_{u,\alpha}$, $k_{v,\beta}$ of the public inputs (for every $\alpha,\ \beta \in \{0,1\}$). Each P_i parses $\tilde{F}_0^i$ as $(\tilde{k}^i_{w,\tilde{e}^0_w} || \tilde{e}^{0,i}_w)$, where $\tilde{k}^i_{w,\tilde{e}^0_w} \in \{0,1\}^{\ell(\kappa)}$, $\tilde{e}^{0,i}_w \in \{0,1\}$.
   (b) Using the multiplication protocols Π_{mult} and $\Pi_{mult}^{\ell(\kappa)}$ the parties:
      i. The parties call Π_{mult} on their inputs λ_u^i and λ_v^i to obtain λ_{uv}^i.
      ii. In parallel, they call three instances of $\Pi_{mult}^{\ell(\kappa)}$. The first input of P_i to each instance is Δ^i, and the second are λ_u^i, λ_v^i, and $(\lambda_w^i \oplus \lambda_{uv}^i)$. The outputs are the shares Λ_u^i, Λ_v^i and Λ_*^i, respectively. I.e., Λ_u^i, Λ_v^i and Λ_*^i are their shares of $\lambda_u \cdot \Delta$, $\lambda_v \cdot \Delta$, and $(\lambda_w \oplus \lambda_u\lambda_v) \cdot \Delta$ respectively.
2. **Computation of the outputs' shares (local):** Each P_i locally computes the following shares: $\tilde{e}_w^{1,i} = \tilde{e}_w^{0,i} \oplus \lambda_v^i$, $\hat{e}_w^{0,i} = \tilde{e}_w^{0,i} \oplus \lambda_w^i \oplus \lambda_{uv}^i$, $\hat{e}_w^{1,i} = \hat{e}_w^{0,i} \oplus \lambda_u^i$, $\tilde{k}^i_{w,\tilde{e}^1_w} = \tilde{k}^i_{w,\tilde{e}^0_w} \oplus \Lambda_v^i$, $\hat{k}^i_{w,\hat{e}^0_w} = k^i_{w,0} \oplus \tilde{k}^i_{w,\tilde{e}^0_w} \oplus \Lambda_*^i$, $\hat{k}^i_{w,\hat{e}^1_w} = k^i_{u,0} \oplus \hat{k}^i_{w,\hat{e}^0_w} \oplus \Lambda_u^i$.

Output Reconstruction: Each P_i locally computes $\tilde{g}_1^i = \tilde{F}_1^i \oplus \left(\tilde{k}^i_{w,\tilde{e}^1_w} || \tilde{e}^{1,i}_w\right)$, and $\hat{g}_\beta^i = \hat{F}_\beta^i \oplus \left(\hat{k}^i_{w,\hat{e}^\beta_w} || \hat{e}^{\beta,i}_w\right)$ for every $\beta \in \{0,1\}$, and sends them to any P_j, $j \neq i$.[a]
Then, each party locally reconstructs the garbled half gates as $\tilde{g}_1 = \bigoplus_{i=1}^n \tilde{g}_1^i$, and $\hat{g}_\beta = \bigoplus_{i=1}^n \hat{g}_\beta^i$ for $\beta \in \{0,1\}$.

[a] See remark 1 on page 31.

Fig. 6. Protocol Π_{AND}^{HG}

As explained, to evaluate the garbled half-gates based AND, an evaluating party decrypts both half gates according to the external values on the input wires and the corresponding keys. When using row-reduction, if $e_u = 0$, the evaluating party uses $\tilde{g}_0 = 0$ as the garbled half-gate. The key and external value on the AND output wire is then computed as the XOR of both values XORed with $e_v \cdot (k_{u,e_u} || e_u)$ (which the evaluating party can compute at this stage). The formal description of Step 2b in the online protocol with half-gates is formally given in Figure 7.

Half Gate AND Evaluation Protocol

2.(b) For an AND gate g with input wires u and v and output wire w, the output external value and key are computed as follows:

$$k_{w,e_w} || e_w = \tilde{g}_{e_u} \oplus F_{k_{u,e_u}}(g_{0,e_u}) \oplus \hat{g}_{e_v} \oplus F_{k_{v,e_v}}(g_{1,e_v}) \oplus e_v \cdot (k_{u,e_u} || e_u).$$

Fig. 7. Replacement Step 2b in Π_{online}^{HG}

Double row reduction. It is also possible to apply row reduction to both halves, by setting both $\hat{g}_0 = 0$ and $\tilde{g}_0 = 0$. This would reduce the garbled AND gate size down to only two rows, one from each half. However, as a result, the AND gate's output wire's keys and masks become dependent on those of its input wires. Hence, using a double row-reduction seems to require computing the AND gates sequentially (in layers). The resulting protocol has number of rounds linear in the depth of the circuit. We give the details in Section 8. Notice, however, that the online phase remains constant round, with only two rounds of interaction.

Additionally, we observe that in certain circuits it is possible to achieve double row-reduction in constant rounds for some or even the majority of the gates. For example, in layered circuits, where the AND gates are layered and outputs of AND gates of layer i are used as inputs only in layer $i+1$. Due to free-XOR, we also require that all XOR gate chains have inputs from some layer i and output to layer $i+1$. We give the details of this construction, as well as generalizations, in Section 8.

6 Proof of Security

In this section, we prove the security of **MYao**. First, we prove that the protocol Π_{offline} securely computes the "Yao"'s circuit, i.e. securely realizes the functionality $\mathcal{F}_{\text{garble}}$ in presence of a semi-honest adversary corrupting any number of parties from $\mathcal{P}$. Then, we prove security of **MYao** in the $\mathcal{F}_{\text{garble}}$-hybrid model assuming that the PRF F is *circular correlation robust*, according to Definition 1.

In this section, by $\mathcal{P}_A, \mathcal{P}_H \subset \mathcal{P}$ we denote the set of corrupt and honest parties respectively, and by $A, H \subset [n]$ sets of their indexes.

6.1 Security of the Garbling

In this section we prove the security of the garbling process. We assume the following ideal functionalities: $\mathcal{F}_F$ for a secure multiparty evaluation of the PRF F, $f_{mult}^{\ell(\kappa)}$ for a multiparty bit-string multiplication for $\ell(\kappa)$-long strings, and f_{mult} for the multiparty bit-bit multiplication. These functionalities are securely realized by the protocols Π_F, $\Pi_{mult}^{\ell(\kappa)}$, and Π_{mult} respectively in the presence of a static semi-honest adversary corrupting any number of parties. Each of these functionalities takes the XOR secret shared inputs from the parties in $\mathcal{P}$, and gives the XOR secret shared outputs from the correspondent function.

The multiplication protocols and functionalities are given in Section 2.1. Protocol Π_F is given in Appendix A. For completeness, we give the gate encryption functionality $\mathcal{F}_{\text{AND}}$ in Figure 8.

Lemma 1. *The protocol Π_{AND} securely computes the n-party functionality $\mathcal{F}_{\text{AND}}$ in the $(\mathcal{F}_F, f_{mult}^{\ell(\kappa)}, f_{mult})$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Functionality $\mathcal{F}_{\text{AND}}$ for Encrypting an AND-gate

Inputs: Each party P_i inputs $k^i_{u,0}$, $k^i_{v,0}$, $k^i_{w,0}$, $\Delta^i \in \{0,1\}^{\ell(\kappa)}$, $\lambda^i_u, \lambda^i_v, \lambda^i_w \in \{0,1\}$.
Computation:

1. The functionality reconstructs the wires' keys and masks, and the global offset as follows: $\Delta = \Delta^1 \oplus \cdots \oplus \Delta^n$, $k_{u,0} = k^1_{u,0} \oplus \cdots \oplus k^n_{u,0}$, $k_{v,0} = k^1_{v,0} \oplus \cdots \oplus k^n_{v,0}$, $k_{w,0} = k^1_{w,0} \oplus \cdots \oplus k^n_{w,0}$, $k_{u,1} = k_{u,0} \oplus \Delta$, $k_{v,1} = k_{v,0} \oplus \Delta$, $\lambda_u = \lambda^1_u \oplus \cdots \oplus \lambda^n_u$, $\lambda_v = \lambda^1_v \oplus \cdots \oplus \lambda^n_v$, $\lambda_w = \lambda^1_w \oplus \cdots \oplus \lambda^n_w$.
2. For every α, $\beta \in \{0,1\}$, compute the conditional external value $e^{\alpha,\beta}_w = \lambda_w \oplus (\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta)$, and the garbled row:

$$\tilde{g}_{\alpha,\beta} = F_{k_{u,\alpha}}(g_{\alpha,\beta}) \oplus F_{k_{v,\beta}}(g_{\alpha,\beta}) \oplus \left[\left(k_{w,0} \oplus (e^{\alpha,\beta}_w \cdot \Delta)\right) || e^{\alpha,\beta}_w\right], \tag{28}$$

   where $g_{\alpha,\beta}$ are unique public indexes correspondent to the AND-gate g's rows, and F is a PRF.

Outputs: The functionality outputs $\{\tilde{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ to every party P_i.

Fig. 8. Functionality $\mathcal{F}_{\text{AND}}$

Proof. The simulator $\mathcal{S}$, in Step 1, simulates all the outputs of $\mathcal{F}_F$ by sending to the adversary the uniformly random strings $F^i_{u,\alpha,\beta}$ and $F^i_{v,\alpha,\beta}$ of length $\ell(\kappa)$ for every $\mathsf{P}_i \in \mathcal{P}_A$, and every $\alpha, \beta \in \{0,1\}$.

In Step 2(a), it simulates the output of f_{mult} sending a uniformly random bit as the share of $\lambda_u\lambda_v$ of any $\mathsf{P}_i \in \mathcal{P}_A$. In Step 2(b), $\mathcal{S}$ simulates the outputs of $f^{\ell(\kappa)}_{mult}$ sending the uniformly random binary strings $d^i_{\alpha,\beta}$ of length $\ell(\kappa)$ as the shares of $e^{\alpha,\beta}_w \cdot \Delta$ for every $\mathsf{P}_i \in \mathcal{P}_A$, and every $\alpha, \beta \in \{0,1\}$.

Finally, in the output step, for every $\tilde{g}_{\alpha,\beta}$ obtained from the ideal functionality $\mathcal{F}_{\text{AND}}$, the simulator computes

$$\tilde{g}^A_{\alpha,\beta} = \tilde{g}_{\alpha,\beta} \oplus \bigoplus_{i\in A} \left(F^i_{u,\alpha,\beta} \oplus F^i_{v,\alpha,\beta} \oplus (k^i_{w,0} \oplus d^i_{\alpha,\beta} || e^{\alpha,\beta,i}_w)\right),$$

XOR-shares it for all $\mathsf{P}_j \in \mathcal{P}_H$, and broadcast on their behalf.

In both real and ideal worlds the view of the adversary together with the output of honest parties is distributed identically, hence the protocol Π_{AND} is secure.

Lemma 2. *The protocol Π_{offline} securely realizes the n-party functionality $\mathcal{F}_{\text{garble}}$ in the $\mathcal{F}_{\text{AND}}$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Proof. The simulator $\mathcal{S}$, getting from the ideal functionality $\mathcal{F}_{\text{garble}}$ the outputs $\{\tilde{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ for all AND gates, Δ^i, and $k^i_{w,0}$ for all input wires w, λ_{out} on each output wire *out*, for all $\mathsf{P}_i \in \mathcal{P}_A$, simulates the view of the adversary as follows. First, it sets the randomness of all corrupt parties P_i according to the $\mathcal{F}_{\text{garble}}$'s choice of Δ^i, $k^i_{w,0}$, and λ^i_{out}. Notice, that, as the adversary is semi-honest, in the real world parties would choose them uniformly at random, as $\mathcal{F}_{\text{garble}}$ does in the ideal world.

The first step of the protocol is local, thus the simulator doesn't send any messages. In the second step of the simulation, $\mathcal{S}$ simulates the output of $\mathcal{F}_{\text{AND}}$ sending $\{\tilde{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ that it got from $\mathcal{F}_{\text{garble}}$ to the adversary as the output of any corrupt party. In the third step, the simulator XOR-shares the bit $\left(\lambda_{out} \oplus \bigoplus_{i\in A} \lambda^i_{out}\right)$ in $|\mathcal{P}_H|$ shares, and broadcasts them on behalf of the parties from $\mathcal{P}_H$.

In both real and ideal worlds the view of the adversary together with the output of honest parties is distributed identically.

Theorem 1. *The protocol Π_{offline} securely realizes the n-party functionality $\mathcal{F}_{\text{garble}}$ in the $(\mathcal{F}_F, f^{\ell(\kappa)}_{mult}, f_{mult})$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Proof. By composition of Lemma 2 and Lemma 1.

6.2 Security of MYao

We denote by $\mathcal{F}_C$ the function computed by a circuit C. We limit the circuit to only the basis of XOR, AND and NOT gates with at most 2 input wires. Also, we require that any gate's input wires are not duplicated, as allowing such duplications introduces a vulnerability for the garbled circuit in general, as was shown by Nieminen and Schneider [30], and in particular for our construction of the two-keyed PRF. This requirement does not easily imposed, as for any bit b, it holds that $b \oplus b = 0$, and $b \cdot b = b$, thus the gates with duplicated inputs can simply be omitted.

