Concrete Efficiency Improvements for Multiparty Garbling with an Honest Majority

Aner Ben-Efraim[1]* and Eran Omri[2]†

[1] Dept. of Computer Science, Ben Gurion University of the Negev, Israel
anermosh@post.bgu.ac.il
[2] Dept. of Computer Science, Ariel University, Israel
omrier@gmail.com

Abstract. Secure multiparty computation is becoming a necessary component in many real-world systems. The efficiency of secure two-party protocols has improved tremendously in the last decade, making such protocols efficient enough for many real-world applications. Recently, much attention is being diverted to making secure *multiparty* computation (for more than two parties) truly practical as well. In particular, the last couple of years saw a resurgence of interest in constant round secure protocols, based on the multiparty garbling paradigm of Beaver et al. (STOC 1990). Such protocols generally offer improved performance in high latency networks, such as the internet.

In this paper we consider the case where a majority of the parties are honest, and construct highly efficient constant round protocols for both the semi-honest setting and the malicious setting. Our protocols in the semi-honest setting significantly improve over the recent multiparty garbling protocols for honest majority of Ben Efraim et al. (ACM CCS 2016), both in asymptotic complexity and in concrete running time.

In the malicious setting, we consider security with abort when assuming more than 2/3 of the parties are honest. We show that by assuming the existence of simple preprocessing primitives, which do not require knowledge of the computed function, we get malicious security at almost the same cost as semi-honest security. I.e., the function dependent preprocessing and the online phase are almost identical to the semi-honest setting.

We ran experiments to measure the effect of our optimizations and to show that our protocols compete with the state-of-the-art constant round protocols.

Keywords: Constant Round MPC, Garbled Circuits, Concrete Efficiency, Honest Majority

*Supported by ISF grants 544/13 and 152/17, by a grant from the BGU Cyber Security Research Center, and by the Frankel Center for Computer Science.

†Supported by an Israel Science Foundation grants 544/13 and 152/17, and by a grant from the Israeli Science and Technology ministry.

1 Introduction

Protocols for secure multiparty computation (MPC) enable a set of mutually distrusting parties to carry out a joint computation on private inputs, correctly and without revealing anything but the output. Secure computation was introduced for the 2-party case by Yao [41], who also introduced the notion of *garbled circuits*. These are essentially encrypted versions of the circuit, which can be evaluated without the evaluator learning intermediate wire values. Definitions and protocols for secure multiparty computation soon followed in [21] and [11, 5]. Multiparty garbled circuits were introduced by Beaver et al. [1], who constructed a constant round multiparty protocol.

These feasibility results were initially thought to be of theoretical interest only, but since the first 2-party implementation of Fairplay [32], a huge amount of work (e.g., [27, 36, 34, 28, 2, 37, 42, 29, 35, 39] to name but a few) has brought down the running time of 2-PC tremendously, making it truly practical today.

The progress for the multiparty case has been somewhat slower, but is quickly catching up. The first implementation of secure *multiparty* computation, FairplayMP [3], was based on the multiparty garbling paradigm. However, since then, most implementations were based on the secret-sharing paradigm, e.g., [8, 9, 16, 7, 12, 18, 17, 25]. In the secret-sharing paradigm, the parties secret-share the values of the input wires of the circuit. Then, for each layer of the circuit, the parties compute shares for the next layer using interaction. Finally, the output wires' values are reconstructed. Thus, the number of rounds depends on the depth of the circuit.

Interestingly, the last couple of years saw a resurgence of interest in constant round protocols based on the multiparty garbling paradigm, e.g., [30, 4, 31, 23, 40]. Such protocols generally offer improved performance in high latency networks, such as the internet, as demonstrated in [4].

Our Results and Techniques. We consider the case where a majority of the parties are honest, and construct highly efficient constant round protocols for both the semi-honest setting and the malicious setting. Our protocols in the semi-honest setting significantly improve over the recently published constant round protocols for honest majority of Ben-Efraim et al. [4], both asymptotically and in concrete running time.

We achieve our main improvement by observing that not all values necessary for computing the multiparty garbled circuit need to be secret-shared. This is done in two steps: First, we replace the round reduction technique of [4] with the round reduction technique for BGW presented by Ishai and Kushilevitz [24]. In [24], the last round of BGW is omitted by performing share conversion to (not uniformly random) additive sharing, and then masking with additive shares of zero. Second, we use the fact that the shares are additive to *locally add* the necessary values after the share conversion step. Using this, we manage to bypass the most computationally expensive part of the protocols of [4]. We then optimize further by observing that in the honest majority setting not all parties need to contribute keys. Also, using standard optimizations from the literature, we efficiently compute additive secret-shares of zero locally and distribute the workload in the last round of the offline phase.

We then investigate malicious *security with abort* when assuming that more than 2/3 of the parties are honest. It is well-known that, from a theoretical standpoint, in this scenario full fairness is achievable. However, we relax the security definition, with the aim of obtaining better running times. We present a new 2-round protocol that is secure against malicious adversaries in the linear preprocessing model defined by Damgård and Ishai [15]. The linear preprocessing functionalities can be used before the function is known. Then, the function dependent preprocessing and the online phase in our malicious protocol are almost identical to our protocols in the semi-honest setting.

We ran experiments to measure the effect of our optimizations. For 31 parties, we found that our optimized protocols are more than 70%, and sometimes even more than 85%, faster than the respective protocols in [4]. We also ran experiments comparing our protocol with existing state-of-the-art constant round MPC protocols that use oblivious transfer. We show that in some scenarios, our new protocols outperform all other constant round protocols.

Related Works. Recently, two works have studied constant round MPC based on multiparty garbling in the malicious setting [23, 40]. The protocols in these works use oblivious transfer and are secure up to $n - 1$ corrupt parties. Thus, assuming oblivious transfer, they provide a stronger security guarantee, and so are suitable also in scenarios where our protocols are not. Nevertheless, we show that in some scenarios our protocols could be preferred. Furthermore, our protocols do not require OT. Therefore, the results of these works are orthogonal to ours. Previous works for malicious security in constant rounds, such as [15, 30, 31], provided no implementation, so it is not clear how concretely efficient they are.

