Efficient Scalable Constant-Round MPC via Garbled Circuits

Aner Ben-Efraim[1(✉)], Yehuda Lindell[2], and Eran Omri[3]

[1] Ben-Gurion University, Be'er Sheva, Israel
anermosh@post.bgu.ac.il
[2] Bar-Ilan University, Ramat Gan, Israel
lindell@biu.ac.il
[3] Ariel University, Ariel, Israel
omrier@ariel.ac.il

Abstract. In the setting of secure multiparty computation, a set of mutually distrustful parties carry out a joint computation of their inputs, without revealing anything but the output. Over recent years, there has been tremendous progress towards making secure computation practical, with great success in the two-party case. In contrast, in the multiparty case, progress has been much slower, even for the case of semi-honest adversaries.

In this paper, we consider the case of constant-round multiparty computation, via the garbled circuit approach of BMR (Beaver et al., STOC 1990). In recent work, it was shown that this protocol can be efficiently instantiated for semi-honest adversaries (Ben-Efraim et al., ACM CCS 2016). However, it scales very poorly with the number of parties, since the cost of garbled circuit evaluation is *quadratic* in the number of parties, per gate. Thus, for a large number of parties, it becomes expensive. We present a new way of constructing a BMR-type garbled circuit that can be evaluated with only a *constant* number of operations per gate. Our constructions use key-homomorphic pseudorandom functions (one based on DDH and the other on Ring-LWE) and are concretely efficient. In particular, for a large number of parties (e.g., 100), our new circuit can be evaluated faster than the standard BMR garbled circuit that uses only AES computations. Thus, our protocol is an important step towards achieving concretely efficient large-scale multiparty computation for Internet-like settings (where constant-round protocols are needed due to high latency).

Keywords: Garbled circuits · Constant round MPC · Key-homomorphic PRFs · Concrete efficiency

The first author was 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. The second author is supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The third author is supported by a grant from the Israeli Science and Technology ministry and by a Israel Science Foundation grant 544/13.

1 Introduction

1.1 Background

Protocols for secure multiparty computation enable a set of parties to carry out a joint computation on private inputs, without revealing anything but the output. In the 1980s, powerful feasibility results were presented, showing that any polynomial-time function can be securely computed [9,18,37]. These feasibility hold both for *semi-honest* adversaries (who follow the protocol specification, but try to learn more than allowed by inspecting the transcript), and for *malicious* adversaries who can run any arbitrary adversarial strategy. Furthermore, protocols for constant-round secure computation were demonstrated both for the two-party case [37] and for the multiparty case [5]. These constant-round protocols work by constructing a garbled circuit, which is essentially an encrypted version of the circuit, that can be evaluated obliviously.

Over the past decade, there has been a major research effort to improve the efficiency of secure computation, with great success. For the two-party case, there are highly efficient protocols for both the semi-honest and malicious cases, and following both the garbled-circuit and secret sharing paradigms (see [2,6, 20,21,21,23,30,34,38] for just a handful of examples). As a result, it is possible to run secure two-party computation protocols in practice, for many real-world problems. We remark that a significant portion of the research effort to achieve efficient secure two-party computation focused on the simpler case of semi-honest adversaries. The results for this case proved to be crucial for obtaining efficient protocols for malicious adversaries as well. Thus, the study of efficiency for semi-honest adversaries has proved itself important in the goal of achieving stronger security as well.

In contrast to the aforementioned success for the specific case of two parties, in the setting of *multiparty* secure computation, with strictly more than two parties, progress has been much slower. In particular, for the case of constant-round protocols for many parties, no honest majority, and semi-honest adversaries, the only work has been in [7,8].[1] The recent work of [8] shows that constant-round secure multiparty computation can be achieved with good performance for the case of semi-honest adversaries. However, the technique of BMR [5] for obtaining constant-round protocols via a multiparty garbled circuit has an inherent scalability problem.

[1] There has been work for the malicious setting, with no honest majority [14,25,27], but these protocols are of course much more expensive. Very recently, the work of [19] showed how to extend the results of [8] to the malicious setting with very little overhead, and [36] also presented similar results for the malicious setting using a different protocol. These results suffer from the same scalability problem of [8] that we describe. We argue that it is important to go back to the semi-honest setting and improve efficiency, in order to enable future improvements for the malicious setting.