For simplicity, we assume that any $\mathsf{P}_i \in \mathcal{P}$ has only a single input bit x_i, and gets the output from the only output wire *out* of C. We'll prove the security of **MYao** in the $\mathcal{F}_{\text{garble}}$-hybrid model. In the real world, the view of the corrupt parties are:

1. The output of $\mathcal{F}_{\text{garble}}$, i.e. the garbled circuit $\{\tilde{g}_{\alpha,\beta}\}_{g\in C,\alpha,\beta\in\{0,1\}}$, the global offset's shares Δ^i, the input wire's keys shares $k^i_{w,0}$, the wire mask λ_{in} for any P_i's input wire, for any $i \in A$ and the output wire's mask λ_{out};
2. The messages of the online round, i.e. the input wire's external values e_w and the correspondent keys' shares k^j_{w,e_w} for $j \in H$.

In the ideal world, the simulator receives $y_{out} = f(x_1, \ldots, x_n)$ from the trusted party, but learns only inputs of parties from $\mathcal{P}_A$. Without knowing inputs of honest parties, it cannot construct the real garbled circuit together with appropriate λ_{out} and input external values. The simulator builds the fake garbled circuit instead, with fake external values and keys, and fake output wire mask. In every fake garbled table, only a single row-on-the-evaluation-path is computed by encryption. Other three rows are chosen by a simulator uniformly at random. The proof of security, thus, is to show that no PPT distinguisher $\mathcal{D}$ can distinguish between the real garbled circuit and the fake one.

The simulator $\mathcal{S}_{\textbf{MYao}}$ is given in Figure 9.

Simulator Algorithm $\mathcal{S}_{\mathbf{MYao}}$

Inputs: The circuit C and the inputs for the corrupted parties $\{x_i\}_{i\in A}$.
Algorithm:

1. The simulator $\mathcal{S}_{\mathbf{MYao}}$ sends $\{x_i\}_{i\in A}$ to the trusted party, and obtains $\mathcal{F}_C(x_1,\ldots,x_n)$.
2. For any $i \in A$, $\mathcal{S}_{\mathbf{MYao}}$ performs the following computations:
   (a) Sample the offset $\hat{\Delta}^i \leftarrow \{0,1\}^{\ell(\kappa)}$;
   (b) For each wire w which is not the output of an XOR-gate, sample $\hat{k}^i_{w,0} \leftarrow \{0,1\}^{\ell(\kappa)}$, and set $\hat{k}^i_{w,1} = \hat{k}^i_{w,0} \oplus \hat{\Delta}^i$.
   (c) In topological order, for any XOR-gate with input wires u and v and output w, compute $\hat{k}^i_{w,0} = \hat{k}^i_{u,0} \oplus \hat{k}^i_{v,0}$, $\hat{k}^i_{w,1} = \hat{k}^i_{w,0} \oplus \hat{\Delta}^i$.
3. For each wire w which is not the output of an XOR-gate, sample the external value $\hat{e}_w \leftarrow \{0,1\}$, for any XOR-gate with input wires u and v, compute in topological order $\hat{e}_w = \hat{e}_u \oplus \hat{e}_v$.
4. For any $j \in H$:
   (a) For each wire w which is not the output of an XOR-gate, sample $\hat{k}^j_{w,\hat{e}_w} \leftarrow \{0,1\}^{\ell(\kappa)}$.
   (b) In topological order, for any XOR-gate with input wires u and v and output w, compute $\hat{k}^j_{w,\hat{e}_w} = \hat{k}^j_{u,\hat{e}_u} \oplus \hat{k}^j_{v,\hat{e}_v}$.
5. For each $i \in A$, if w is the input wire of P_i, set $\hat{\lambda}_w = \hat{\lambda}^i_w = x_i \oplus \hat{e}_w$.
6. For an output wire, set $\hat{\lambda}_{out} = f(x_1,...,x_n) \oplus \hat{e}_{out}$.
7. For each AND-gate with input wires u and v and output wire w, generate a fake garbled gate $\{\hat{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ as follows:
   (a) Reconstruct the keys: $\hat{k}_{u,\hat{e}_u} = \bigoplus_{i=1}^n \hat{k}^i_{u,\hat{e}_u}$, $\hat{k}_{v,\hat{e}_v} = \bigoplus_{i=1}^n \hat{k}^i_{v,\hat{e}_v}$, $\hat{k}_{w,\hat{e}_w} = \bigoplus_{i=1}^n \hat{k}^i_{w,\hat{e}_w}$.
   (b) Set $\hat{g}_{\hat{e}_u,\hat{e}_v} = F_{\hat{k}_{u,\hat{e}_u}}(g_{\hat{e}_u,\hat{e}_v}) \oplus F_{\hat{k}_{v,\hat{e}_v}}(g_{\hat{e}_u,\hat{e}_v}) \oplus \left(\hat{k}_{w,\hat{e}_w} || \hat{e}_w\right)$.
   (c) For $(\alpha,\beta) \neq (\hat{e}_u,\hat{e}_v)$, sample $\hat{g}_{\alpha,\beta} \leftarrow \{0,1\}^{\ell(\kappa)+1}$.

Outputs (Simulation of the real-world MYao protocol):

1. The simulator $\mathcal{S}_{\mathbf{MYao}}$ simulates the output of $\mathcal{F}_{\text{garble}}$ to the corrupt parties by sending the fake garbled circuit $\{\hat{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$, together with the corrupted parties' randomness: $\hat{\Delta}^i$, $\hat{k}^i_{w,0}$ for any input wire w of the circuit, $\hat{\lambda}_{in_i}$ for any P_i's input wire in_i, for any $i \in A$, and $\hat{\lambda}_{out}$.
2. The simulator $\mathcal{S}_{\mathbf{MYao}}$ simulates the messages of the honest parties in the first step of Π_{online}: external value $\hat{e}_{in_j}$ for all input wires of $\mathsf{P}_j \in \mathcal{P}_H$, and input key shares $\hat{k}^j_{w,\hat{e}_w}$ for any input wire w of the circuit, and all $j \in H$.

Fig. 9. Algorithm $\mathcal{S}_{\mathbf{MYao}}$ for generating a fake garbled circuit and simulation of a real-world **MYao** protocol

Theorem 2. *Let C be some circuit, and denote by $\mathcal{F}_C$ the n-party functionality computing it. Assuming F is circular correlation robust, the protocol Π_{online} securely realizes the n-party functionality $\mathcal{F}_C$ in the $\mathcal{F}_{\text{garble}}$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Proof. The output of the protocol is fully determined by the inputs $x_1, \ldots, x_n$ and equal in both worlds. Thus, in order to prove the semi-honest security of **MYao**, we need to prove that for every input $x_1, \ldots, x_n$, the ideal-world view of the corrupt parties simulated by $\mathcal{S}_{\mathbf{MYao}}$ (Figure 9) is computationally indistinguishable from the real-view of these parties in an execution of the protocol in $\mathcal{F}_{\text{garble}}$-hybrid model.

The real-world view of the corrupt parties includes the garbled circuit $\{\tilde{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$, and, additionally, the set $\left\{\{\Delta^i, \{k^i_{w,0}\}_w, \lambda_{in_i}\}_{i\in A}, \lambda_{out}, \left\{e_{in_j}, \{k^j_{w,e_w}\}_w\right\}_{j\in H}\right\}$, where w are input wires of the circuit, and in_i is the input wire, which belongs to P_i. The latter is simulated by $\mathcal{S}_{\mathbf{MYao}}$ by the fake garbled circuit $\{\hat{g}_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ together with the set $\left\{\left\{\tilde{\Delta}^i, \{\tilde{k}^i_{w,0}\}_w, \tilde{\lambda}_{in_i}\right\}_{i\in A}, \tilde{\lambda}_{out}, \left\{\tilde{e}_{in_j}, \{\tilde{k}^j_w, \tilde{e}_w\}_w\right\}_{j\in H}\right\}$. Both sets are the sets of a uniformly random strings of the appropriate length, thus it remains to prove that the real and fake garbled circuits are indistinguishable for a PPT adversary.

We prove that the real-world and the ideal-world are indistinguishable similarly to [12, Thm. 3]. Namely, we assume towards a contradiction that there exists a PPT $\mathcal{D}_{\mathbf{MYao}}$ (a distinguisher), which for some fixed inputs $(x_1, \ldots, x_n)$ can distinguish between the ideal-world view of the adversary, created by the simulator $\mathcal{S}_{\mathbf{MYao}}$ with the fake garbled circuit, and real-world view of the adversary consisting of the real garbled circuit. To obtain a contradiction, we construct the PPT distinguisher $\mathcal{D}_{CCR}$ (Figure 10) that distinguishes between the oracles $Circ^F_{\tilde{\Delta}}$ and $Rand$ (defined in Section 3.1) with non-negligible probability, and thus, breaks the circular correlation robustness of the wPRF F.

The distinguisher $\mathcal{D}_{CCR}$ is given access to an oracle $\mathcal{O} \in \left\{Circ^F_{\tilde{\Delta}}, Rand\right\}$ and constructs an adversarial view, using only legal oracle queries as shown in Figure 10. In every garbled gate with the input wires u and v and the output wire w, $\mathcal{D}_{CCR}$ chooses one row indexed by the external values e_u and e_v to encrypt by calling F. For other rows indexed by $(\alpha, \beta) \neq (e_u, e_v)$, it makes queries to the oracle $\mathcal{O}$ as follows:

- if $\alpha = e_u \oplus 1$, $\beta = e_v$, it queries for $O^{g_{\alpha,\beta}}_{\alpha} = \mathcal{O}(k_{u,e_u}, g_{\alpha,\beta}, b_{\alpha,\beta})$;
- if $\alpha = e_u$, $\beta = e_v \oplus 1$, it queries for $O^{g_{\alpha,\beta}}_{\beta} = \mathcal{O}(k_{v,e_v}, g_{\alpha,\beta}, b_{\alpha,\beta})$;
- if $\alpha = e_u \oplus 1$, $\beta = e_v \oplus 1$, it queries for both $O^{g_{\alpha,\beta}}_{\alpha} = \mathcal{O}(k_{u,e_u}, g_{\alpha,\beta}, b_{\alpha,\beta})$, and $O^{g_{\alpha,\beta}}_{\beta} = \mathcal{O}(k_{v,e_v}, g_{\alpha,\beta}, 0)$.

Notice that as $g_{\alpha,\beta}$ is unique for any garbled table row in the circuit, and there are no gates with duplicated inputs, each query of the form $(k, g, \cdot)$ is made just once for any pair (k, g), hence all queries are legal.[5]

The bit $b_{\alpha,\beta}$ is 0 if, in a real garbled circuit, the rows (α, β) and (e_u, e_v) encrypt the same output key. Otherwise, it is 1. In other words, $b_{\alpha,\beta}$=0, iff

[5] If $g_{\alpha,\beta}$ are created by the pseudorandom hash-function from the counter, the collision is possible with the negligible probability.

Distinguisher Algorithm $\mathcal{D}_{CCR}$

Inputs: The circuit C, a fixing of parties' inputs $x_1 \dots x_n$.

1. **The wires' values extraction:** In a topological order, for every wire w compute the true value v_w.
2. **Setting the randomness for corrupt parties:** For any $\mathsf{P}_i \in \mathcal{P}_A$, sample $\Delta^i \leftarrow \{0,1\}^{\ell(\kappa)}$ and, for each wire w which is not the output of a XOR-gate, sample $k^i_{w,0} \leftarrow \{0,1\}^{\ell(\kappa)}$.
3. **Computing the keys for corrupt parties:** For any $\mathsf{P}_i \in \mathcal{P}_A$, and each wire w, in topological order, if w is the output of an XOR-gate with the input wires u, v, compute $k^i_{w,0} = k^i_{u,0} \oplus k^i_{v,0}$. Additionally, for each wire w, compute $k^i_{w,1} = k^i_{w,0} \oplus \Delta^i$.
4. **Computing the keys and external values for the circuit:**
   (a) For each wire w which is not the output of an XOR-gate, sample $e_w \leftarrow \{0,1\}$; for each w which is the output of an XOR-gate with the input wires u, v, compute $e_w = e_u \oplus e_v$. If w is an input wire of a corrupt P_i, compute $\lambda_w = e_w \oplus x_i$.
   (b) For each wire w which is not the output of an XOR-gate, and for each $\mathsf{P}_j \in \mathcal{P}_H$, sample $k^j_{w,e_w} \leftarrow \{0,1\}^{\ell(\kappa)}$.
   (c) For each wire w which is the output of an XOR-gate with the input wires u, v, and for each $\mathsf{P}_j \in \mathcal{P}_H$, compute $k^j_{w,e_w} = k^j_{u,e_u} \oplus k^j_{v,e_v}$.
   (d) For each wire w, compute $k_{w,e_w} = \bigoplus_{i=1}^{n} k^i_{w,e_w}$.
5. **Computing the circuit:** For every AND-gate g with the input wires u and v, and the output wire w, sample the uniformly random indexes $g_{\alpha,\beta}$ for $\alpha, \beta \in \{0,1\}$, and compute four garbled rows:
   (a) $\tilde{G}^g_{e_u,e_v} = F_{k_{u,e_u}}(g_{e_u,e_v}) \oplus F_{k_{v,e_v}}(g_{e_u,e_v}) \oplus (k_{w,e_w} || e_w)$;
   (b) For $\alpha = e_u \oplus 1$, $\beta = e_v$, query the oracle $\mathcal{O}$ for $O^{g_{\alpha,\beta}}_{\alpha} = \mathcal{O}(k_{u,e_u}, g_{\alpha,\beta}, v_u \oplus 1)$, and compute $\tilde{G}^g_{\alpha,\beta} = O^{g_{\alpha,\beta}}_{\alpha} \oplus F_{k_{v,e_v}}(g_{\alpha,\beta}) \oplus (k_{w,e_w} || e_w)$.
   (c) For $\alpha = e_u$, $\beta = e_v \oplus 1$, query the oracle $\mathcal{O}$ for $O^{g_{\alpha,\beta}}_{\beta} = \mathcal{O}(k_{v,e_v}, g_{\alpha,\beta}, v_v \oplus 1)$, and compute $\tilde{G}^g_{\alpha,\beta} = F_{k_{u,e_u}}(g_{\alpha,\beta}) \oplus O^{g_{\alpha,\beta}}_{\beta} \oplus (k_{w,e_w} || e_w)$.
   (d) For $\alpha = e_u \oplus 1$, $\beta = e_v \oplus 1$, query the oracle $\mathcal{O}$ for $O^{g_{\alpha,\beta}}_{\alpha} = \mathcal{O}(k_{u,e_u}, g_{\alpha,\beta}, v_u \oplus v_v \oplus 1)$, and for $O^{g_{\alpha,\beta}}_{\beta} = \mathcal{O}(k_{v,e_v}, g_{\alpha,\beta}, 0)$, and compute $\tilde{G}^g_{\alpha,\beta} = O^{g_{\alpha,\beta}}_{\alpha} \oplus O^{g_{\alpha,\beta}}_{\beta} \oplus (k_{w,e_w} || e_w)$.
6. **Computing the output permutation bit:** for the output wire *out*, set $\lambda_{out} = e_{out} \oplus \mathcal{F}_C(x_1, \dots, x_n)$.
7. **Output to $\mathcal{D}_{\mathbf{MYao}}$:** Hand to $\mathcal{D}_{\mathbf{MYao}}$ the following: the corrupt parties' randomness Δ^i, $k^i_{w,0}$ for each $\mathsf{P}_i \in \mathcal{P}_A$ and for input wire w; the wire masks λ_{in_i} for each input wire of P_i; the real/fake garbled circuit $\{\tilde{G}^g_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$ for all AND-gates g, together with the public randomness $g_{\alpha,\beta}$; the output wire mask λ_{out}; the input wires external values and keys e_{in_j} and k^j_{w,e_w} (where in_j is the input wire belonging to $\mathsf{P}_j \in \mathcal{P}_H$, and w is the circuit's input wire).