Other works, such as [13, 33, 10], study constant round MPC only in a very restrictive scenario, e.g., only 3 or 5 parties and/or only 1 or 2 corrupt parties. It is not clear if and how the results of [13, 33] generalize to an arbitrary number of parties. The extension of [10] to n parties allows only up to $\sqrt{n}$ corrupt parties.

There has also been a significant amount of work done in non-constant round MPC, e.g., [18, 17, 25, 19, 26], but for deep circuits in high latency setting, these protocols have a slow online phase due to the number of rounds. Hence, they are incomparable with our work.

Organization. In Section 2, we review multiparty garbling. In Section 3, we explain our optimized protocols for the semi-honest setting. In section 4, we present a new protocol that is secure against malicious adversaries, in the linear preprocessing model of Damgård and Ishai [15]. In Section 5, we present experimental results, measuring the efficiency of our optimizations and comparing our protocols to state-of-the-art constant round protocols.

2 Preliminaries

In this section, we recall the basic definitions and constructions of multiparty garbling. We assume familiarity with the definitions of secure multiparty computation, with Yao's garbled circuit construction [41], with Shamir secret-sharing [38], and with the BGW protocol and its improvement [5, 20].

Conventions and Notations. We list some of the conventions and notations that we will use throughout this paper. We assume the existence of a circular 2-correlation robust PRF, which we denote by $\mathcal{F}^2$. When separating the offline and online phases, we even assume that $\mathcal{F}^2$ is a random oracle.[3] We consider a static adversary $\mathcal{A}$ corrupting a strict minority of the parties (or when stated also less than 1/3 of the parties). In the malicious setting we consider security with abort, namely, the adversary is allowed to prematurely abort the computation, even after seeing the output. In this case, honest parties that need to output a result should output $\perp$. We denote by $||$ concatenation of strings. The circuit of the function to be computed is denoted by C, and $g \in C$ denotes both the gate and its index. The set of wires is denoted by W, and $\mathcal{W}$ denotes the wires that are *not* outputs of XOR gates. The number of parties in the protocol is n, and t is the bound on the number of corrupt parties. The security parameter will be $\kappa = 128$, and $\oplus$ will denote both XOR of strings and addition in fields of characteristic 2.

Multiparty Garbling. In the multiparty setting, the first proposal for constructing a multiparty garbled circuit was given in [1]. We follow a simplified description for the semi-honest model, given in [4], which also adopts the free-XOR technique of Kolesnikov and Schneider [27].

The protocol consists of two phases. In the first phase, often called the offline phase, the parties collaboratively construct a garbled circuit. Then, in the online phase, the parties exchange masked input values and the corresponding keys. After that, each party locally computes the outputs of the function.

For constructing the garbled circuit, each party P_i chooses, for each wire $\omega \in \mathcal{W}$, two random keys, $k^i_{\omega,0}$ and $k^i_{\omega,1}$. To enable the free-XOR technique [27], the parties need to choose the keys such that $k^i_{\omega,1} = k^i_{\omega,0} \oplus R^i$ for some global offset R^i. A sometimes useful notion is the term "superseed", which refers to the concatenation of the keys of all the parties, for some wire ω and either the zero or the one key, i.e., $k_{\omega,0} \overset{\text{def}}{=} k^1_{\omega,0}||\ldots||k^n_{\omega,0}$ and $k_{\omega,1} \overset{\text{def}}{=} k^1_{\omega,1}||\ldots||k^n_{\omega,1}$.

Each wire ω in the circuit is assigned a random secret permutation bit λ_ω. This bit masks the real values of the wires during the online phase. For an AND gate with input wires $in1$, $in2$ and output wire out, the garbled gate is the encryptions $\tilde{g}^1_{\alpha,\beta}, \ldots, \tilde{g}^n_{\alpha,\beta}$ for $(\alpha, \beta) \in \{0,1\}^2$, where

$$\tilde{g}^j_{\alpha,\beta} = \left(\bigoplus_{i=1}^{n} \mathcal{F}^2_{k^i_{in1,\alpha}, k^i_{in2,\beta}}(g||j) \right) \oplus k^j_{out,0} \oplus \left(R^j \cdot ((\lambda_{in1} \oplus \alpha) \cdot (\lambda_{in2} \oplus \beta) \oplus \lambda_{out}) \right). \tag{1}$$

Notice that all the values are "encrypted" by *all* the parties. XOR gates are computed using the free-XOR technique of Kolesnikov and Schneider [27] – the permutation bit and keys on the output wire are set to be the XOR of those on the input wires; they require no cryptographic operations or communication.

[3] This is to allow the garbled circuit to be revealed at the end of the offline phase.

For the circuit output wires, the permutation bits are revealed. For input wires of party P_i, the corresponding permutation bits are disclosed to party P_i.

During the online phase, an evaluating party learns at each wire a bit, called the *external* or *public* value, and the corresponding superseed. The external value is the XOR of the real value with the random permutation bit. Since the permutation bit is random and secret, the external value reveals nothing about the real value to the evaluating party. The evaluating party uses the external value and keys to continue the evaluation of the proceeding garbled gates. For the output wires of the circuit, the permutation bit values are revealed, and thus the output is learnt by XORing with the external values.

In the following last part of this section, we explain an abstraction of the general multiparty garbling paradigm. We do this informally and slightly imprecisely for clarity. Precise treatments can be found in [23] and in the full version.

The offline phase can be seen as an invocation of the functionality $\mathcal{F}_{GC}$, given in Figure 1, which constructs the garbled circuit and assigns the random permutation bits. In the $\mathcal{F}_{GC}$-hybrid model, after invoking the functionality $\mathcal{F}_{GC}$, running the online protocol $\mathit{\Pi}_{online}$, given in Figure 2, securely computes the output of the function in the semi-honest model.