In addition, there has been work—e.g., in [13]—for the semi-honest setting that follows the GMW paradigm [18] and so has a number of rounds equal to the depth of the circuit being computed. Such protocols can perform very well in very fast networks, but not in Internet-like networks with high latency.

In order to understand this, we first remark that the BMR protocol can be divided into two phases: In the first phase, the parties run a secure protocol to construct a multiparty garbled circuit. This phase can be run even before the inputs are provided, and involves relatively heavy computation in order to securely build the garbled circuit. Then, in the second online phase, after the parties receive their inputs, the parties merely send keys on the input wires associated with their inputs, and then each party can locally evaluate the garbled circuit and obtain output. This paradigm is very attractive since the online phase requires almost no communication, and efficient local computation only. Note that the evaluation of the garbled circuit requires symmetric decryption operations only which are extremely fast in practice using AES-NI instructions. Despite the above, [8] discovered that even the local evaluation computation can become very expensive when the number of parties is large. The reason for this is that each gate requires $O(n^2)$ AES operations, when the number of parties is n. Thus, for a large number of parties—say $n = 100$—the number of AES operations per gate is 10,000. Therefore, the cost of evaluating a BMR garbled circuit for 100 parties is about *10,000 times higher* than the cost of evaluating a two-party (Yao) garbled circuit.

1.2 Our Results

Motivated by the results in [8] and the inherent scalability problem with BMR garbled circuits, our aim in this paper is to construct a variant of the BMR garbled circuit that scales well as the number of parties grow. We remark that if one only focuses on the problem of the cost of the online phase, then the scalability problem can be easily solved. Specifically, one can use any generic multiparty protocol like that of [18] to securely compute a standard Yao two-party garbled circuit, with no party knowing any of the actual keys on the wires (and the parties receiving secret shares of the keys on the input wires). Then, in the online phase, they merely need to exchange shares on the input wires, and can compute the output by evaluating a standard two-party garbled circuit. From a theoretical perspective, this method has many attractive properties; amongst other things, the online time is independent of the number of parties. However, if we are interested in constructing *concretely efficient* protocols that can be implemented and run in practice, then this approach completely fails. The reason for this is that constructing a Yao garbled circuit via multiparty computation is completely unrealistic in practice. This is because the encryption function itself must be computed inside the secure computation, multiple times for every gate.

The above leads us to the following important research goal:

Design a new BMR-type garbled circuit that can be constructed securely with concrete efficiency in the offline phase, and can be efficiently evaluated in the online phase at a cost that is independent of the number of parties.

As discussed above, our goal is *concrete efficiency*, and thus we are interested in obtaining constructions that can be implemented and run in practice, and are faster than previous approaches. Thus, our goal is to obtain a method that is strictly faster than the optimized version of [8] for the case of a large number of parties.

Our method for achieving the above goal utilizes *key-homomorphic pseudorandom functions* (KHPRF), introduced by Naor et al. [31] in the random oracle model, and by Boneh et al. [10] in the standard model. Informally speaking, a pseudorandom function is key-homomorphic if there exist appropriate operations $\widetilde{+}$, $\widetilde{\cdot}$ such that for every pair of keys k_1, k_2 and every input x, it holds that $F_{k_1 \widetilde{+} k_2}(x) = F_{k_1}(x) \widetilde{\cdot} F_{k_2}(x)$. Intuitively, this means that n parties with independent keys $k_1, \ldots, k_n$ can compute $F_K(x)$ for $K = k_1 \widetilde{+} \cdot \widetilde{+} k_n$, by each locally computing $F_{k_i}(x)$, and then using secure computation to compute $F_K(x) = F_{k_1}(x) \widetilde{\cdot} \cdots \widetilde{\cdot} F_{k_n}(x)$. We now informally explain how this can be used to construct a scalable BMR-type circuit.