Output: Output whatever $\mathcal{D}_{\mathbf{MYao}}$ outputs.

Fig. 10. Algorithm $\mathcal{D}_{CCR}$ for generating a garbled circuit for the distinguisher $\mathcal{D}_{\mathbf{MYao}}$

$v_u v_v = (\alpha \oplus \lambda_u)(\beta \oplus \lambda_v)$. The distinguisher can compute the real wires' values v_u and v_v and extract the wires' masks $\lambda_u = e_u \oplus v_u$ and $\lambda_v = e_v \oplus v_v$. Then,

$b_{\alpha,\beta}$ can be computed as

$$b_{\alpha,\beta} = (\alpha \oplus e_u \oplus v_u)(\beta \oplus e_v \oplus v_v) \oplus v_u v_v,$$

hence $b_{\alpha,\beta} = v_u \oplus 1$, if $\alpha = e_u \oplus 1$, $\beta = e_v$; $b_{\alpha,\beta} = v_v \oplus 1$, if $\alpha = e_u$, $\beta = e_v \oplus 1$; and $b_{\alpha,\beta} = v_u \oplus v_v \oplus 1$, if $\alpha = e_u \oplus 1$, $\beta = e_v \oplus 1$.

From the construction of the distinguisher $\mathcal{D}_{CCR}$ almost immediately follows the reduction. If $\mathcal{O} = Circ^F_{\tilde{\Delta}}$, then the view of the adversary, which is given by $\mathcal{D}_{CCR}$ to $\mathcal{D}_{\mathbf{MYao}}$, is distributed identical to the real-world view. In particular, the constructed garbled circuit is identical to the garbled circuit computed by $\mathcal{F}_{\text{garble}}$. If $\mathcal{O} = Rand$, then the adversarial view that $\mathcal{D}_{CCR}$ gives to $\mathcal{D}_{\mathbf{MYao}}$, is distributed identically to the ideal-world view of the adversary. In particular, the garbled circuit is identical to the garbled circuit constructed by the simulator $\mathcal{S}_{\mathbf{MYao}}$. The PPT distinguisher $\mathcal{D}_{CCR}$ then can distinguish between the oracles $Circ^F_{\tilde{\Delta}}$ and $Rand$ with the same non-negligible probability as $\mathcal{D}_{\mathbf{MYao}}$ distinguished between the real and ideal worlds, in contradiction to F being circular correlation robust.

Lastly, we consider the probability of a bad case $k_{u,0} = k_{v,0}$ or $k_{u,0} = k_{v,1}$ which is not covered by the simulation. As in C there are no gates with duplicated inputs, this case would only occur with overwhelmingly small probability.

– Bad case (illegal queries to $\mathcal{O}$): there is an AND gate in C with two equal input keys, i.e $k_{u,0} = k_{v,0}$ or $k_{u,0} = k_{v,0} \oplus \Delta$. The probability of such case for a single gate is $2^{-\ell(\kappa)+1}$; for the entire circuit, by the union bound, the probability is bounded by $|C| \cdot 2^{-\ell(\kappa)+1}$, and is negligible.

Thus, apart from the negligible probability, the simulation works in the ideal world.

Remark 1. As can be observed, our security theorem, as the security theorem of [12], is not in the offline-online model. To turn our construction to be provable in the offline-online model, it is possible to delay revealing the garbled circuit to the online protocol.

7 Proof of Security for MYao with Half-Gates

In this section, for completeness, we prove the security of our half-gates **MYao** with the row-reduction. We focus on the case of the parallel gate garbling, i.e., when the reduction occurs in only the first half-gate, i.e., when only $\tilde{g}_0 = 0$. As for the basic **MYao**, we first prove that the protocol $\Pi^{HG}_{\text{offline}}$ securely computes the "Yao"'s circuit, i.e., securely realizes the functionality $\mathcal{F}_{\text{garble}}$ in the presence of a semi-honest adversary corrupting any number of parties from $\mathcal{P}$. Then, we prove security of **MYao** in the $\mathcal{F}_{\text{garble}}$-hybrid model, assuming that the PRF F is *circular correlation robust*, according to Definition 1.

7.1 Security of the Half-Gate Garbling

As well as for the basic **MYao**, we assume the ideal functionalities $\mathcal{F}_F$, $f^{\ell(\kappa)}_{mult}$, and f_{mult}, which are securely realized by the protocols Π_F, $\Pi^{\ell(\kappa)}_{mult}$, and Π_{mult},

respectively, in the presence of a static semi-honest adversary corrupting any number of parties. For more details, see Section 2.1 and Appendix A. For completeness, we give the half-gate encryption functionality $\mathcal{F}_{\text{AND}}^{HG}$ in Figure 11.

Functionality $\mathcal{F}_{\text{AND}}^{HG}$ for Encrypting AND half-gates

Inputs: Each party P_i inputs $k_{u,0}^i$, $k_{v,0}^i$, $k_{w,0}^i$, $\Delta^i \in \{0,1\}^{\ell(\kappa)}$, $\lambda_u^i, \lambda_v^i, \lambda_w^i \in \{0,1\}$.
Computation:

1. The functionality reconstructs the wires' keys and masks, and the global offset as follows: $\Delta = \Delta^1 \oplus \cdots \oplus \Delta^n$, $k_{u,0} = k_{u,0}^1 \oplus \cdots \oplus k_{u,0}^n$, $k_{v,0} = k_{v,0}^1 \oplus \cdots \oplus k_{v,0}^n$, $k_{w,0} = k_{w,0}^1 \oplus \cdots \oplus k_{w,0}^n$, $k_{u,1} = k_{u,0} \oplus \Delta$, $k_{v,1} = k_{v,0} \oplus \Delta$, $\lambda_u = \lambda_u^1 \oplus \cdots \oplus \lambda_u^n$, $\lambda_v = \lambda_v^1 \oplus \cdots \oplus \lambda_v^n$, $\lambda_w = \lambda_w^1 \oplus \cdots \oplus \lambda_w^n$.
2. The functionality computes the garbled half-gates $\tilde{g}_1$ and $\hat{g}_\beta$ for $\beta \in \{0,1\}$ as follows.
   (a) Compute $F_{k_{u,0}}(g_{0,0})$ and parse it as $(\tilde{k}_{w,\tilde{e}_w^0} || \tilde{e}_w^0)$, where $\tilde{k}_{w,\tilde{e}_w^0} \in \{0,1\}^{\ell(\kappa)}$, $\tilde{e}_w^0 \in \{0,1\}$.
   (b) Compute the partition of the external values: $\tilde{e}_w^1 = \tilde{e}_w^0 \oplus \lambda_v$, $\hat{e}_w^0 = \tilde{e}_w^0 \oplus \lambda_w \oplus \lambda_u\lambda_v$, $\hat{e}_w^1 = \hat{e}_w^0 \oplus \lambda_u$,
   and of the keys: $\tilde{k}_{w,0} = \tilde{k}_{w,\tilde{e}_w^0} \oplus \tilde{e}_w^0 \cdot \Delta$, $\tilde{k}_{w,1} = \tilde{k}_{w,0} \oplus \Delta$, $\hat{k}_{w,0} = k_{w,0} \oplus \tilde{k}_{w,0}$, $\hat{k}_{w,1} = \hat{k}_{w,0} \oplus \Delta$.
   (c) Compute and output to the parties the reduced half gates

$$\tilde{g}_1 = F_{k_{u,1}}(g_{0,1}) \oplus \left(\tilde{k}_{w,\tilde{e}_w^1} || \tilde{e}_w^1\right); \quad \hat{g}_\beta = F_{k_{v,\beta}}(g_{1,\beta}) \oplus \left(\hat{k}_{w,\hat{e}_w^\beta} \oplus \beta \cdot k_{u,0} || \hat{e}_w^\beta\right), \tag{29}$$

   where $\beta \in \{0,1\}$, and F is a PRF.

Outputs: The functionality outputs $\tilde{g}_1$ and $\hat{g}_\beta$ for $\beta \in \{0,1\}$ to every party P_i.

Fig. 11. Functionality $\mathcal{F}_{\text{AND}}^{HG}$

Lemma 3. *The protocol Π_{AND}^{HG} securely computes the n-party functionality $\mathcal{F}_{\text{AND}}^{HG}$ in the $(\mathcal{F}_F, f_{mult}^{\ell(\kappa)}, f_{mult})$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Proof. The simulator $\mathcal{S}$ works as follows. While simulating Step 1(a) of the protocol Π_{AND}^{HG}, it simulates all the outputs of $\mathcal{F}_F$ by sending to the adversary the uniformly random strings $F_{u,\alpha}^i$ and $F_{v,\beta}^i$ of length $\ell(\kappa)$ for every $\mathsf{P}_i \in \mathcal{P}_A$, and every $\alpha, \beta \in \{0,1\}$. In step 1(b), it simulates the outputs of f_{mult} and $f_{mult}^{\ell(\kappa)}$ by sending uniformly random bits λ_u^i, λ_v^i, λ_{uv}^i, and uniformly random $\ell(\kappa)$-long strings Λ_u^i, Λ_v^i, Λ_*^i, respectively, for any $\mathsf{P}_i \in \mathcal{P}_A$.

Then, locally, for each $\mathsf{P}_i \in \mathcal{P}_A$, the simulator computes the following shares: $\tilde{e}_w^{1,i} = \tilde{e}_w^{0,i} \oplus \lambda_v^i$, $\hat{e}_w^{0,i} = \tilde{e}_w^{0,i} \oplus \lambda_w^i \oplus \lambda_{uv}^i$, $\hat{e}_w^{1,i} = \hat{e}_w^{0,i} \oplus \lambda_u^i$, $\tilde{k}_{w,\tilde{e}_w^1}^i = \tilde{k}_{w,\tilde{e}_w^0}^i \oplus \Lambda_v^i$, $\hat{k}_{w,\hat{e}_w^0}^i = k_{w,0}^i \oplus \tilde{k}_{w,\tilde{e}_w^0}^i \oplus \Lambda_*^i$, $\hat{k}_{w,\hat{e}_w^1}^i = k_{u,0}^i \oplus \hat{k}_{w,\hat{e}_w^0}^i \oplus \Lambda_u^i$.

In the output step, for the garbled gate $\{\tilde{g}_1, \hat{g}_0, \hat{g}_1\}$ obtained from the ideal functionality $\mathcal{F}_{\text{AND}}^{HG}$, the simulator computes

$$\tilde{g}_1^A = \tilde{g}_1 \oplus \bigoplus_{i \in A} \left(\tilde{F}_1^i \oplus \left(\tilde{k}_{w,\tilde{e}_w^1}^i || \tilde{e}_w^{1,i} \right) \right), \qquad \hat{g}_\beta^A = \hat{g}_\beta \oplus \bigoplus_{i \in A} \left(\hat{F}_\beta^i \oplus \left(\hat{k}_{w,\hat{e}_w^\beta}^i || \hat{e}_w^{\beta,i} \right) \right),$$

for every $\beta \in \{0,1\}$, XOR-shares them for all $\mathsf{P}_j \in \mathcal{P}_H$, and broadcast on their behalf.

In both real and ideal worlds, the view of the adversary together with the output of honest parties are distributed identically, hence the protocol Π_{AND}^{HG} is secure.