In fact, a much stronger statement is true, as shown in [23]: running the online protocol in the $\mathcal{F}_{GC}$-hybrid model is secure also against malicious adversaries, even if the adversary is allowed to abort (i.e., see the output and hide it from the honest parties). Furthermore, the correctness requirement of $\mathcal{F}_{GC}$ can be weakend – the adversary can choose either to abort or to insert any additive error into the garbled gates *after seeing the garbled circuit*. The intuition is that the adversary cannot base its additive error on the permutation bits, or on the keys and inputs of the honest parties – the attack is only based on the garbled circuit, which appears random to the adversary. Thus, in the online phase, an honest evaluator will either notice the error and output $\perp$, or output the correct result. We denote this modified functionality, introduced in [23], by $\mathcal{F}_{GC}^{mal}$.

3 Optimizations for BGW based Sub-Protocols

In this section we describe our protocols that are secure in the semi-honest setting. These protocols compute a multiparty garbled circuit using the BGW protocol [5, 20]. Our protocols are similar in spirit to the protocols that were presented in [4] for honest majority, specifically to $BGW3$ and $BGW2$. We first present the main ideas of the optimizations in Sections 3.1 and 3.2. Then, we give a full description of our protocol with all the optimizations in Section 3.3. We state security of our optimized protocol in Section 3.4. The proof will appear in the full version.

The protocols we describe in this section are secure against semi-honest adversaries. However, in Section 4 we show that our 2-round protocol can be made secure also for malicious adversaries. As often happens, the optimizations presented in this section, in the semi-honest model, carry over also to the malicious model.

Functionality $\mathcal{F}_{GC}$

Computation Course:

1. The functionality assigns[a] a random global offset $R^i \in \{0,1\}^\kappa$ for every party P_i.
2. For each wire $\omega \in \mathcal{W}$, the functionality assigns[a]
   - A random permutation bit $\lambda_\omega \in \{0,1\}$.
   - Random zero keys $k^i_{\omega,0} \in \{0,1\}^\kappa$, and the one keys $k^i_{\omega,1} = k^i_{\omega,0} \oplus R^i$, for each party P_i.
3. For each XOR gate $g \in C$ with input wires $in1, in2$ and output wire out, the functionality computes
   - The permutation bit of the output wire $\lambda_{out} = \lambda_{in1} \oplus \lambda_{in2}$.
   - The zero keys of the output wire, $k^i_{out,0} = k^i_{in1,0} \oplus k^i_{in2,0}$ for each party P_i.
4. The functionality computes the garbled circuit GC. For every AND gate $g \in C$ with input wires $in1, in2$ and output wire out, every $\alpha, \beta \in \{0,1\}$, and every $j \in [n]$, compute:

$$\tilde{g}^j_{\alpha,\beta} = \left(\bigoplus_{i=1}^{n} \mathcal{F}^2_{k^i_{in1,\alpha},k^i_{in2,\beta}}(g||j)\right) \oplus k^j_{out,0} \oplus \Big(R^j \cdot ((\lambda_{in1} \oplus \alpha) \cdot (\lambda_{in2} \oplus \beta) \oplus \lambda_{out})\Big)$$

Outputs:

1. The functionality outputs the garbled gates, $\left(\tilde{g}^1_{\alpha,\beta}|| \cdots ||\tilde{g}^n_{\alpha,\beta}\right)$ for every $\alpha, \beta \in \{0,1\}$ and every $g \in C$, to the evaluating party.[b]
2. For output wires of the circuit, the functionality outputs the permutation bits to the evaluating party.[b]
3. The functionality outputs to each party P_i its global difference R^i, its zero key, $k^i_{w,0}$, for each wire $w \in \mathcal{W}$, and the permutation bit of each of its input wires.

[a] Unlike [4], we follow [23] and make this functionality inputless.

[b] We slightly divert from [4, 23] by having only one evaluating party.

Fig. 1: Functionality $\mathcal{F}_{GC}$ for Constructing a Multiparty Garbled Circuit

3.1 Reducing the Computational Complexity

In this section we explain how we reduce the computational complexity of the BGW based protocols described in [4] from being cubic in the number of parties to being quadratic in the number of parties. This involves using techniques suggested in [24, Appendix A]. The idea there was to reduce the number of rounds for computations that use the BGW protocol, by omitting the last degree reduction step. This is done by multiplying the shares by the interpolation constants,

Protocol $\mathbf{\Pi_{online}}$

1. **Send garbled labels associated with inputs:** For every circuit-input wire w:
   (a) Let P_i be the party whose input is associated with wire w and let x_{i_w} be P_i's input bit associated with the wire. Then, P_i sends $e_w = x_{i_w} \oplus \lambda_w$ to all parties. For every wire w, we denote by e_w the XOR of the actual value on the wire (based on the input) and λ_w; we call this the external value.
   (b) Each party P_j sends its part k^j_{w,e_w} of the garbled label on w to the evaluating party.
   (c) At this point, the evaluating party holds $k^1_{w,e_w}, \ldots, k^n_{w,e_w}$ for every circuit-input wire.
2. **Local circuit computation:** The evaluating party P_0 locally evaluates the garbled circuit by traversing the circuit in a topological order, computing gate by gate. Let g be the current gate with input wires *in*1, *in*2 and output wire *out*. Let e_{in1} and e_{in2} be the *extrenal values* on wires *in*1 and *in*2, respectively.
   (a) If g is a XOR gate, then P_0 sets $e_{out} = e_{in1} \oplus e_{in2}$. In addition, for every $j = 1, \ldots, n$, it computes $k^j_{out,e_{out}} = k^j_{in1,e_{in1}} \oplus k^j_{in2,e_{in2}}$.
   (b) If g is an AND gate, then P_0 computes

$$k^j_{out,e_{out}} = \tilde{g}^j_{\alpha,\beta} \oplus \left(\bigoplus_{i=1}^{n} \mathcal{F}^2_{k^i_{in1,\alpha}, k^i_{in2,\beta}}(g||j) \right)$$

   for every $j \in [n]$, and for $\alpha = e_{in1}, \beta = e_{in2}$, which are the external values on wires *in*1, *in*2 as above. Given $k^0_{out,e_{out}}$, the evaluating party P_0 compares it to the garbled labels $k^0_{out,0}, k^0_{out,1}$ that it received in the offline phase on this wire. If it equals $k^0_{out,0}$ then $e_{out} = 0$. If it equals $k^0_{out,1}$ then $e_{out} = 1$. Otherwise, an honest P_0 outputs $\perp$.[a]
3. **Output determination:** For every output wire w, the evaluating party computes the *real* output bit of wire w to be $e_w \oplus \lambda_w$, where e_w is the external value on wire w and λ_w is as received in the offline phase.