In a BMR garbled circuit, for every wire w, each party P_i chooses two keys $k^0_{w,i}, k^1_{w,i}$. Then, a garbled gate with input wires u, v and output wire w is constructed by masking all of the keys on the output wire with the appropriate keys on the input wires. For example, in an AND gate, the values $(k^0_{w,1}, \ldots, k^0_{w,n})$ need to be masked with the combinations of $\big((k^0_{u,1}, \ldots, k^0_{u,n}), (k^0_{v,1}, \ldots, k^0_{v,n})\big)$, $\big((k^0_{u,1}, \ldots, k^0_{u,n}), (k^1_{v,1}, \ldots, k^1_{v,n})\big)$, and $\big((k^1_{u,1}, \ldots, k^1_{u,n}), (k^0_{v,1}, \ldots, k^0_{v,n})\big)$, whereas the values $(k^1_{w,1}, \ldots, k^1_{w,n})$ need to be masked with $\big((k^1_{u,1}, \ldots, k^1_{u,n}), (k^1_{v,1}, \ldots, k^1_{v,n})\big)$. This ensures that if the parties have the appropriate keys on the input wires of the gate, then they will obtain the appropriate keys on the output wire of the gate. Now, in order to ensure security, it must be that every *single* key on the input wires suffices to mask the output. Thus, for example, each of $k^0_{u,i}$ and $k^0_{v,i}$ must mask all of $(k^0_{w,1}, \ldots, k^0_{w,n})$. This is achieved by setting the ciphertext $C_{0,0}$, associated with inputs $(0, 0)$, to be

$$C_{0,0} = \left(\bigoplus_{i=1}^{n} F_{k^0_{u,i}}(g\|1)\| \ldots \|F_{k^0_{u,i}}(g\|n) \right) \oplus \left(\bigoplus_{i=1}^{n} F_{k^0_{v,i}}(g\|1)\| \ldots \|F_{k^0_{v,i}}(g\|n) \right) \oplus \left(k^0_{w,1}\| \ldots \|k^0_{w,n}\right),$$

where $\|$ denotes concatenation, g is the gate identity, and F is a pseudorandom function (in practice, AES). Each gate is then constructed as four ciphertexts, for all of the four combinations of input values. Observe that using this method, if party P_i alone is honest, then its single key suffices for masking the output (because the pseudorandom function is used to obtain a long pseudorandom string which masks the keys on the output wire).

Given the above, it is now clear that in order to evaluate a garbled gate, the parties need to invoke the pseudorandom function $2n^2$ times. Specifically, given keys on the inputs wires $\left(k^0_{u,1}\| \ldots \|k^0_{u,n}\right)$ and $\left(k^0_{v,1}\| \ldots \|k^0_{v,n}\right)$, the pseudorandom function is invoked n times for each of the $2n$ keys. Concretely, for $n = 100$, this means that 20,000 pseudorandom computations are made *for every gate*. In the two-party case, only two invocations are needed (or one, using the fixed-key method of [6]).

Now, consider the possibility of constructing ciphertexts as above, but using a key-homomorphic pseudorandom function instead. Concretely, we now define the ciphertext $C_{0,0}$ as follows:

$$C_{0,0} = \left(\widetilde{\Pi}_{i=1}^{n}(F_{k_{u,i}^0}(g))\right) \tilde{\cdot} \left(\widetilde{\Pi}_{i=1}^{n}(F_{k_{v,i}^0}(g))\right) \tilde{\cdot} \left(k_{w,1}^0 \tilde{+} \ldots \tilde{+} k_{w,n}^0\right),$$

where $\tilde{\cdot}$, $\tilde{+}$ are the key-homomorphic operations informally defined above and $\widetilde{\Pi}_{i=1}^{n}(y_i) \overset{\text{def}}{=} y_1 \tilde{\cdot} \ldots \tilde{\cdot} y_n$. Intuitively, such a ciphertext could be securely computed in the offline phase at a cost that is comparable to the original BMR ciphertext, by replacing $\oplus$ with the $\tilde{\cdot}$ operation. Of course, in order to fulfill our goal, the offline phase must also be concretely efficient and thus we do indeed show that this equation can be efficiently computed securely. Now, the important observation is that the ciphertext $C_{0,0}$ above is actually *equal* to