Lemma 4. *The protocol $\Pi_{\text{offline}}^{HG}$ securely realizes the n-party functionality $\mathcal{F}_{garble}^{HG}$ in the $\mathcal{F}_{\text{AND}}^{HG}$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Proof. The simulator $\mathcal{S}$, getting from the ideal functionality $\mathcal{F}_{garble}^{HG}$ the outputs $\{\tilde{g}_1, \hat{g}_0, \hat{g}_1\}$ for all AND gates, Δ^i, and $k_{w,0}^i$ for all input wires w, λ_{out} on each output wire *out*, for all $\mathsf{P}_i \in \mathcal{P}_A$, simulates the view of the adversary as follows. First, it sets the randomness of all corrupt parties P_i according to the $\mathcal{F}_{garble}^{HG}$'s choice of Δ^i, $k_{w,0}^i$, and λ_{out}^i. Notice that, as the adversary is semi-honest, in the real world parties would choose them uniformly at random, as $\mathcal{F}_{garble}^{HG}$ does in the ideal world.

The first step of the protocol is local, so the simulator doesn't send any messages. In the second step of the simulation, $\mathcal{S}$ simulates the output of $\mathcal{F}_{\text{AND}}^{HG}$ sending $\{\tilde{g}_1, \hat{g}_0, \hat{g}_1\}$, which it got from $\mathcal{F}_{garble}^{HG}$, to the adversary as the output of any corrupt party. In the third step, the simulator XOR-shares the bit $\left(\lambda_{out} \oplus \bigoplus_{i \in A} \lambda_{out}^i\right)$ in $|\mathcal{P}_H|$ shares and broadcasts them on behalf of the parties from $\mathcal{P}_H$.

In both real and ideal worlds, the view of the adversary together with the output of honest parties are distributed identically.

Theorem 3. *The protocol $\Pi_{\text{offline}}^{HG}$ securely realizes the n-party functionality $\mathcal{F}_{garble}^{HG}$ in the $(\mathcal{F}_F, f_{mult}^{\ell(\kappa)}, f_{mult})$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.*

Proof. By composition of Lemma 4 and Lemma 3.

7.2 Security of MYao with Half-Gates

As before, we denote by $\mathcal{F}_C$ the function computed by a circuit C that contains only XOR, AND, and NOT gates where the gate's input wires are not duplicated, and assume for simplicity that any $\mathsf{P}_i \in \mathcal{P}$ has only a single input bit x_i and gets the output from the only output wire *out* of C. We'll prove the security of Half-Gates **MYao** in the $\mathcal{F}_{garble}^{HG}$-hybrid model. In the real world, the view of the corrupt parties are:

1. The output of $\mathcal{F}_{garble}^{HG}$, i.e. the garbled circuit $\{\tilde{g}_1,\ \hat{g}_0,\ \hat{g}_1\}_{g\in C}$, the global offset's shares Δ^i, the input wire's keys shares $k_{w,0}^i$, the wire mask λ_{in_i} for any P_i's input wire, for any $i \in A$ and the output wire's mask λ_{out};
2. The messages of the online round, i.e. the input wire's external values e_w and the correspondent keys' shares k_{w,e_w}^j for $j \in H$.

In the ideal world, the simulator receives $y_{out} = C(x_1, \ldots, x_n)$ from the trusted party but learns only the inputs of parties from $\mathcal{P}_A$. As well as for the basic **MYao**, the simulator builds the fake garbled circuit, where only the sum of two half gates (which are on-the-evaluation-path) is computed by encryption. Every sum that completes any other full gate is chosen by the simulator uniformly at random. As before, the simulator creates all the input keys' and masks' shares uniformly at random. The proof of security, thus, is to show that no PPT distinguisher can distinguish between the real garbled circuit and the fake one.

The simulator $\mathcal{S}_{\mathbf{MYao}}^{HG}$ is given in Figure 12. As the simulator works exactly as $\mathcal{S}_{\mathbf{MYao}}$ except from Step 7 and output, in Figure 12 we give only the replacement for these steps in $\mathcal{S}_{\mathbf{MYao}}$.

Simulator Algorithm $\mathcal{S}_{\mathbf{MYao}}^{HG}$

Inputs: The circuit C and the inputs for the corrupted parties $\{x_i\}_{i\in A}$.
Algorithm:
Steps 1-6 are identical to Simulator $\mathcal{S}_{\mathbf{MYao}}$.

7. For each AND-gate with input wires u and v and output wire w, generate a fake garbled gate $\{\tilde{g}_1, \hat{g}_0, \hat{g}_1\}$ as follows:
   (a) Reconstruct the keys: $\hat{k}_{u,\hat{e}_u} = \bigoplus_{i=1}^n \hat{k}_{u,\hat{e}_u}^i$, $\hat{k}_{v,\hat{e}_v} = \bigoplus_{i=1}^n \hat{k}_{v,\hat{e}_v}^i$, $\hat{k}_{w,\hat{e}_w} = \bigoplus_{i=1}^n \hat{k}_{w,\hat{e}_w}^i$.
   (b) Sample $\tilde{g}_1 \leftarrow \{0,1\}^{\ell(\kappa)+1}$, $\hat{g}_{\hat{e}_v\oplus 1} \leftarrow \{0,1\}^{\ell(\kappa)+1}$
   (c) Set $\tilde{g}_0 = 0$, $\hat{g}_{\hat{e}_v} = F_{k_{u,\hat{e}_u}}(g_{0,\hat{e}_u}) \oplus F_{k_{v,\hat{e}_v}}(g_{1,\hat{e}_v}) \oplus \tilde{g}_{\hat{e}_u} \oplus \hat{e}_v \cdot (k_{u,\hat{e}_u}||1) \oplus (k_{w,\hat{e}_w}||\hat{e}_w)$.

Outputs (Simulation of the real-world Half-Gates MYao protocol):

1. The simulator $\mathcal{S}_{\mathbf{MYao}}^{HG}$ simulates the output of $\mathcal{F}_{garble}^{HG}$ to the corrupt parties by sending the fake garbled circuit $\{\tilde{g}_1, \hat{g}_0, \hat{g}_1\}$, together with the corrupted parties' randomness: $\hat{\Delta}^i$, $\hat{k}_{w,0}^i$ for any input wire w of the circuit, $\hat{\lambda}_{in_i}$ for any P_i's input wire in_i, for any $i \in A$, and $\hat{\lambda}_{out}$.
2. The simulator $\mathcal{S}_{\mathbf{MYao}}^{HG}$ simulates the messages of the honest parties in the first step of Π_{online}^{HG}: external value $\hat{e}_{in_j}$ for all input wires of $\mathsf{P}_j \in \mathcal{P}_H$, and input key shares $\hat{k}_{w,\hat{e}_w}^j$ for any input wire w of the circuit, and all $j \in H$.

Fig. 12. Algorithm $\mathcal{S}_{\mathbf{MYao}}^{HG}$ for generating a fake garbled circuit with half-gates and simulation of a real-world **MYao** protocol

Theorem 4. *Let C be some circuit, and denote by $\mathcal{F}_C$ the n-party functionality computing it. Assuming F is circular correlation robust, the protocol Π_{online}^{HG}*

securely realizes the n-party functionality $\mathcal{F}_C$ in the $\mathcal{F}_{garble}^{HG}$-hybrid model in the presence of a static semi-honest adversary corrupting any number of parties.

Proof. The output of the protocol is fully determined by the inputs $x_1, \ldots, x_n$ and equal in both worlds. Thus, in order to prove the semi-honest security of **MYao**, it is only needed to prove that for every input $x_1, \ldots, x_n$, the ideal-world view of the corrupt parties simulated by $\mathcal{S}_{\mathbf{MYao}}^{HG}$ (Figure 12) is computationally indistinguishable from the real-world view of these parties in an execution of the protocol in $\mathcal{F}_{garble}^{HG}$-hybrid model.

The real-world view of the corrupt parties includes the garbled circuit $\{\tilde{g}_1, \hat{g}_0, \hat{g}_1\}_{g \in C}$, and, additionally, the set $\left\{\{\Delta^i, \{k^i_{w,0}\}_w, \lambda_{in_i}\}_{i \in A}, \lambda_{out}, \left\{e_{in_j}, \{k^j_{w,e_w}\}_w\right\}_{j \in H}\right\}$, where w are input wires of the circuit, and in_i is the input wire, which belongs to P_i. The latter is simulated by $\mathcal{S}_{\mathbf{MYao}}^{HG}$ by sending the fake garbled circuit together with the set $\left\{\left\{\tilde{\Delta}^i, \{\tilde{k}^i_{w,0}\}_w, \tilde{\lambda}_{in_i}\right\}_{i \in A}, \tilde{\lambda}_{out}, \left\{\tilde{e}_{in_j}, \{\tilde{k}^j_w, \tilde{e}_w\}_w\right\}_{j \in H}\right\}$. Both sets are the sets of a uniformly random strings of the appropriate length, thus it remains to prove that the real and fake garbled circuits are indistinguishable for a PPT adversary.

We assume towards a contradiction that there exists a PPT distinguisher $\mathcal{D}_{\mathbf{MYao}}^{HG}$ which, for any fixed input $(x_1, \ldots, x_n)$, can distinguish between the ideal-world view of the adversary, created by the simulator $\mathcal{S}_{\mathbf{MYao}}^{HG}$ with the fake garbled circuit, and real-world view of the adversary consisting of the real garbled circuit. To obtain a contradiction, we construct the PPT distinguisher $\mathcal{D}_{CCR}^{HG}$ which can distinguish between the oracles $Circ_{\tilde{\Delta}}^{F}$ and $Rand$ (defined in Section 3.1) with non-negligible probability, and, therefore, to break the circular correlation robustness of the wPRF F.

The distinguisher $\mathcal{D}_{CCR}^{HG}$ is given access to an oracle $\mathcal{O} \in \left\{Circ_{\tilde{\Delta}}^{F}, Rand\right\}$ and constructs an adversarial view, using only legal oracle queries as shown in Figure 13. In every garbled gate with the input wires u and v and the output wire w, $\mathcal{D}_{CCR}^{HG}$ chooses one row in the evaluator half gate indexed by the external value e_v to encrypt by calling F. Note that if $e_u = 0$ then the correspondent half gate row $\tilde{g}_0$ is fixed to be 0, and, therefore, $\hat{g}_{e_v}$ is fully determined by the encryption keys and the output key of the gate. Otherwise, if $e_u = 1$, then the correspondent halves $\tilde{g}_1$ and $\hat{g}_{e_v}$ are XOR-shares of such an encryption.

Note that from Equations (17), (18), and (22), and from the row reduction $\tilde{g}_0 = 0$, it follows that

$$\tilde{g}_1 = F_{k_{u,0}}(g_{0,0}) \oplus F_{k_{u,1}}(g_{0,1}) \oplus \lambda_v(\Delta||1); \tag{30}$$

$$\begin{aligned} \hat{g}_0 \oplus \hat{g}_1 = F_{k_{v,0}}(g_{1,0}) \oplus F_{k_{v,1}}(g_{1,1}) \oplus \lambda_u(\Delta||1) \oplus (k_{u,0}||0) = \\ = F_{k_{v,0}}(g_{1,0}) \oplus F_{k_{v,1}}(g_{1,1}) \oplus (k_{u,\lambda_u}||\lambda_u). \end{aligned} \tag{31}$$

Thus, to compute the first half-gate $\tilde{g}_1$ for the fake circuit, the distinguisher makes query to the oracle $\mathcal{O}$ as $O_u = \mathcal{O}(k_{u,e_u}, g_{0,e_u \oplus 1}, v_v \oplus e_v)$. If $\mathcal{O} = Circ_{\Delta}^{F}$ then $O_u = F_{k_{u,e_u \oplus 1}}(g_{0,e_u \oplus 1}) \oplus (e_v \oplus v_v)(\Delta||1) = F_{k_{u,e_u \oplus 1}}(g_{0,e_u \oplus 1}) \oplus \lambda_v(\Delta||1)$, and a uniformly random $\ell(\kappa) + 1$-bit string otherwise. Then the distinguisher,

according to Equation (30), uses the response O_u to compute the first half as follows:

$$\tilde{g}_1 = F_{k_{u,e_u}}(g_{0,e_u}) \oplus O_u.$$

For the second half, $\hat{g}_0$ and $\hat{g}_1$, the distinguisher computes

$$\hat{g}_{e_v} = \tilde{g}_{e_u} \oplus F_{k_{u,e_u}}(g_{0,e_u}) \oplus F_{k_{v,e_v}}(g_{1,e_v}) \oplus (k_{w,e_w}||e_w) \oplus e_v(k_{u,e_u}||e_u).$$

The latter follows from Equation (20) taking $\alpha = e_u$, $\beta = e_v$.

In order to compute $\hat{g}_{e_v \oplus 1}$, the distinguisher makes a query $O_v = \mathcal{O}(k_{v,e_v}, g_{1,e_v \oplus 1}, v_u)$, and sets $\hat{g}_{e_v \oplus 1}$ as

$$\hat{g}_{e_v \oplus 1} = \hat{g}_{e_v} \oplus F_{k_{v,e_v}}(g_{1,e_v}) \oplus O_v \oplus (k_{u,e_u}||e_u).$$

Note that if $\mathcal{O} = Circ^F_{\Delta}$, then $O_v = F_{k_{v,e_v \oplus 1}}(g_{1,e_v \oplus 1}) \oplus v_u(\Delta||1) =$
$= F_{k_{v,e_v \oplus 1}}(g_{1,e_v \oplus 1}) \oplus (e_u \oplus \lambda_u)(\Delta||1)$, and a uniformly random $\ell(\kappa)+1$-bit string, otherwise. Which, in case $\mathcal{O} = Circ^F_{\Delta}$ is followed by $\hat{g}_{e_v \oplus 1} \oplus \hat{g}_{e_v} = F_{k_{v,e_v}}(g_{1,e_v}) \oplus F_{k_{v,e_v \oplus 1}}(g_{1,e_v \oplus 1}) \oplus (e_u \oplus \lambda_u)(\Delta||1) \oplus (k_{u,e_u}||e_u) = F_{k_{v,0}}(g_{1,0}) \oplus F_{k_{v,1}}(g_{1,1}) \oplus (k_{u,\lambda_u}||\lambda_u)$, i.e. Equation (31).