[a] Note that outputting $\perp$ can only happen in the malicious setting.

Fig. 2: The online phase – circuit evaluation

to convert the Shamir shares into additive shares, and then masking these with additive shares of 0.[4] Thus, the last round reduction becomes redundant.

In [4], they instead reduced the number of rounds by sharing the PRF values at a higher degree at the onset. Thus, the multiplication of shares were added to the PRF value shares, resulting in legitimate Shamir shares. Therefore, additive shares of 0 were not needed.

The problem with *all* previous *BGW-based* protocols for computing a multiparty garbled circuit, such as in [4], is that the outputs of the PRFs (or PRGs) were secret-shared using Shamir secret-sharing.[5] Since the length of these outputs is linear in the number of parties, and Shamir secret-sharing is computationally quadratic in the number of parties, the total computational complexity of these protocols is cubic in the number of parties.

To overcome this, we show that the PRF values, as well as the keys, do not need to be shared at all! Instead, only the global offset and permutation bits are shared in Shamir secret-sharing. The keys and outputs of the PRFs are instead simply added, after the share-conversion step. Since the result is then masked by additive shares of 0, this is secure. The formal statement is given in Section 3.4.

The number of values shared (by each party) using Shamir secret-sharing in the resulting scheme is independent of the number of parties. Therefore, the total complexity of computing these Shamir shares is quadratic in the number of parties. Furthermore, using an observation made in [24], each of these values needs to be shared only *once*. This is regardless of the number of times the value is used. The number of secrets shared additively is linear in the number of parties, but computing additive sharing is also linear in the number of parties. Thus, the total complexity of computing the additive shares is also quadratic in the number of parties.

To summarize, our new protocols have computational complexity $O(|C|\kappa n^2)$ while the computational complexity of previous honest majority protocols, such as those of [4], is $O(|C|\kappa n^3)$. Our protocols also require slightly less communication.

3.2 Further Optimizations

In this section, we describe some further optimizations to our protocol.

Shortening the "Superseed". When assuming an honest majority, or even just a known bound on the number of corrupt parties, the length of the "superseeds" can be shortened. That is, only a subset of the parties need to choose input keys. If the bound on the number of corrupt parties is t, then it is sufficient that $t+1$ parties choose keys, so the length of a "superseed" is $(t+1)\kappa$, where κ is the security parameter.

[4] Note that in our protocol, as well as in [24], this "share conversion" is done on multiplication of shares, so the resulting "shares" are not fully random. Nevertheless, they indeed sum to the secret, and the security is maintained by the masking.

[5] An exception is [15] who used the outputs of PRGs for encrypting only, but they instead secret-shared 0's of the same length using Shamir secret-sharing.

Assuming up to $n-1$ corrupt parties results in the standard "superseed" length. However, when assuming an honest majority, the length of the "superseed" is cut in half. This gives a significant improvement to running time, both in the offline phase and the online phase. To the best of our knowledge, this optimization has not been noticed, or at least not published, in previous works for multiparty garbling with an honest majority, e.g., [1, 3, 4, 15].

This optimization is "for free" when only the evaluating party needs to recover the output. However, when more than one party needs to recover the output, we need to explain how the parties receive the output, because not all the parties can actually perform the evaluation. In the semi-honest model, we can let the evaluating party simply send back the output to all the parties.

In the malicious model, we can solve the issue using a broadcast channel. The evaluating party broadcasts the superseeds of the output wires. Then, each of the $t+1$ parties that contributed for the superseed broadcasts the external value of the output wires (which they can check by comparing with their key). If any single party aborts or if any two parties broadcast different sets of values, then all honest parties abort. Otherwise, $t+1$ parties broadcast the same value. Since at least one of these parties is honest, it can be shown that this is the true output – informally, if the adversary can fake the other key for an honest party, then the adversary can recover the global offset of an honest party, which means the adversary already effectively broke security. A formal statement and proof will be given in the full version.

More Efficient Zero Secret Sharing. Assuming the existence of a PRG, there is a standard method to generate additive zero shares locally, and therefore more efficiently in terms of communication, and hence, also in concrete running time. This is done as follows:

Setup: Each party P_i that will need to share 0, sends a private random seed s_i^j to every other party P_j.

Computation: In order for party P_i to additively share 0 among all the parties
- Each party P_j locally computes its share of desired length by using the PRG with the seed s_i^j.
- Party P_i locally computes the sum (XOR) of all the above as its share.[6]

Distributing the Last Round. In the last round in our offline protocol, all parties send their shares to the evaluating party. The evaluating party receives the messages from all other parties and sums them together (along with its own shares). This implies that the evaluating party is performing much more work than all other parties. However, in many cases, the computational power and network capabilities of all the parties is approximately equal. Therefore, *in some scenarios*, it is preferable to distribute the workload between all the parties.

To do this, we use a technique known as "hypercube" [6]. The idea is that the 'last round' in the offline phase is done in $\log n$ rounds (instead of 1). At each

[6] In fields of characteristic other than 2, party P_i computes the *negation* of the sum.

round, the remaining parties pair up, and at each pair one sends the message to the other, which sums up the shares with its own shares, and proceeds to the next round. It might seem counter-intuitive to go from a constant round protocol to one which has $\log n$ number of rounds, but the idea is to to evenly distribute the workload. Notice that the overall amount of information sent over the network remains unchanged.