$$C_{0,0} = \left(F_{K_u^0 \tilde{+} K_v^0}(g)\right) \tilde{\cdot} K_w^0$$

where $K_u^0 = k_{u,1}^0 \tilde{+} \ldots \tilde{+} k_{u,n}^0$, $K_v^0 = k_{v,1}^0 \tilde{+} \ldots \tilde{+} k_{v,n}^0$, and $K_w^0 = k_{w,1}^0 \tilde{+} \ldots \tilde{+} k_{w,n}^0$. Thus, the result is a garbled circuit that can be evaluated using only one invocation of the pseudorandom function, *irrespective of the number of parties.* It is important to note that key-homomorphic pseudorandom functions are much more expensive to compute than a plain pseudorandom function. Nevertheless, by implementing and running a comparison with the code of [8], we show that for a large number of parties—say $n = 100$—it is faster to compute a key-homomorphic pseudorandom function than $2n^2$ AES computations (even using AES-NI with a fixed key, as first proposed in [6]).

We mention that [1] used key-homomorphic properties of the LWE-based fully-homomorphic encryption schemes of [11,12] to obtain a secure multiparty protocol (in the CRS model) with only three rounds of interaction and communication complexity that is independent of the underlying function. However, their construction utilizes fully homomorphic encryption, and thus requires parties to locally preform very heavy computation. Furthermore, the intensive part of the computation requires the encryption of the inputs, and therefore done in the online phase. Thus, it is less relevant for the goal of concrete efficiency in the online phase that we consider in this work.

Instantiations and implementation. We present two concrete instantiations of our protocol, using two different key-homomorphic pseudorandom functions; the first is secure under the DDH assumption, and the second is secure assuming Ring-LWE. For each instantiation, we describe a concretely efficient protocol for securely generating the appropriate multiparty garbled circuit in the offline phase. We implemented the online version of our protocols, which is dominated by the local evaluation of the garbled circuit. In Sect. 6, we describe our implementation and results. Figure 1 contains a graph of the online circuit evaluation time for different schemes: BMR refers to the original BMR circuit and is clearly quadratic; all of the other lines are different versions of our protocol, and the

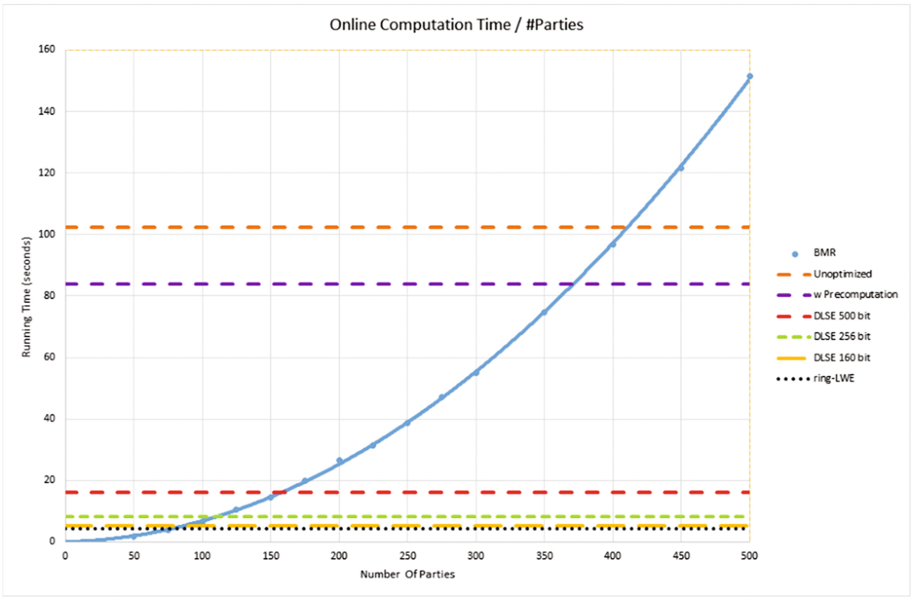

Fig. 1. Online computation time

circuits have evaluation time that is independent of the number of parties. As we mentioned above, this clearly demonstrates that even though the original BMR circuit uses only very fast primitives (AES), it runs slower than all other schemes when enough parties participate.