Notice that for any gate g, the distinguisher makes only two legal queries, as $g_{\alpha,\beta}$ is unique for any g, α, β, and there are no gates with duplicated inputs. Thus, each query of the form $(k, g, \cdot)$ is made just once for any pair (k, g).[6]

From the construction of the distinguisher $\mathcal{D}^{HG}_{CCR}$ almost immediately follows the reduction. If $\mathcal{O} = Circ^F_{\Delta}$, then the view of the adversary, which is given by $\mathcal{D}^{HG}_{CCR}$ to $\mathcal{D}^{HG}_{\mathbf{MYao}}$, is distributed identical to the real-world view. In particular, the constructed garbled circuit is identical to the garbled circuit computed by $\mathcal{F}^{HG}_{garble}$. If $\mathcal{O} = Rand$, then the adversarial view $\mathcal{D}^{HG}_{CCR}$ gives to $\mathcal{D}^{HG}_{\mathbf{MYao}}$, distributes identically to the ideal-world view of the adversary. In particular, the garbled circuit is identical to the garbled circuit constructed by the simulator $\mathcal{S}^{HG}_{\mathbf{MYao}}$. The PPT distinguisher $\mathcal{D}^{HG}_{CCR}$ then can distinguish between the oracles $Circ^F_{\Delta}$ and $Rand$ with the same non-negligible probability as $\mathcal{D}^{HG}_{\mathbf{MYao}}$ distinguished between the real and ideal worlds, in contradiction to F being circular correlation robust.

The formal description of the distinguisher $\mathcal{D}^{HG}_{CCR}$ is given in Figure 13. Steps 1-4 are the same as for $\mathcal{D}_{CCR}$, therefore we omit them in the description of the distinguisher algorithm.

Thus, apart from the negligible probability, the simulation works in the ideal world.

8 Double Row-Reduction via Half-Gates.

In this section, we explain how to achieve the double row reduction in **MYao**. This comes on the price of the sequential garbling of the gates, as the output keys are derived from the input ones. Below we give the technique which allows to perform the double row reduction in some fraction of the gates, while in the

[6] The collision is possible with only the negligible probability.

Distinguisher Algorithm $\mathcal{D}^{HG}_{CCR}$

Inputs: The circuit C, a fixing of parties' inputs $x_1 \dots x_n$.
Steps (1)-(4) are same as those in Figure 10.

5. **Computing the circuit:** For every AND-gate g with the input wires u and v, and the output wire w, sample the uniformly random indexes $g_{\alpha,\beta}$ for $\alpha, \beta \in \{0,1\}$, and compute three half-gates' garbled rows:
   (a) Query the oracle $\mathcal{O}$ for $O_u = \mathcal{O}(k_{u,e_u}, g_{0,e_u \oplus 1}, v_v \oplus e_v)$, and $O_v = \mathcal{O}(k_{v,e_v}, g_{1,e_v \oplus 1}, v_u)$.
   (b) compute $\tilde{G}_1 = F_{k_{u,e_u}}(g_{0,e_u}) \oplus O_u$.
   (c) compute $\hat{G}_{e_v} = \tilde{G}_{e_u} \oplus F_{k_{u,e_u}}(g_{0,e_u}) \oplus F_{k_{v,e_v}}(g_{1,e_v}) \oplus (k_{w,e_w} || e_w) \oplus e_v(k_{u,e_u} || e_u)$.
   (d) compute $\hat{G}_{e_v \oplus 1} = \hat{G}_{e_v} \oplus F_{k_{v,e_v}}(g_{1,e_v}) \oplus O_v \oplus (k_{u,e_u} || e_u)$.
6. **Computing the output permutation bit:** for the output wire *out*, set $\lambda_{out} = e_{out} \oplus \mathcal{F}_C(x_1, \dots, x_n)$.
7. **Output to $\mathcal{D}^{HG}_{\mathbf{MYao}}$:** Hand to $\mathcal{D}^{HG}_{\mathbf{MYao}}$ the following: the corrupt parties' randomness Δ^i, $k^i_{in,0}$ for each $\mathsf{P}_i \in \mathcal{P}_A$ and for input wire *in*; the wire masks λ_{in} for each input wire of P_i; the real/fake garbled circuit $\{\tilde{G}_1, \hat{G}_0, \hat{G}_1\}$ for all AND-gates g, together with the public randomness $g_{\alpha,\beta}$; the output wire mask λ_{out}; the input wires external values and keys e_{in_j} and k^j_{w,e_w} (where in_j is the input wire belonging to $\mathsf{P}_j \in \mathcal{P}_H$, and w is the circuit's input wire).

Output: Output whatever $\mathcal{D}^{HG}_{\mathbf{MYao}}$ outputs.

Fig. 13. Algorithm $\mathcal{D}^{HG}_{CCR}$ for generating a garbled circuit for the distinguisher $\mathcal{D}^{HG}_{\mathbf{MYao}}$

rest of them reducing only a single row, and thus keeps the garbling phase a constant round.

We start from the half-gates equations (17) and (18). From the equality (17) due to $\tilde{g}_0 = 0$, as in the case of the single row reduction, follows the equality (21). Hence $\tilde{e}^0_w$ can be parsed, and $\tilde{k}_{w,0}$ computed from $F_{k_{u,0}}(g)$ and Δ. Similarly, from $\hat{g}_0 = 0$ by Equation (18) it follows that

$$F_{k_{v,0}}(g_{1,0}) = (\hat{k}_{w,0} \oplus \hat{e}^0_w \cdot \Delta || \hat{e}^0_w), \tag{32}$$

from where in the same way one can get $\hat{e}^0_w$ and $\hat{k}_{w,0}$. Then the output mask is inevitably $\lambda_w = \tilde{e}^0_w \oplus \hat{e}^0_w \oplus \lambda_u \lambda_v$, and the output key is $k_{w,0} = \tilde{k}_{w,0} \oplus \hat{k}_{w,0}$. Thus, if all the AND-gates are encrypted with the double-row half-gates reduction, the constant-round garbling protocol would be impossible, as the gates are encrypted in a topological order, rather than in parallel.

We give the protocol Π^{HG-2}_{AND} to encrypt a single AND-gate with a double row-reduction in Figure 14. We denote the input wires of the gate by u, v, and the output wire by w. As an input, the gate encryption protocol Π^{HG-2}_{AND} takes from each party P_i it's shares of the input keys $k^i_{u,0}$, $k^i_{v,0}$, of the input wires' masks λ^i_u, λ^i_v, and of the global offset Δ^i. As the output, it gives not only the reduced half-gate $\tilde{g}^1_w$, $\hat{g}^1_w$, but also the shares of the output key $k_{w,0}$ and of the permutation bit λ_w.

First, similarly to Π^{HG}_{AND}, parties jointly compute the PRF's for the public gate indexes on the secret shared input keys, and get secret shares of the outputs, which we denote in the same way as for the single-row reduction case by $\tilde{F}^i_\alpha$, and $\hat{F}^i_\beta$ for $\alpha, \beta \in \{0, 1\}$.

In order to perform a double-row reduction, i.e. to set $\tilde{g}_0 = \hat{g}_0 = 0$, each P_i parses $\tilde{F}^i_0$ and $\hat{F}^i_0$ as shares of $(\tilde{k}_{w,*}||\tilde{e}^0_w)$ and $(\hat{k}_{w,*}||\hat{e}^0_w)$ respectively. From Equations (17), (18), the parties can locally compute shares of

$$\tilde{e}^1_w = \tilde{e}^0_w \oplus \lambda_v;\ \ \hat{e}^1_w = \hat{e}^0_w \oplus \lambda_u. \tag{33}$$

In order to compute shares of the output mask, parties need to perform a multi-party bit multiplication over the shares of the input masks, and then to compute shares of

$$\lambda_w = \tilde{e}^0_w \oplus \hat{e}^0_w \oplus \lambda_u\lambda_v. \tag{34}$$

For the output keys, the following relations hold:

$$\tilde{k}_{w,*} = \tilde{k}_{w,0} \oplus \tilde{e}^0_w \cdot \Delta;\ \ \hat{k}_{w,*} = \hat{k}_{w,0} \oplus \hat{e}^0_w \cdot \Delta;\ \ k_{w,0} = \tilde{k}_{w,0} \oplus \hat{k}_{w,0}. \tag{35}$$

Then, the output partitioned keys are

$$\tilde{k}_{w,\tilde{e}^1_w} = \tilde{k}_{w,0} \oplus \tilde{e}^1_w \cdot \Delta = \tilde{k}_{w,0} \oplus (\tilde{e}^0_w \oplus \lambda_v) \cdot \Delta = \tilde{k}_{w,*} \oplus \lambda_v \cdot \Delta; \tag{36}$$

$$\hat{k}_{w,\hat{e}^1_w} = \hat{k}_{w,0} \oplus k_{u,0} \oplus \hat{e}^1_w \cdot \Delta = \hat{k}_{w,0} \oplus k_{u,0} \oplus (\hat{e}^0_w \oplus \lambda_u) \cdot \Delta = \hat{k}_{w,*} \oplus k_{u,0} \oplus \lambda_u \cdot \Delta. \tag{37}$$

The parties can compute their shares of the output partitioned keys $\tilde{k}_{w,\tilde{e}^1_w}$ and $\hat{k}_{w,\hat{e}^1_w}$ on the price of two instances of $\Pi^{\ell(\kappa)}_{mult}$ to obtain shares of $\lambda_v \cdot \Delta$ and $\lambda_u \cdot \Delta$. The output wire's key is

$$k_{w,0} = \tilde{k}_{w,0} \oplus \hat{k}_{w,0} = k_{w,0} = \tilde{k}_{w,*} \oplus \hat{k}_{w,*} \oplus (\tilde{e}^0_w \oplus \hat{e}^0_w) \cdot \Delta, \tag{38}$$

and the parties compute shares of $k_{w,0}$ for the price of one more $\Pi^{\ell(\kappa)}_{mult}$ call.

Finally, according to the Equations (17), (18), parties locally compute shares of

$$\tilde{g}_1 = \tilde{F}_1 \oplus (\tilde{k}_{w,\tilde{e}^1_w}||\tilde{e}^1_w),\ \text{ and }\ \hat{g}_1 = \hat{F}_1 \oplus (\hat{k}_{w,\hat{e}^1_w}||\hat{e}^1_w),$$

and reconstruct the half-gates from them.

A garbled half-gate with a double-row reduction can be decrypted in the *online phase* in the same way as the half-gate with no row reduction taking $\tilde{g}_0 = \hat{g}_0 = 0$, i.e. applying the equality (20).

Combined row reduction for the layered and odd-even circuits. Next we consider a row-reduction via half-gates for a layered circuits, i.e. circuits which are possible to break in a multiplicative layers, where inputs of each layer are only outputs from the previous one.[7]

Applying double-row reduction to all the AND-gates in the circuit would result in increasing the number of rounds proportionally to the multiplicative

[7] A XOR-gate belongs to the latest of the layers it gets the input from.

Protocol Π_{AND}^{HG-2} for computing the garbled half-gate with a double-row reduction.

Inputs: The private inputs of each party P_i are it's shares of the input keys $k_{u,0}^i$, $k_{v,0}^i$, of the input wires' masks λ_u^i, λ_v^i, and of the global offset Δ^i. Additionally, each party holds the public numbers $g_{0,0}, g_{0,1}, g_{1,0}$, and $g_{1,1}$.
Computation:

1. The parties perform two parallel steps:
   (a) **Encryption of the gate number:** The parties call a multiparty protocol Π_F four times in parallel to compute shares $\bar{F}_\alpha^i$, $\hat{F}_\beta^i$ of four values $\bar{F}_\alpha = F_{k_{u,\alpha}}(g_{0,\alpha})$, $\hat{F}_\beta = F_{k_{v,\beta}}(g_{1,\beta})$ as PRF's on the secret-shared keys $k_{u,\alpha}$, $k_{v,\beta}$ of the public inputs (for every $\alpha,\ \beta \in \{0,1\}$). Each P_i parses $\bar{F}_0^i$ as $(\bar{k}_{w,*}^i || \bar{e}_w^{0,i})$, and $\hat{F}_0^i$ as $(\hat{k}_{w,*}^i || \hat{e}_w^{0,i})$,where $\bar{k}_{w,*}^i, \hat{k}_{w,*}^i \in \{0,1\}^{\ell(\kappa)}$, $\bar{e}_w^{0,i}, \hat{e}_w^{0,i} \in \{0,1\}$.
   (b) **Secure multiplication-1:** The parties call Π_{mult} on their inputs λ_u^i and λ_v^i and therefore obtain λ_{uv}^i. In parallel, they call two instances of $\Pi_{mult}^{\ell(\kappa)}$. The first input of P_i to each instance is Δ^i, and the second are λ_u^i and λ_v^i. The outputs are the shares Λ_u^i and Λ_v^i respectively.
2. **Secure multiplication-2:** The parties call $\Pi_{mult}^{\ell(\kappa)}$ on their inputs Δ^i and $(\bar{e}_w^{0,i} \oplus \hat{e}_w^{0,i})$, and get output shares Λ_w^i.
3. **Computation of the outputs' shares (local):** Each P_i locally computes the following shares: $\bar{e}_w^{1,i} = \bar{e}_w^{0,i} \oplus \lambda_v^i$, $\hat{e}_w^{1,i} = \hat{e}_w^{0,i} \oplus \lambda_u^i$, $\lambda_w^i = \bar{e}_w^{0,i} \oplus \hat{e}_w^{0,i} \oplus \lambda_{uv}^i$, $\bar{k}_{w,\bar{e}_w^1}^i = \bar{k}_{w,*}^i \oplus \Lambda_v^i$, $\hat{k}_{w,\bar{e}_w^1}^i = k_{u,0}^i \oplus \hat{k}_{w,*}^i \oplus \Lambda_u^i$, $k_{w,0}^i = \bar{k}_{w,*}^i \oplus \hat{k}_{w,*}^i \oplus \Lambda_w^i$.
4. **Outputs reconstruction:** Each P_i locally computes $\bar{g}_1^i = \bar{F}_1^i \oplus \left(\bar{k}_{w,\bar{e}_w^1}^i || e_w^{1,i}\right)$, and $\hat{g}_1^i = \hat{F}_1^i \oplus \left(\bar{k}_{w,\bar{e}_w^1}^i || e_w^{1,i}\right)$, and sends them to any P_j, $j \neq i$.