An example scenario where this optimization could be particularly useful is when the parties are in two far away clusters. Then, all the parties at each cluster can sum their shares together, and only one message needs to be sent over the high latency network. We will see in Section 5 that this optimization resulted in significant improvement for the LAN setting. However, we remark that preliminary results for the WAN setting suggest this optimization might not be suitable when there is high latency between *every pair* of parties.

3.3 The Optimized Protocol

In this section, we give the full details of our offline protocols, that include all the above mentioned optimizations. We recall that in the offline phase, the parties securely compute functionality $\mathcal{F}_{GC}$. The online protocol we use, described in Figure 2, is the same as [4], with the only difference being that here only one party evaluates.

We first give a description of our 3-round protocol, which optimizes $BGW3$ from [4]. We will refer to it as $BGW3_{opt}$.

Protocol Description. All $\tilde{g}^j_{\alpha,\beta}$ values (for all $j = 1, \ldots, t+1$, all $\alpha, \beta \in \{0,1\}$ and all gates) are computed in parallel. Each value is computed as follows (with some of the computations done only once globally, as explained in Section 3.1):

Communication round 1: Each party shares a random bit λ^i_ω for every wire $\omega \in \mathcal{W}$, using $(t+1)$-out-of-n Shamir secret sharing, with $t = \lceil n/2 \rceil - 1$. For an input wire, the λ is shared only by the party choosing the input of that wire. Each party P_i, $i \in \{1, \ldots, t+1\}$ also shares its random offset R^i using $(t+1)$-out-of-n Shamir secret sharing. In addition, the first $t+1$ parties share a 0-string of length $4|C|(t+1)\kappa$ in an n-out-of-n additive (XOR) secret-sharing scheme.

Local computation 1: The parties carry out local additions, as required by the circuit (e.g., free-XOR), in order to obtain shares of $\lambda_\omega = \oplus^n_{i=1} \lambda^i_\omega$ for every wire of the circuit. For every AND gate with input wires u, v, the parties locally multiply their shares of λ_u and λ_v. Denote this product by λ_{uv}.

Communication round 2: The parties run the GRR [20] degree reduction step on the product λ_{uv} to recover Shamir shares of $\lambda_u \lambda_v$ of degree t.

Local computation 2: The parties locally compute shares of $(\lambda_u \oplus \alpha)(\lambda_v \oplus \beta) \oplus \lambda_\omega$ by a linear combination of their shares of λ_u, λ_v, λ_w, and $\lambda_u \lambda_v$. Next, the parties locally multiply the result with their share of R^j. The parties next perform a share-conversion step, where each party multiplies the result by the reconstruction constant.[7] Next, each party P_i locally adds $\mathcal{F}^2_{k^i_{u,\alpha},k^i_{v,\beta}}(g||j)$, and party P_j also locally adds $k^j_{\omega,0}$, to the result. This is then masked with *a fresh* additive sharing of 0.

Communication round 3: The parties send the above masked values to the evaluator, who sums the received shares to obtain $\tilde{g}^j_{\alpha,\beta}$. As explained in Section 3.2, it is sometimes preferable to perform this step in $\log n$ rounds, using the hypercube technique, instead of in one round. In addition, for each circuit output wire ω, each party sends its share of λ_ω to the evaluator, and the evaluator recovers λ_ω.

The above protocol has 3 rounds (or $2 + \log n$ rounds) of interaction. As observed in [4] for their 3-round protocol, if more than 2/3 of the parties are honest, then $3t < n$. Thus, the degree reduction round can be omitted, resulting in the 2-round protocol $BGW2$.

It turns out that also here, if more than 2/3 of the parties are honest, then the degree reduction is unnecessary by setting $t = \lceil n/3 \rceil - 1$. The parties use their local multiplication, λ_{uv}, for the rest of the computations. The reconstruction of the gates works as before. This gives us a two (or $1 + \log n$) round protocol, which we call $BGW2_{opt}$.

3.4 Security

In this section we state the security of the protocols presented in Section 3.3 in the semi-honest model. Following [4, Theorem 2.1], if the above protocols securely compute $\mathcal{F}_{GC}$, then there is an efficient constant round MPC protocol secure in the semi-honest setting. Thus, we give our main security statement. The proof is given in the full version.

Theorem 1. *The protocols $BGW3_{opt}$ and $BGW2_{opt}$ securely compute $\mathcal{F}_{GC}$ in the standard model, when there are at most t semi-honest corrupt parties, for $t < \frac{n}{2}$ and $t < \frac{n}{3}$ respectively.*

4 Protocol for the Malicious Model

In this section we describe a new protocol that is secure against malicious adversaries, based on protocol $BGW2_{opt}$. The protocol requires that $> 2/3$ of the parties are honest and allows the adversary to abort prematurely (i.e., learn the output and hide it from the honest parties).

The protocol we describe shares many similarities with the constant round protocol of Damgård and Ishai [15], and uses the same preprocessing functionalities. The protocol of Damgård and Ishai guarantees full security, while ours guarantees only security with abort. However, by allowing the adversary to abort, we gain several efficiency benefits:

[7] If there are redundant parties for interpolation, then these parties multiply by 0 instead.

- Our online is significantly faster: in [15] they perform error correction and polynomial interpolation for reconstructing the gates in the online phase. In our protocol the interpolation is implicitly achieved using share conversion, in the offline phase. Errors result in abort.
- Our computational complexity and concerete efficiency are better than those of [15]. In particular, allowing abort enables us to use the optimizations described in Section 3.
- We can allow a greater number of corrupt parties for a two round MPC protocol in the linear preprocessing model than the two-round protocol of [15] – we require $> 2/3$ honest parties whereas they require either $> 4/5$ honest parties or more rounds.