We remark that the "unoptimized" and "with precomputation" lines refer to a construction based on the standard DDH assumption with a 1024-bit safe prime, whereas the DLSE lines refer to an instantiation based on the assumption that DDH is still hard with *short exponents*; larger primes were also tested and the results appear in Sect. 6. In [24], it was proven that the hardness of DDH with short exponents is implied by the standard DDH assumption and the assumption that the discrete log problem is hard with short exponents. We prove variants of this that are needed for our optimized key-homomorphic pseudorandom function. The "Ring-LWE" refers to our instantiation based on Ring-LWE.

Paper organization. In Sect. 2, we recall the basic definitions required in this work, including those of secure multiparty computation, pseudo-random functions, and the Ring-LWE, DDH and DLSE problems. In Sect. 3, we describe our general paradigm construction, prove its correctness and state our main security theorem. The proof of security will appear in the full version. In Sect. 4, we show how to instantiate our general paradigm based on the DDH problem: In Sect. 4.1, we review the DDH and DLSE problems and prove a few statements. In Sect. 4.2, we describe an instantiation in the random oracle model. In Sect. 4.3, we show an instantiation without a random oracle, and a significantly optimized instantiation based on DDH and DLSE. In Sect. 4.4, we describe two possible offline protocols, one based on the BGW protocol [9] assuming an honest majority, and another that is secure up to $n-1$ corrupt parties in the OT-hybrid model. In Sect. 5, we explain an instantiation based on Ring-LWE: in Sect. 5.1 we explain the online phase. In Sect. 5.2 we explain a "tailored" offline protocol that runs in time quasilinear in the number of parties. In Sect. 6, we provide experimental results of the online computation time of our protocols and a comparison with the online computation time of BMR.

2 Preliminaries

A function $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. We use the abbreviation PPT to denote probabilistic polynomial-time. A *probability ensemble* $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ is an infinite sequence of random variables indexed by κ. Two distribution ensembles $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$ are computationally indistinguishable, denoted $X \overset{c}{\equiv} Y$, if for every PPT D, there exists a negligible function $\mu(\cdot)$ such that for every $\kappa \in \mathbb{N}$, $\Big|\Pr[\mathsf{D}(X_\kappa, 1^\kappa)) = 1] - \Pr[\mathsf{D}(Y_\kappa, 1^\kappa) = 1]\Big| < \mu(\kappa)$.

For a distribution D over a finite set A, we let $x \leftarrow D$ denote the selection of an element $x \in A$ according to distribution D. If D is the uniform distribution over A, we simply write $x \leftarrow A$. For $i \in \mathbb{N}$, we denote by U_i the random variable defined by $x \leftarrow \{0,1\}^i$. For an integer ℓ, we denote $[\ell] = \{1, \ldots, \ell\}$.

2.1 The DLSE and the DDH Problems over Short Exponents

The DLSE (discrete logarithm over short exponents) problem was first introduced in [32]. Following the presentation of this problem in [24], we provide a parameterized version of it. Let $\kappa \in \mathbb{N}$ be the security parameter, let q be a κ-bit prime, and let $c \in \mathbb{N}$ is such that $0 \leq c < \kappa$.

Notation 1. *For $0 \leq c \leq \kappa$, let $R_{\kappa-c} = \{2^{\kappa-c}u \mid 0 \leq 2^{\kappa-c}u < q\}$.*

Specifically, $R_\kappa = \mathbb{Z}_q$. As we will see below, the set $R_{\kappa-c}$ will denote the domain from which exponents are chosen in the short discrete log and DDH problems.