Outputs: Each P_i outputs $k_{w,0}^i$, λ_w^i, $\bar{g}_1 = \bar{g}_1^1 \oplus \cdots \oplus \bar{g}_1^n$, and $\hat{g}_1 = \hat{g}_1^1 \oplus \cdots \oplus \hat{g}_1^n$.

Fig. 14. Protocol Π_{AND}^{HG-2}

depth of the circuit. This comes from the necessity of garbling reduced gates layer by layer, as the output keys and permutation bits depend on the input ones. Breaking a layered circuit in pairs of the consecutive layers, the parties set all the input keys to the AND-gates in the first layer of the pair randomly, as in Π_{offline}. Then, encrypting all AND-gates in the first layers in parallel by Π_{AND}^{HG-2}, the parties obtain derived output wires' keys and masks from the gates of these layers, which become the input ones to the next, second, layers. After that, parties call Π_{AND}^{HG} for every AND-gate in the second layers in parallel.

This combined approach allows to reduce the number of rows on every AND-gate in the first layers down to two, and in the second layers to 3 while still keeping the constant number of rounds.

Remark 2. The combined row reduction via half gates could be generalized for the groups of s consecutive layers. The first layer of any group gets the randomly chosen input wires' keys and masks, while all the intermediate wires' keys and masks are derived layer by layer. In comparison with the single row reduction

offline phase, it increases the number of rounds approximately s times, which is still constant, as all the groups of layers are garbled in parallel.

Remark 3. Similar generalization is possible for the odd-even circuit, i.e. such a layered circuits where the output of the odd layers come only to the even layers, and vice versa. Then, the input wires' keys and masks to the odd layers are chosen uniformly at random, and all the AND-gates in the odd layers are encrypted in a parallel invocations of Π_{AND}^{HG-2}. After that, all the AND-gates in the even layers are encrypted in parallel by calling Π_{AND}^{HG} for every such a gate.

9 Efficient instantiation via weak PRF

In this section we explain the (weak) PRF we use for an efficient computation of a MYao garbled circuit. To compute the garbled gates in MYao, the output of a PRF F with an $\ell(\kappa)$-long key on a public index $g_{\alpha,\beta}$ is used for an XOR-encryption. Computing a PRF in MPC is generally a very complicated task, likely resulting in an inefficient protocol. Hence, the construction of F should be based on "MPC friendly" PRFs.

To reduce the communication and number of rounds in the offline phase, we base F on the LPN-style "MPC friendly" weak PRF of [15],[8] which we denote by f (see definition in Section 2, Equation (5)). According to [15], the recommended key, plaintext, and output size of f should be 2κ, 2κ, and κ respectively. Hence, $\ell(\kappa) = 2\kappa$ and we set $\kappa = 128$.[9] Note that f uses Binary keys, as required for free-XOR. However, it is a *weak* PRF and *shrinking*, as the output length is $\frac{\ell(\kappa)}{2}$. Therefore, we next explain the required modifications:

Since a wPRF requires random inputs, it is mandatory to replace the unique index $g_{\alpha,\beta}$ in all protocols with a (pseudo)-random input. One method is to use a hash function, such as SHA256, on $g_{\alpha,\beta}$, and use this as the input to f.[10]

In order to obtain a wPRF F with output length $\ell(\kappa)$, as required for Equation (15), we define the wPRF F on $\ell(\kappa) = 2\kappa$-long keys and (pseudo)-random $2\ell(\kappa)$-long input g, parsed as $g_1||g_2$, as follows:

$$F_k(g) = f_k(g_1)||f_k(g_2), \tag{39}$$

where f is the LPN-style wPRF of [15]. By simple hybrid arguments, it can be shown that F is a secure wPRF with output length $\ell(\kappa)$.[11]

Lemma 5. *If f is a wPRF, then F defined by Equation (39) is also a wPRF.*

As we use free-XOR, we further assume F is circular correlation-robust. For more details on the security of f and the MPC protocol for computing it see in [15]; the protocol appears in Appendix A.

[8] An alternative we considered is LowMC [2]; see Appendix A for a discussion on that.
[9] Note that as we use f with 1 non-random bit in the key, we lose 1 bit of security out of the $\kappa = 128$ bits of security.
[10] We remark that [10] showed several methods to optimize over this method.
[11] This construction does not work directly for PRFs as the inputs must be uncorrelated.

10 Preprocessing Protocol

The offline (garbling) phase of **MYao** includes communications for encryption of AND-gates, and for recovering the circuit, input keys and input/output masks. The encryption of the gates calls three main building blocks: the multiparty multiplication protocols Π_{mult} and $\Pi_{mult}^{\ell(\kappa)}$, and the multiparty wPRF computation protocol Π_F. The computation of F occurs in a correlated randomness model, when for each bit conversion, parties need to have an additive sharing of the same random bit over two fields $\mathbb{Z}_2$ and $\mathbb{Z}_3$ (*dBits*). Below we describe the protocol which is possible to use for such a CR-generation, and give the communication complexity of the preprocessing according to it. Any future work which obtains the significant improvement in *dBits* generation, would also improve the efficiency of **MYao**.

10.1 Correlated Randomness Generation.

The correlated randomness which parties from $\mathcal{P}$ need for the bit conversion protocols are two sharings of the same bit $u \in \{0,1\}$. Each P_i needs to obtain random $b^i \in \mathbb{Z}_2$ and $t^i \in \mathbb{Z}_3$ such that

$$\bigoplus_{i=1}^{n} b^i = \sum_{i=1}^{n} t^i \mod 3 = u. \tag{40}$$

We suggest the following generic approach to construct a protocol for such a correlated randomness computation. First, each P_i chooses a random bit b^i. Each party except from P_n also chooses a random ternary number $t^i \in \mathbb{Z}_3$ and represents it as a 2-bit string (i.e. $t^i \in \{00, 01, 10\}$). Then, parties compute a secure binary circuit to obtain shares of

$$t^n = \left(u - \sum_{i=1}^{n-1} t^i \right) \mod 3. \tag{41}$$

Finally, they send all shares of t^n to P_n, who reconstructs its ternary share of u.

Ternary addition. The base operation in this circuit is the addition modulo 3 of two ternary numbers represented by a bitstring. Next we consider how to compute $c = a + b \mod 3$. For a ternary number x, we denote by x_1 its most significant, and by x_0 – its lest significant bit. Also notice, that never $x_0 = x_1 = 1$. Then $c = c_1||c_0$, where $c_1 = a_1 \oplus b_1 \oplus a_0 b_0 \oplus a_0 b_1 \oplus a_1 b_0$, and $c_0 = a_0 \oplus b_0 \oplus a_0 b_1 \oplus a_1 b_0 \oplus a_1 b_1$. This operation requires 3 parallel calls of Π_{mult} for a secure multiparty computation of $(a_0 \oplus a_1)(b_0 \oplus b_1)$, $a_0 b_0$, and $a_1 b_1$.

Ternary subtraction. The final operation in the MPC circuit is the subtraction $c = a - b \mod 3$, where a is a single bit, and b is a 2-bit string where never $b_0 = b_1 = 1$. The result can be computed as $c_1 = b_0 \oplus a b_0 \oplus a b_1$, $c_0 = a \oplus b_1 \oplus a b_0$. This operation requires 2 parallel calls of Π_{mult} for a secure multiparty computation of ab_0 and ab_1, or a single Π_{mult}^2 call to compute $a \cdot b$.

An MPC circuit to compute t^n. The parties compute (41) in a layered circuit. In the first layer, they compute pairwise sums $(t^{2i+1} + t^{2i+2}) \mod 3$ for $i \in \{0, ..., n/2 - 1\}$. In each summation take part only parties holding these values, i.e. P_{2i+1} and P_{2i+2}. If some party has no value or its share, we set its share to be 0. Each of $(n-1)/2$ pairs call $\Pi_\times$ three times to complete a ternary sum. The first layer, in this way, requires a single OT-round, where each party takes part in 3 bit-OTs.

In the second layer the parties, in groups by 4, compute a pairwise sums of the results from the first layer. Out of each four parties $\{\mathsf{P}_{4i+1}, ..., \mathsf{P}_{4i+4}\}$, only the first two have a non-zero shares of the first term, and only the last two have a non-zero shares of the second term. To compute shares of 3 bit products, these parties call a 4-party Π_{mult} with 4 bit-OT calls, where each party takes part in 2 bit-OTs. In general, the second layer takes $3 \cdot 4 \cdot (n-1)/4 = 3(n-1)$ bit-OTs when each party takes part in 6 of them.

Similarly, the third, fourth, etc., $\log(n-1)$th layers work, where the jth layer requires each party taking part in $3 \cdot 2^{j-1}$ bit-OTs, with the overall OTs number equal to $3 \cdot 2^{j-2}(n-1)$.

The last layer of the circuit is the ternary subtraction computing t^n and recovering it to P_n. All n parties take part in it, which requires 2 calls to Π_{mult} with $(n-1)^2$ OTs each, where every party takes part in $2n-3$ out of them, and P_n in only $n-1$. The last round of the circuit is the recovering of the result to P_n where all parties send a 2-bit long share of the result to P_n.

Overall, the computation of the MPC circuit requires $\log(n) + 1$ OT-rounds and $3 \cdot (n-1) \sum_{j=1}^{\log(n)} 2^{j-2} + 2(n-1)^2 \approx 3.5n^2$ bit-OTs, where each party takes part in $3 \sum_{j=1}^{\log(n)} 2^{j-1} + 2(2n-3) \approx 7n$ bit-OTs, or in $3.5n$ bit-OTs as a sender.

For $\mathbb{Z}_3 \to \mathbb{Z}_2$ conversion, additionally to the shares of u in two fields, parties need to hold the binary shares of $(u+1 \mod 3) \mod 2$, which could be computed by parties in a binary circuit using one multiparty ternary addition when parties $\mathsf{P}_1, ...\mathsf{P}_{n-1}$ reuse their shares of the sum $\sum_{i=1}^{n-1} t^{(i)} \mod 3$, and P_n holds the value $(t^{(n)} + 1) \mod 3$ as its input. The ternary addition, as was explained above, requires 3 bit multiplications, but in this case it is enough to multiply only 2 bits, as the parties need to compute only the least significant bit of the result. Thus, in the second correlated randomness for $\mathbb{Z}_3 \to \mathbb{Z}_2$ conversion, each party is involved in only 2 OTs with P_n, and only P_n performs $2(n-1)$ bit-OTs. This round could be done after the last round of the source CR generation for the same conversion, where P_n performs less OTs than other parties, thus we can say that this computation is very cheap.

10.2 Communication and Round Complexity of Preprocessing.

The communication complexity of the preprocessing is overwhelmed by generating correlated random bits for Π_F. In Table 1 we summarize the number of basic operations needed in the garbling phase to compute the wPRF calls for a single AND-gate and, consequentially, the number of CR-bits, and bit-OTs per party in the preprocessing phase. Data is given per AND-gate in **MYao** with

no optimizations (NoOpt), with only row reduction (RR), and with half-gate optimization (HG).

Table 1. Communication complexity of the preprocessing per AND-gate

Gate opt.	# F	# $\mathbb{Z}_2 \to \mathbb{Z}_3$	# $\mathbb{Z}_3 \to \mathbb{Z}_2$	# CR bits	# bit-OT's per party
NoOpt, RR	8	$4\ell(\kappa)$	$16\ell(\kappa)$	$20\ell(\kappa)$	$70n\ell(\kappa)$
HG	4	$4\ell(\kappa)$	$8\ell(\kappa)$	$12\ell(\kappa)$	$42n\ell(\kappa)$

11 Implementation and Comparison

In this section we describe our implementation, code optimizations, and compare MYao with alternative multiparty garbled circuit solutions in terms of garbled circuit size, online computation time, and offline and online communication. To showcase the effect of free-XOR, we compare both on a circuit containing only AND gates, and on the AES-128 circuit that is composed mainly of XOR gates.

Basic implementation. We implemented the online computation part of the basic MYao protocol (Section 4) (without half-gates and row reduction), in C++. We expect the running times to also be representative for the protocol with half-gates and row reduction, as the overwhelming majority of the time is dedicated to the decryption of the garbled rows, and the number of decryptions is equal in both cases. We used building blocks taken from the implementation of [15] for a basic implementation of the "LPN-style" wPRF.[12] We then optimized the wPRF implementation as explained below.

We ran experiments to test our implementation and the effect of our code optimization, as well as to compare MYao with alternative multiparty garbled circuit solutions. Experiments were run on an Intel i7-6500u CPU 2.5 GHz machine with 8 GB of RAM with Ubuntu 22.04 OS, running on a single thread. Our code will be made public.