The Linear Preprocessing Model. The functionalities we require appear in [15], under the name *the linear preprocessing model*. We did not implement these functionalities, but we note that they are used only independently of the computed function. A suggested implementation appears in [15], but we believe it can be much improved using modern techniques. We leave a fast implementation of these functionalities for future work. We recall the functionalities here:

$\mathbf{RandSS(t)}$ – Each party P_i obtains $f(i)$, where f is a random polynomial over GF(2^κ) of degree at most t.

$\mathbf{RandSS_0(t)}$ – Same as $\mathbf{RandSS(t)}$, except that f is subject to the restriction that $f(0) = 0$.

$\mathbf{RandSS_{bin}(t)}$ – Same as $\mathbf{RandSS(t)}$, except that f is subject to the restriction that $f(0) \in \{0, 1\}$.

$\mathbf{RandSS^{P_i}(t)}$ – Same as $\mathbf{RandSS(t)}$, except that party P_i additionally receives the polynomial f.

$\mathbf{RandSS^{P_i}_{bin}(t)}$ – Same as $\mathbf{RandSS_{bin}(t)}$, except that party P_i additionally receives the polynomial f.

We further add another simple preprocessing functionality:
AdditiveZeroSharing(AZS) – The parties receive additive (XOR) n-out-of-n shares of 0.

In the semi-honest model, these functionalities are easily achieved, e.g., in $\mathbf{RandSS_{bin}(t)}$ each party shares a random bit using Shamir secret-sharing and the parties sum their received shares. This does not extend to the malicious model, as the parties could share a number not in $\{0, 1\}$, or even inconsistent shares.

In contrast, for the functionality **AZS**, using the semi-honest protocol in which $t+1$ parties additively share 0 and then the parties sum the received shares, suffices for our protocol – an attack on this protocol would be translated to an additive attack on the output of $\mathcal{F}^{mal}_{GC}$, which is allowed. The formal statement and proof will be given in the full version.

Fixing the External Values of the Output Wires. In most descriptions of the BMR protocol, e.g., [1, 3, 4, 23], the evaluating parties receive the hidden permutation bits of the output wires. That way, they can XOR this value with the external value that they recover from the evaluation of the circuit, and thus recover the output.

A possible solution would be to force the parties to commit to their shares of the output wires, as done, e.g., in [23]. However, in order to base our protocol solely on the above preprocessing functionalities, we proceed in a slightly different manner; instead of revealing the hidden output bits of the output wires, they are fixed to be 0. Therefore, the evaluating party can recover the true output value from the output external value, as they are equal.

There are two obvious obstacles. The first is that an output wire may be an output wire of an XOR or XNOR gate.[8] These gates are free in the BMR protocol [4], and this effectively means that the permutation bits of these wires are correlated with permutation bits of the input wires of the gate; the shares of the output wires of XOR gates are computed by XORing the shares of the input wires of the respective gates.

This obstacle can be overcome by changing output "free-XOR" gates to regular garbled XOR gates. But a more efficient solution is to add garbled buffer gates for each output wire which is the output wire of an XOR/XNOR gate. These buffer gates are garbled as follows: for input wire *in* and output wire *out*, the garbled rows are

$$\tilde{g}^j_\alpha = \left(\bigoplus_{i=1}^{n} \mathcal{F}^2_{k^i_{in,\alpha}}(g||j)\right) \oplus k^j_{out,0} \oplus \left(R^j \cdot ((\lambda_{in} \oplus \alpha) \oplus \lambda_{out})\right), \qquad (2)$$

for $\alpha \in \{0,1\}$. Now the permutation bit λ_{out} can be set to 0, simplifying further. Notice that the size of a garbled buffer gate is half the size of a garbled AND gate. Thus, the total size of the garbled circuit is increased by only two garbled rows for every output wire that is the output of an XOR/XNOR gate.

The second obstacle is if the different parties are supposed to recover different outputs. If each wire is supposed to be recovered by either exactly one party or by all parties, then for output wires recovered by party P_i, we could use $\mathbf{RandSS}^{\mathbf{P_i}}_{\mathbf{bin}}(t)$ to share its permutation bit. Allowing other types of subsets would require adding another preprocessing functionality, in order to fix the permutation bits on those output wires.

4.1 Protocol Description

In this section we describe our maliciously secure protocol, in the linear preprocessing model.

Upon receiving the circuit, the parties first add to the circuit a buffer gate for each output wire that is an output of a XOR/XNOR gate.[9] These buffer gates will be garbled (cf. Equation 2). Now the parties follow the protocol $BGW2_{opt}$, with the following changes to the first communication round:

[8] We ignore NOT gates, as they can be eliminated by modifying the circuit, without enlarging the number of garbled gates.

[9] We again assume that there are no NOT gates in the circuit.

1. For permutation bits, the parties execute $\mathbf{RandSS_{bin}(t)}$ instead of each party sharing a random bit and then summing the received shares.
2. For permutation bits of input wires of party P_i, party P_i receives the random permutation bit and all parties receive the shares by executing the functionality $\mathbf{RandSS^{P_i}_{bin}(t)}$.
3. For each required i, each party P_i receives its random offset $R^i \in \mathrm{GF}(2^\kappa)$, and all parties receive Shamir shares of R^i, by executing functionality $\mathbf{RandSS^{P_i}(t)}$.
4. Shares of the permutation bits of the output wires, which are fixed to 0, are received using functionality $\mathbf{RandSS_0(t)}$.
5. Additive shares of 0 are generated using functionality **AZS**.

The rest of the protocol is identical to $BGW2_{opt}$ (recall that communication round 2 in $BGW3_{opt}$ is omitted in $BGW2_{opt}$), except that the output permutation bit shares are not sent (the output permutation bits are fixed to 0). This implies that the function dependent offline phase of our maliciously secure protocol is almost the same as the function dependent preprocessing of our semi-honest protocol – the only difference is the extra garbled buffer gates sent and that the shares of the output permutation bits are not sent. We call the above protocol $BGW2^{mal}_{opt}$.