In our implementation, we used the following optimizations of [15]: **(1)** Bit packing, **(2)** Mod-3 bit slicing used for the addition and multiplication of $\mathbb{Z}_3$ elements, which we optimized even further as we explain below, **(3)** Efficient Toeplitz matrix representation, **(4)** Lookup table for multiplication with a constant matrix.

Optimizing the LPN-syle wPRF Implementation. Due to the bit slicing optimization, any $\mathbb{Z}_3$ element z is represented by 2 bits, the LSB $l_z = z \bmod 2$, and the MSB $m_z = 1$ if $z = 2$ and 0 otherwise. As explained in [15], addition

[12] We remark that [15] did not implement their "LPN-style" wPRF, only their $(2,3)$-wPRF. Nevertheless, it was possible to construct a basic implementation of the LPN-style wPRF using their protocol and the implemented building blocks, e.g., of Toeplitz matrix-vector multiplication and efficient $\mathbb{Z}_3$ operations.

and multiplication could then be computed using 7 bit operations and 6 bit operations, respectively.

To optimize the Toeplitz by vector multiplication *even further*, we make the following key observation: in the algorithm, there is a step where a *Binary* vector and a *Binary* Toeplitz matrix (represented by a vector) are viewed over $\mathbb{Z}_3$ and multiplied element-wise. It follows that the values in them are in fact limited to 0 and 1 only. This vector is then added to an accumulator vector with general elements in $\mathbb{Z}_3$. This allows to optimize the following two operations: multiplication of $\mathbb{Z}_3$ values where both values are limited to 0 or 1, and addition of two $\mathbb{Z}_3$ values where one of them is limited to 0 or 1.

- Multiplication optimization - instead of using the $\mathbb{Z}_3$ multiplication protocol, a simple AND is enough. This reduces the bit operations needed for multiplication from 6 to 1.
- Addition optimization - as the elements in the vector added to the accumulator all have MSB 0, the required number of bit operations for addition is reduced from 7 to 5.

As can be observed in Table 2, the above optimizations reduce the running time of the wPRF computation by approximately 70%.

Table 2. Running time of the wPRF f in microseconds, averaged over 100000 runs.

	Time(μ)
Baseline implementation	23.27
With optimized multiplication	11.02
With optimized addition and multiplication	7.17

11.1 Comparison

We next compare MYao to the following protocols and implementations: the DDH-based key-homomorphic protocol of [10], which we call **DDH**, the LWE-based key-homomorphic protocol of [10], which we call **LWE**, the LPN-based key-homomorphic protocol of [8], which we call **LPN**, the AHE-BGV-based key-homomorphic protocol of [18], which we call **BGV** and the standard **BMR** (using the semi-honest implementation of [9]). We additionally explain on comparisons with the Tinykeys [21] protocol and the protocol of Yang et al. [36]. We compare on 2 circuits: The AES-128 circuit, consisting of 6400 AND gates and 28176 XOR gates and a circuit consisting of 100000 AND gates.

Garbled circuit size. The size in bits of a garbled gate is: **MYao:** $3 \cdot 256$; **DDH:** $4 \cdot 2048$;[13] **LPN:** $4 \cdot 4845$;[14] **LWE:** $4 \cdot 10752$;[15] **BGV:** $4 \cdot 184320$;[16] **BMR:** $4 \cdot 128 \cdot n$. Notice that a MYao AND gate incorporates row-reduction (hence 3 instead of 4). Garbled circuit size is additionally affected by free-XOR; As **LWE**, **DDH** and **BGV** cannot incorporate free-XOR, the garbled circuit size of these protocols suffers significantly for XOR-heavy circuits such as AES.

The comparison is depicted in Table 3. It can be observed that MYao's garbled circuit is at least 90% smaller than protocols based on key-homomorphic PRFs. The MYao garbled circuit size is significantly smaller than the BMR garbled circuit size even for a medium number of parties. We mention that the *maliciously secure* protocol of Yang et al. [36] achieved a $\frac{2}{n}$-row reduction, and had a designated evaluating party that does not have a key (hence, the external value was added). However, asymptotically their garbled circuit size is almost the same as regular BMR, with garbled gate size of $4 \cdot \left(128 \cdot (n - \frac{2}{4} - 1) + 1\right)$ bits. The Tinykeys [21] protocol achieves significantly smaller garbled circuits than BMR, but requires assuming some fraction of honest parties. Furthermore, asymptotically, the circuit size behaves similarly to BMR. The smallest garbled circuits for Tinykeys in [21] are for 20 parties, where 16 of them are honest, requiring 1044 bits per AND gate. MYao's garbled circuit size is approximately 26% smaller, while not making any assumption on the number of honest parties. For a larger number of parties this advantage becomes much more significant.

Table 3. Sizes of garbled circuits for the AES-128 circuit and the 100000 AND gate circuit, and the size of a single garbled AND gate for n parties.

Protocol	Single AND gate	100000 AND gates	AES-128
MYao	**768 bits**	**9.2 MB**	**0.6 MB**
DDH [10]	8,192 bits	97.6 MB	33.8 MB
LPN [8]	19,380 bits	231.0 MB	14.8 MB
LWE [10]	43,008 bits	512.7 MB	177.3 MB
BGV [18]	737,280 bits	8.6 GB	2.9 GB
BMR [9]	$512n$ bits	$6.1n$ MB	$0.4n$ MB

Online computation time. The running times of our experiments are given in Table 4. Observe that for the circuit with only AND gates, MYao is 13% faster than LWE, while for AES-128 it is approximately 80% faster than LWE, as LWE does not have free-XOR. We did not run the DDH code, as [10] reported it to

[13] For DDH [10] we took the security hardness assumption of discrete log in a 2048-bit safe prime group, as it most closely corresponds to 128-bit security [34].

[14] For LPN [8] we took the parameters that resulted in minimal garbled circuit size.

[15] In the LWE-based key-homomorphic protocol, the garbled row contain 512 elements, each requiring 21 bits to represent (in the implementation they used 32 bits).

[16] In [18], no implementation was made, authors suggest using AHE-BGV encryption scheme where the ciphertext contains 4096 elements, each requiring ≥ 45 bits.

be slower than LWE. We thus expect the margin between MYao and DDH to be even bigger than the margin between MYao and LWE on both circuits.

Although the LPN code of [8] is not published, we can compare to their work: they incorporate free-XOR but reported slower times than [10] for AND gates. Therefore, as MYao outperforms LWE and incorporates free-XOR, we expect to be more than 13% faster than the LPN implementation of [8] on both circuits.[17] The online phase of the garbled circuit scheme of [18] wasn't implemented, but we estimate their online computation time to be significantly slower than LWE [10] as the encryption schemes are similar, but [18]'s is more complex and has longer keys. Therefore, we expect the online computation time of [18] to be much slower than of MYao's.

As can be observed from Table 4, the cut-off point against the BMR implementation of [9] is around 75-80 parties. Due to the quadratic complexity of the BMR online phase, the advantage increases rapidly as the number of parties grows.

Table 4. Online computation runtime, in seconds, on a circuit with 100000 AND gates, and the AES-128 circuit, averaged over 50 runs. Standard deviation was less then 5%.

Protocol	Circuit	$n = 70$	$n = 80$	$n = 90$	Circuit	$n = 70$	$n = 80$	$n = 90$
MYao	100000	3.85	3.85	3.85		0.30	0.30	0.30
LWE [10]	AND	4.42	4.42	4.42	AES-128	1.48	1.48	1.48
BMR (impl. of [9])	gates	3.16	3.93	5.30		0.22	0.31	0.39

Online communication. In MPC protocols based on garbled circuits, the online communication consists of 2 rounds. The first round consists only of each party sending the external value e_w on each wire w of its input wires, and thus is very cheap. The second round consists of each party P_i sending their keys (or their shares of the keys) associated with these external values, i.e., k^i_{w,e_w} for every input wire w. Thus, the majority of the communication comes from the second round and corresponds directly to the length of the keys. Since MYao has shorter keys than protocols based on key-homomorphic PRFs, MYao requires less online communication.

Offline communication. The offline communication is generally split into two parts: The first part is the preprocessing, which can be computed before knowing the function that is evaluated. In MYao, this consists of computing the correlated randomness bits needed for the PRF computation, i.e., the $\mathbb{Z}_2$–$\mathbb{Z}_3$ dBits. This is explained in detail in Section 10. The second part is the function-dependent offline, which is detailed in Section 5.[18]

Aiming to fairly compare with [8], in the following we consider a semi-honest version of their protocol, removing the added communication required in their protocol for ensuring malicious security. We took here their parameters that

[17] Note [8] achieve malicious security, whereas MYao is only semi-honestly secure.

[18] Several methods to move even more of the communication from the function-dependent phase to the preprocessing exist, but are outside the scope of this paper.

give the smallest offline communication, in particular the parameters with key length 664 bits, garbled row length 7140 bits, and error-rate of $\frac{1}{8}$. Thus, they require preprocessing 28,560 secret-shared "error bits" per gate. We remark that although preprocessing an "error bit" with error-rate $\frac{1}{8}$ requires only 2 multiplications (via OT), and thus is simpler than preprocessing a $\mathbb{Z}_2$–$\mathbb{Z}_3$ dBit required for MYao, LPN [8] requires significantly more preprocessed bits than MYao due to the large size of their garbled gates.

As can be observed in Table 5, MYao requires substantially less offline communication than the key-homomorphic based garbled circuit solutions, with approximately five times less communication than [8] and orders of magnitude less than the protocols of [10] (the protocols of [10] also do not have free-XOR). Due to moving to a lookup-table + HE based protocol, rather than an OT based one, the protocol of [18] requires less offline communication than MYao when $n \geq 60$, but comes with an additional cost of significant offline computation. Note that the protocol of [18] does not support free-XOR, so for most circuits the above threshold is even higher. We also note that the number of rounds in the offline phase of [18] is linear in the number of parties, which would be very costly in high latency networks where the number of parties is very large. It can also be observed that the prominent bottleneck of MYao is the preprocessing phase, taking up almost 95% of the total offline communication cost. Thus, a more efficient method for generating dBits would greatly enhance the efficiency of MYao and make it fully practical. We leave this as an open question.

Table 5. Comparison of the per party offline phase communication complexity, for a single garbled gate.

	Preprocessing	**Function-dependent offline**		**Total offline**
Protocol	# bit-OT's	# bit-OT's	non-OT	**(approx.)**
MYao	$10,752n$	$768n$	11 Kb	$23n$ Kb
LPN [8]	$57,120n$	$1,992n$	28.5 Kb	$118.2n$ Kb
LWE [10]	–	$906n$**K**	43 Kb	$1.8n$ Mb
DDH [10]	–	$50n$**M**	8.2 Kb	$100.7n$ Mb
BGV [18]	$2n$	–	1.32 Mb	1.32 Mb

11.2 Comparison to TinyKeys [21] and Yang et al. [36].

Comparison to TinyKeys [21] and Yang et al. [36] In this part, we compare the size of garbled circuits with the TinyKeys protocol and with the protocol of Yang et al. Like BMR, Tinykeys also incorporate free-XOR. We recall that TinyKeys works for some portion of honest parties, though not necessarily an honest majority. We compare to the setting with 20 parties, of which 16 are honest, as this is the smallest garbled circuit size given in [21]. In this scenario, the authors of [21] suggest using key length 13, which results in a garbled row size of $13 \cdot 20 + 1 = 261$ bits, and a garbled AND gate size of 1044 bits. As can be observed from Table 15, even when comparing to the TinyKeys protocol, MYao

produced a garbled circuit smaller by approximately 26%, even though MYao makes no assumption on the number of honest parties.

The *maliciously secure* protocol of Yang et al. [36] is similar to BMR with free-XOR, but has additionally a $2/n$-row reduction and a designated evaluating party that does not have a key; therefore, the external value must be added. Thus, its garbled gates are slightly smaller than BMR, and have only $4 \cdot \left(128 \cdot (n - \frac{2}{4} - 1) + 1\right)$ bits. As in the case of BMR, the garbled circuit size of [36] is significantly bigger than MYao's even for a moderate number of parties.

Protocol	Single AND gate	100000 AND gates	AES-128
MYao	768	76.8M	4.91M
TinyKeys* [21]	1044	104.4M	6.68M
Yang et al. [36]	$512(n - \frac{3}{2}) + 4$	$\sim 51.2(n - \frac{3}{2})$M	$\sim 3.27(n - \frac{3}{2})$M

Fig. 15. Sizes in bits of garbled circuits for the AES-128 circuit and the 100000 AND gate circuit, and the size of a single garbled AND gate. *For Tinykeys the numbers are for the specific case of 20 parties, of which 16 are honest.

Bibliography

[1] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 191–219. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

[2] Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. In: Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34. pp. 430–454. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

[3] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Transactions on Symmetric Cryptology **2020**, 1–45 (2020)

[4] Aly, A., Orsini, E., Rotaru, D., Smart, N.P., Wood, T.: Zaphod: Efficiently combining lsss and garbled circuits in scale. In: Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 33–44. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338469.3358943

[5] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 503–513. Association for Computing Machinery, New York, NY, USA (1990). https://doi.org/10.1145/100216.100287

[6] Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: a system for secure multi-party computation. In: Proceedings of the 15th ACM conference on Computer and communications security. pp. 257–266. Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1455770.1455804

[7] Ben-Efraim, A.: On multiparty garbling of arithmetic circuits. In: Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24. pp. 3–33. Springer, Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_1

[8] Ben-Efraim, A., Cong, K., Omri, E., Orsini, E., Smart, N.P., Soria-Vazquez, E.: Large scale, actively secure computation from LPN and free-XOR garbled circuits. In: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part III. pp. 33–63. Springer, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_2

[9] Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 578–590. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978347

[10] Ben-Efraim, A., Lindell, Y., Omri, E.: Efficient scalable constant-round MPC via garbled circuits. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 471–498. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_17

[11] Boneh, D., Ishai, Y., Passelègue, A., Sahai, A., Wu, D.J.: Exploring crypto dark matter: New simple prf candidates and their applications. In: Theory of Cryptography Conference. pp. 699–729. Springer, Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-03810-6_25

[12] Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.S.: On the security of the "free-XOR" technique. In: Theory of Cryptography Conference. pp. 39–53. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_3

[13] Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38

[14] Dinur, I., Goldfeder, S., Halevi, T., Ishai, Y., Kelkar, M., Sharma, V., Zaverucha, G.: MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In: Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 12828, pp. 517–547. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_18

[15] Dinur, I., Goldfeder, S., Halevi, T., Ishai, Y., Kelkar, M., Sharma, V., Zaverucha, G.: MPC-friendly symmetric cryptography from alternating moduli: candidates, protocols, and applications. In: Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41. pp. 517–547. Springer, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_18

[16] Dobraunig, C., Kales, D., Rechberger, C., Schofnegger, M., Zaverucha, G.: Shorter signatures based on tailor-made minimalist symmetric-key crypto. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 843–857. Association for Computing Machinery, New York, NY, USA (2022)

[17] Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: Advances

in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II 40. pp. 823–852. Springer, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_29

[18] Garg, R., Yang, K., Katz, J., Wang, X.: Scalable mixed-mode mpc. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 106–106. IEEE Computer Society (2024)

[19] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC. pp. 218–229. ACM New York, NY, USA, Association for Computing Machinery, New York, NY, USA (1987). https://doi.org/10.1145/28395.28420

[20] Grassi, L., Øygarden, M., Schofnegger, M., Walch, R.: From farfalle to megafono via ciminion: the prf hydra for mpc applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 255–286. Springer, Springer Nature Switzerland, Cham (2023)

[21] Hazay, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: Tinykeys: A new approach to efficient multi-party computation. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III. pp. 3–33. Springer (2018). https://doi.org/10.1007/s00145-022-09423-5

[22] Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. Journal of cryptology **33**(4), 1732–1786 (2020). https://doi.org/10.1007/s00145-020-09355-y

[23] Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer–efficiently. In: Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28. pp. 572–591. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_32

[24] Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: Flexible garbling for XOR gates that beats free-XOR. In: Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34. pp. 440–457. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_25

[25] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35. pp. 486–498. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40

[26] Makri, E., Rotaru, D., Vercauteren, F., Wagh, S.: Rabbit: Efficient comparison for secure multi-party computation. In: International Conference on Financial Cryptography and Data Security. pp. 249–270. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_12

[27] Makri, E., Wood, T.: Full-threshold actively-secure multiparty arithmetic circuit garbling. In: Progress in Cryptology–LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6–8, 2021, Proceedings 7. pp. 407–430. Springer, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-88238-9_20

[28] Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 591–602. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813705

[29] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce. pp. 129–139. Association for Computing Machinery, New York, NY, USA (1999). https://doi.org/10.1145/336992.337028

[30] Nieminen, R., Schneider, T.: Breaking and fixing garbled circuits when a gate has duplicate input wires. Journal of Cryptology **36**(4), 34 (Aug 2023). https://doi.org/10.1007/s00145-023-09472-4, https://doi.org/10.1007/s00145-023-09472-4

[31] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15. pp. 250–267. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15

[32] Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41. pp. 94–124. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_5

[33] Rotaru, D., Wood, T.: Marbled circuits: Mixing arithmetic and boolean circuits with active security. In: International Conference on Cryptology in India. pp. 227–249. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-35423-7_12

[34] of Standards, N.I., Technology: Recommendation for key management: Part 1 – general. Tech. Rep. NIST Special Publication 800-57 Part 1 Revision 5, U.S. Department of Commerce, Washington, D.C. (2020). https://doi.org/10.6028/NIST.SP.800-57pt1r5

[35] Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 39–56. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3133979

[36] Yang, K., Wang, X., Zhang, J.: More efficient MPC from improved triple generation and authenticated garbling. In: Proceedings of the 2020 ACM

SIGSAC Conference on Computer and Communications Security. pp. 1627–1646. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3417285

[37] Yao, A.C.C.: How to generate and exchange secrets. In: 27th annual symposium on foundations of computer science (Sfcs 1986). pp. 162–167. IEEE (1986). https://doi.org/10.1109/SFCS.1986.25

[38] Zahur, S., Rosulek, M., Evans, D.: 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, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34. pp. 220–250. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8

A The MPC-friendly (weak) PRF

When choosing the PRF for using in MYao, we look for PRFs that allow free-XOR (i.e., binary keys), and also minimize the amount of communication and number of communication rounds. Standard algorithms such as AES, although providing much more efficient online time, would be inefficient to compute in MPC, making MYao impractical. Therefore, we look at "MPC friendly" PRFs. Note that some candidates, such as Farfalle [20] and MiMC [1], work in fields with odd characteristic, and would therefore not be compatible with free-XOR. The candidates we found which are compatible with free-XOR are the LPN-style "MPC-friendly" wPRF of [15], LowMC [2], Rain [16], and Rescue [3]. Out of these protocols, the wPRF of [15] and LowMC require the least amount of communication for computing the garbled circuit. On the one hand, LowMC generally requires more communication rounds, as it requires at least 14 rounds which would have to be computed sequentially in MPC; some versions of LowMC require even more rounds. On the other hand, the preprocessing of LowMC requires computing only multiplication triples, which has been more extensively studied and has more efficient protocols than dBits. We decided to instantiate MYao with the wPRF of [15], though it seems that LowMC [2] is a possible alternative that should also be considered.

The function on our choice, i.e., the LPN-style weak PRF (wPRF) by Dinur et al. [15], is defined as follows. For a key $k \in \mathbb{Z}_2^\ell$, the matrix $K \in \mathbb{Z}_2^{m \times \ell}$ is constructed by having k as its first row, and every other row of K is obtained from the previous one by a cyclic rotation by one place, and therefore K is a Toeplitz matrix. The public matrix $B \in \mathbb{Z}^{t \times m}$, on an input $x \in \mathbb{Z}_2^\ell$, the wPRF f_k is defined by

$$f_k(x) = B \cdot [K \cdot x \oplus (K \cdot x \mod 3) \mod 2], \tag{42}$$

where in $(K \cdot x \mod 3)$ both K and x are reinterpret over $\mathbb{Z}_3$.

To withstand known attacks with κ-bit security, [15] recommend $\ell = m = 2\kappa$, $t = \kappa$. The wPRF, in this way, shrinks the input into the 2 times shorter output.

Moreover, the key is twice longer than the output of the function, therefore for the garbling, where a PRF is used for the XOR-encryption of the output key, we build the wPRF with the longer output from the LPN-style wPRF (42) as follows:

$$F_k(g) = f_k(g_1)||f_k(g_2), \tag{43}$$

where the $2m$-long (pseudo)random input g is parsed as $(g_1||g_2)$ with $|g_1| = |g_2| = m$.

Protocol Π_F for securely computing the wPRF F (Equation (39))

Inputs: Each party P_i holds an XOR-share $k^i \in \{0,1\}^{\ell(\kappa)}$ of the input key k.
Public input: A random $g \in \{0,1\}^{2\ell(\kappa)}$, and the constant matrix $B \in \mathbb{Z}_2^{\ell(\kappa)\times(\ell(\kappa)+1)/2}$.
Correlated randomness: As required for $\mathbb{Z}_2 \to \mathbb{Z}_3$ and $\mathbb{Z}_3 \to \mathbb{Z}_2$ bit conversions.
Computation: The parties compute the following steps:

1. $\mathbb{Z}_2 \to \mathbb{Z}_3$ **key conversion:** The parties call the protocol $Convert^{\ell(\kappa)}_{(2,3)}(k)$ using their shares of the keys as inputs, to obtain additive $\mathbb{Z}_3$-shares of the same key $k^{*i} \in \mathbb{Z}_3^{\ell(\kappa)}$.
2. **Local key-input multiplications:** Every P_i locally constructs two circulant matrices $K^i \in \mathbb{Z}_2^{\ell(\kappa)\times\ell(\kappa)}$ and $K^{*i} \in \mathbb{Z}_3^{\ell(\kappa)\times\ell(\kappa)}$ with first rows k^i and k^{*i} respectively. Then P_i parses g as $(g_1||g_2)$, where $g_1, g_2 \in \{0,1\}^{\ell(\kappa)}$, and locally computes $x^i = (K^i \cdot g_1||K^i \cdot g_2)$. Also, it locally computes $x^{*i} = (K^{*i} \cdot g_1 \mod 3||K^{*i} \cdot g_2 \mod 3)$.
3. $\mathbb{Z}_3 \to \mathbb{Z}_2$ **intermediate vector conversion:** The parties call the protocol $Convert^{2\ell(\kappa)}_{(3,2)}(x^*)$ using their shares x^{*i} as inputs, to obtain XOR-shares y^i of $y = (y^* \mod 2) \in \mathbb{Z}_2^{2\ell(\kappa)}$.

Output Step: Using the above results, every P_i locally computes $z^i = (x^i \oplus y^i)$, parses the result as $(z_1^i||z_2^i)$, where $z_1^i, z_2^i \in \{0,1\}^{\ell(\kappa)}$, and outputs the $\ell(\kappa)+1$-long share of the wPRF's output $F_k^i(g) = (B \cdot z_1^i||B \cdot z_2^i)$.

Fig. 16. Protocol Π_F

The n-party semi-honest MPC protocol to compute f in the CR-model on the public input is given in [15], and can be done in two rounds of communication to recover two consequent masked values. Each of these recovery steps is needed to convert the key and the intermediate result into the other field. However it is more communication efficient to make it in four rounds using the single party as the collector, who receives all the masked secret shares and sums them.

Also, we note that, according to Equation (43) for every key k, the parties compute the function twice: for the input g_1, and for the input g_2. Hence, they could convert the secret shares of k from $\mathbb{Z}_2$ to $\mathbb{Z}_3$ just once for both $f_k(g_1)$ and $f_k(g_2)$. Moreover, for **MYao** without the half-gates optimization, it is possible to reuse the same $\mod 3$-sharing of k to compute F_k twice for two rows in the garbled gate.

For completeness, we give the protocol Π_F for computing the function $F_k(\cdot)$ in Figure 16.

The protocol Π_F uses conversion protocols $Convert^{\ell(\kappa)}_{(2,3)}(\cdot)$ and $Convert^{2\ell(\kappa)}_{(3,2)}(\cdot)$ that calls $\ell(\kappa)$ and $2\ell(\kappa)$, respectively, parallel instances of the bit conversion protocols $Convert_{(2,3)}(\cdot)$ and $Convert_{(3,2)}(\cdot)$, respectively, in the correlated randomness model, which are given in Figure 17 and Figure 18, respectively.

$\mathbb{Z}_2 \to \mathbb{Z}_3$ bit conversion protocol

Input: Each party P_i has the XOR-share x^i shares of $x \in \{0,1\}$.
Correlated randomness: Each party gets shares $\tilde{x}^i$ over $\mathbb{Z}_2$ and $\tilde{x}^{*i}$ over $\mathbb{Z}_3$ of the same random bit $\tilde{x}$.
Computation: Each party P_i sends to every other party P_j the masked input share $\hat{x}^i = x^i \oplus \tilde{x}^i$. By XORing all $\hat{x}^i$, each party reconstructs $\hat{x}$.
Output reconstruction: Party P_i locally computes and outputs

$$x^{*i} = \hat{x}^i + \tilde{x}^i + (\hat{x} \cdot \tilde{x}^i) \mod 3.$$

Fig. 17. Protocol $Convert_{(2,3)}(x)$.

$\mathbb{Z}_3 \to \mathbb{Z}_2$ bit conversion protocol

Input: Each party P_i has the mod-3 additive share x^{*i} of $x^* \in \mathbb{Z}_3$.
Correlated randomness: Each party gets shares $\tilde{x}^i$ over $\mathbb{Z}_2$ and $\tilde{x}^{*i}$ over $\mathbb{Z}_3$ of the same random bit $\tilde{x}$. Additionally, it gets the share $\tilde{y}^i$ over $\mathbb{Z}_2$ of bit $\tilde{y} = (\tilde{x} + 1 \mod 3) \mod 2$.
Computation: Each party P_i sends to every other party P_j the masked input share $\hat{x}^{*i} = (x^{*i} + \tilde{x}^{*i}) \mod 3$. Summing all $\hat{x}^{*i}$ in $\mathbb{Z}_3$, each party reconstructs $\hat{x}^*$.
Output reconstruction: Party P_i locally computes and outputs

$$x^i = \begin{cases} 1 - \tilde{x}^i - \tilde{y}^i & \text{if } \hat{x}^* = 0; \\ \tilde{y}^i & \text{if } \hat{x}^* = 1; \\ \tilde{x}^i & \text{if } \hat{x}^* = 2. \end{cases}$$

Fig. 18. Protocol $Convert_{(3,2)}(x^*)$.

The conversion protocols $Convert_{(2,3)}(\cdot)$ and $Convert_{(3,2)}(\cdot)$ work in the correlated randomness preprocessing model, where the correlated randomness is the same uniformly random bit shared in two different fields $\mathbb{Z}_2$ and $\mathbb{Z}_3$. This primitive is a specific case of *dBits* studied in a number of works [7, 11, 33, 4, 27, 17, 26, 15]. Some of the suggested protocols for dBits are constant round, however, for sake of the overall communication efficiency, we also suggest the $O(\log(n))$-round protocol with complexity $10,5n$ bit-OT's per encrypted bit, which we give in Section 10.