Security. We next give the security statement of our protocol. The proof is given in the full version. Using the result of Hazay et al. [23], this suffices to realize an efficient constant round MPC protocol in the malicious setting.

Theorem 2. *Protocol $BGW2^{mal}_{opt}$ securely computes $\mathcal{F}^{mal}_{GC}$ in the linear preprocessing model in the presence of t maliciously corrupt parties, with $t < \frac{n}{3}$.*

5 Experimental Results

In this section we measure the running times of our protocols and compare our results with state-of-the-art constant round protocols.

Implementation and Running Environment. Our code was written in C++; it is publicly available at https://github.com/Aner2005/Protocols. For implementing $\mathcal{F}^2$, we used 128-bit fixed-key AES, as suggested in [2],[10] with pipelined AES-NI. For efficient field multiplications in $\mathrm{GF}(2^{128})$, we used the CLMUL commands [22]. We ran our experiments in a computer cluster comprised of Intel XEON 2.20 GHz machines (E5-2420) with 6 cores running Linux (Ubuntu1404-64-STD), and with a 1Gb connection and approximately 0.2ms ping time.

We benchmarked the timing of our protocols for 13 and for 31 parties on both the AES circuit, consisting of 6800 AND gates, and the SHA256 circuit, consisting of 90875 AND gates. All experiment results are the average on 25 protocol timings - 5 runs with 5 repetitions in each run. Synchronization steps were placed before and after each timed phase described below. Following convention, e.g., [35, 23, 40], we split our timings into the following phases:

[10] This gives slightly less than 128 bits of security, see [2].

Function Independent Preprocessing The knowledge required at this phase is only an upper bound on the number of AND gates in the circuit. This part consists of Shamir secret-sharing the permutation bits and offsets, and also of the additive zero-sharing.

Function Dependent Preprocessing This phase requires knowledge of the function to be evaluated. It consists of local multiplication, share conversion and reconstruction of the garbled circuit by the evaluator.[11] In $BGW3$ there is also a degree reduction round performed at this phase.

Online This phase requires the inputs of the parties. It consists of exchanging masked inputs, sending the corresponding input keys to the evaluator, and the evaluation of the garbled circuit by the evaluator.

In addition, there is a short **Setup** phase that is done once regardless of the number of functions/gates evaluated (but requires fixing the parties that will participate throughout). In this phase the parties exchange private keys and precompute the reconstruction constants.

Optimizations Measurements. We tested the effect of our different optimizations with comparison to the original protocols of [4]. We give here the results for 31 parties on the SHA-256 circuit. Basic refers to the original sub-protocol of [4] (with only one evaluator). Reduced Complexity is including the optimization described in Section 3.1. Short Superseeds, Efficient Zero-Sharing and Hypercube are including the respective optimizations described in Section 3.2 (optimizations are aggregated). For 31 parties our optimizations reduced the total time by over 70% in $BGW3$ and over 80%, and in some case over 85%, in $BGW2$!

SHA256, 31 parties		Basic	Red. Complexity	Short S.seeds	Eff. Zero-Sharing	Hypercube
	Ind. Pre.	28.006	13.027	6.944	6.094	6.196
BGW3	Dep. Pre.	6.557	6.427	3.224	3.222	1.092
($t = 15$)	Online	1.151	1.153	0.381	0.376	0.401
	Ind. Pre.	27.827	12.710	4.697	4.296	4.26
BGW2	Dep. Pre.	6.345	6.186	2.088	2.085	0.712
($t = 10$)	Online	1.127	1.164	0.27	0.257	0.282

Table 1: Measuring the effect of our different optimizations on a LAN. Times are average in seconds.

Remark 1. We did not measure the running time of our malicious protocol, as we did not implement the necessary preprocessing functionalities in Section 4. However, we note that the running time of the function dependent preprocessing and the running time of the online time should be almost identical to the semi-honest case – the only difference is adding buffer gates to the output wires which are output of XOR gates. This corresponds to $< 1\%$ and $< 0.2\%$ of the total number

[11] As mentioned, performing the reconstruction of the garbled circuit at the offline phase, before the inputs are given, requires assuming RO for proving security.

of AND gates in the AES and the SHA256 circuits respectively (recall that buffer gates are only half the size of an AND gate). As for the function indpenedent preprocessing, we believe that a fast implementation of the preprocessing functionalities, which would make our malicious protocol truly practical, is possible with today's techniques. We leave this for future work. Furthermore, as noted in [15], if the number of parties is small, then these preprocessing functionalities can be computed locally, by using a one-time setup and share conversion [14] (but this works only for a very small number of parties, as the setup time grows exponentially with the number of parties).

Comparison with the State-of-the-Art. The state-of-the-art, concretely efficient, constant round, secure multiparty (for an arbitrary number of parties) computation protocols are implementations of multiparty garbled circuits using oblivious transfer – the work of [4] for the semi-honest case,[12] and the works of [23] and [40] for the malicious case.

We note that, in contrast to our protocols, these protocols are secure up to $n-1$ corrupt parties. Thus, there are 2 possible comparisons – with respect to the same number of parties or with respect to the same number of *corrupt* parties. For example, for 13 participating parties, if an honest majority is assumed then one could compare running the OT protocols with 13 parties or with 7 parties. If more than $2/3$ of the parties are assumed to be honest, one could compare to 13 parties or to 5 parties. We give a comparison of our protocols, for 13 parties, with the BGW and OT protocols of [4]. [13]

We see that for the same number of parties, our protocols outperform the protocols of [4]. When comparing the same number of *corrupt* parties, we observe that although our total offline time is slightly slower than that of the OT version of [4], our function dependent preprocessing time is significantly faster. In some scenarios, the function might not be known a long time in advance, and this could therefore be significant. Also, our setup time is faster, because we don't need to perform baseOTs. We remark that since we used the same online protocol as [4], the timing for the same number of *corrupt* parties should be approx. equal, with ours being slightly slower because more parties need to exchange masked inputs.

Unfortunately, at the time of writing, the codes of [23] and of [40] were not yet publicly available. However, as noted in both [23] and [40], their timings are very similar to the timings of the OT protocol of [4] (albeit having a much stronger security guarantee). Thus, following Remark 1, it seems that for the function dependent preprocessing and for the online phase, the comparison of $BGW2_{opt}$ with the OT protocol of [4] is a good indication for the comparison of $BGW2_{opt}^{mal}$ with the protocols of [23] and [40]. For the function independent preprocessing times, see Remark 1.

[12] The work of [4] also presented the BGW protocols $BGW2$ and $BGW3$ along with their OT protocol. However, in their work they found that their OT protocol almost always significantly outperforms their BGW protocols for the same number of parties.

[13] For fair comparison, we changed the code of [4] so only one party evaluates the circuit also there.

		$BGW3_{opt}$	OT protocol of [4]		$BGW3$ of [4]
		13 parties	13 parties	7 parties	13 parties
AES	Ind. Pre.	0.115	0.01	0.007	0.315
	Dep. Pre.	0.05	0.244	0.102	0.142
	Online	0.021	0.037	0.018	0.033
SHA256	Ind. Pre.	1.322	0.125	0.089	3.563
	Dep. Pre.	0.448	2.389	1.090	1.154
	Online	0.149	0.303	0.144	0.309

		$BGW2_{opt}$	OT protocol of [4]		$BGW2$ of [4]
		13 parties	13 parties	5 parties	13 parties
AES	Ind. Pre.	0.084	0.01	0.006	0.308
	Dep. Pre.	0.025	0.244	0.071	0.119
	Online	0.016	0.037	0.011	0.034
SHA256	Ind. Pre.	1.005	0.125	0.079	3.514
	Dep. Pre.	0.234	2.388	0.802	1.071
	Online	0.1	0.303	0.093	0.303

Table 2: Comparison of our protocols with the protocols of [4]. Times are average in seconds. For the OT protocol, we compare for both the same number of parties and the same number of *corrupt* parties.

To conclude, our protocols are competitive with the state of the art constant round protocols in the semi-honest setting. In the linear preprocessing model, our $BGW2^{mal}_{opt}$ protocol is also competitive in the malicious setting. Thus, in some circumstances, our protocols might be considered as good alternatives.

Acknowledgements. The authors would like to thank Yehuda Lindell, Amos Beimel and Roi Inbar for helpful discussions.

Bibliography

[1] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM. ISBN 0-89791-361-2. doi: 10.1145/100216.100287. URL http://doi.acm.org/10.1145/100216.100287.

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

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

[4] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 578–590, 2016.

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

[6] D. Bertsekas, C. Özveren, G. Stamoulis, P. Tseng, and J. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11(4):263 – 275, 1991. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/0743-7315(91)90033-6. URL http://www.sciencedirect.com/science/article/pii/0743731591900336.

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

[8] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

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

[10] N. Chandran, J. Garay, P. Mohassel, and S. Vusirikala. Efficient, constant-round and actively secure MPC: Beyond the three-party case. In *24rd ACM Conference on Computer and Communications Security*, ACM CCS '17, 2017. To appear.

[11] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 11–19, 1988.

[12] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *Topics in Cryptology–CT-RSA 2012*, pages 416–432. Springer, 2012.

[13] S. G. Choi, J. Katz, A. J. Malozemoff, and V. Zikas. Efficient three-party computation from cut-and-choose. In *International Cryptology Conference*, pages 513–530. Springer, 2014.

[14] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure distributed computing. In J. Kilian, editor, *Proc. of the Second Theory of Cryptography Conference – TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer-Verlag, 2005.

[15] I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings*, pages 378–394, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31870-5. doi: 10.1007/11535218_23. URL http://dx.doi.org/10.1007/11535218_23.

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

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

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

[19] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 225–255. Springer, 2017.

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

[21] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the 19th ACM Symp. on the Theory of Computing*, pages 218–229, 1987.

[22] S. Gueron and M. E. Kounavis. Efficient implementation of the galois counter mode using a carry-less multiplier and a fast reduction algorithm. *Inf. Process. Lett.*, 110(14-15):549–553, 2010.

[23] C. Hazay, P. Scholl, and E. Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Proceedings of the 23rd International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '17, 2017. To appear.

[24] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 294–304, 2000. doi: 10.1109/SFCS.2000.892118.

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

[26] M. Keller, E. Orsini, D. Rotaru, P. Scholl, E. Soria-Vazquez, and S. Vivek. *Faster Secure Multi-party Computation of AES and DES Using Lookup Tables*, pages 229–249. 2017.

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

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

[29] Y. Lindell and B. Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 579–590, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813666. URL http://doi.acm.org/10.1145/2810103.2813666.

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

[31] Y. Lindell, N. P. Smart, and E. Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In M. Hirt and A. Smith, editors, *Theory of Cryptography: 14th International Conference, TCC 2016-B, Proceedings, Part I*, pages 554–581, 2016.

[32] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[33] P. Mohassel, M. Rosulek, and Y. Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 591–602, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813705. URL http://doi.acm.org/10.1145/2810103.2813705.

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

[35] J. B. Nielsen, T. Schneider, and R. Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *Network and Distributed System Security Symposium,NDSS*, 2017.

[36] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 250–267, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-10365-0. doi: 10.1007/978-3-642-10366-7_15. URL http://dx.doi.org/10.1007/978-3-642-10366-7_15.

[37] T. Schneider and M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security*, pages 275–292. Springer, 2013.

[38] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[39] X. Wang, S. Ranellucci, and J. Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *24rd ACM Conference on Computer and Communications Security*, ACM CCS '17, 2017. To appear.

[40] X. Wang, S. Ranellucci, and J. Katz. Global-scale secure multiparty computation. In *24rd ACM Conference on Computer and Communications Security*, ACM CCS '17, 2017. To appear.

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

[42] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 220–250, 2015.