Let $\mathcal{G}$ be a generation algorithm that on input 1^κ returns a triplet $(\mathbb{G}, q, h)$, where $\mathbb{G}$ is a cyclic group of order q (with q of length κ), and h is a generator of $\mathbb{G}$. The discrete-log over short exponents problem, denoted DLSE_c, is defined as follows:

Definition 1 (discrete logarithm over short exponents). *Let $c \in \mathbb{N}$ be a constant. The DLSE_c problem is hard with $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$, such that*

$$\Pr_{(\mathbb{G},q,h) \leftarrow \mathcal{G}(1^\kappa), v \leftarrow R_{\kappa-c}} [\mathcal{A}(\mathbb{G}, q, h, h^v) = v] \leq \mu(\kappa).$$

The standard DL problem is DLSE_c with $c = \kappa$, and the hardness of DLSE_c clearly depends on c. Using similar notation as above, we define the DDH (decisional Diffie-Hellman) problem over short exponents, as first considered by Koshiba and Kurosawa [24].

Definition 2 (decisional Diffie-Hellman over short exponents). *Let $c_1, c_2 \in \mathbb{N}$ be constants. Then, the DDH_{c_1,c_2} problem is hard with $\mathcal{G}$, if*

$$\{(\mathbb{G}, q, h, h^x, h^y, h^{xy})\}_{\kappa\in\mathbb{N}} \overset{\mathrm{c}}{\equiv} \{(\mathbb{G}, q, h, h^x, h^y, h^z)\}_{\kappa\in\mathbb{N}},$$

where the distributions are generated by choosing $(\mathbb{G}, q, h) \leftarrow \mathcal{G}(1^\kappa)$, and then choosing $x, y, z \leftarrow R_{\kappa-c_1} \times R_{\kappa-c_2} \times \mathbb{Z}_q$.

The DDH_{c_1,c_2} is also referred to as the (short-short)-DDH if $c_1 = c_2$ are much smaller than κ, and as the (short-full)-DDH if c_1 is much smaller than κ and $c_2 = \kappa$ (i.e., y is uniform in $\mathbb{Z}_q$). The standard DDH problem is $\mathrm{DDH}_{\kappa,\kappa}$.

In [24], it was shown that if both DDH and DLSE are hard then so are the the (short-full) -DDH and (short-short)-DDH, and further that the converse also holds. That is, amongst other things, they proved the following:

Theorem 2 *[24, Theorem 2]. Let $\mathcal{G}$ be a generation algorithm and let $c \in \mathbb{N}$. If both* DDH *and* DLSE_c *are hard with $\mathcal{G}$, then* $\mathrm{DDH}_{c,\kappa}$ *is hard with $\mathcal{G}$.*

2.2 The Ring-LWE Problem

We briefly state a simple variation of the Ring-LWE hardness assumption. A more complete definition can be found in [28]. Let $p = 2N + 1$ be a prime, where N, called the dimension or security parameter, is a power of two. We fix the polynomial ring $\mathcal{R}_p = \mathbb{Z}_p[X]/(X^N + 1)$, i.e., the polynomials over $\mathbb{Z}_p$ modulo $X^N + 1$.

Definition 3. *The* decisional Ring-LWE *hardness assumption states that it is hard to distinguish between the following two sets of pairs:*

1. $\{(a_i, b_i)\}_{i\in I}$
2. $\{(a_i, a_i \cdot k + e_i)\}_{i\in I}$

where $\{a_i\}_{i\in I}, \{b_i\}_{i\in I}$, and k are chosen uniformly at random from the ring, and $\{e_i\}_{i\in I}$ are chosen from a spherical Gaussian distribution. Furthermore, by transforming to the "Hermite normal form", the assumption holds also if k is chosen from a spherical Gaussian distribution as well.

In general, it is necessary to bound $|I|$, i.e., the number of samples, usually by some $\ell = O(1)$ or $\ell = O(\log N)$. More information, generalizations, and improvements can be found in [28,29].

2.3 Key Homomorphic Pseudorandom Functions

Recall that a function is pseudorandom if no PPT adversary given oracle access to either the function with a randomly chosen key, or to a truly random function, can distinguish between the cases. A weak pseudorandom function is as above, except that the adversary only receives the function output on randomly chosen inputs.

We next recall the definition of key-homomorphic PRFs. Key-homomorphic PRFs were introduced by Naor et al. [31]. The formal definition and first key-homomorphic PRF without a random oracle, based on LWE, were introduced by Boneh et al. [10].

Definition 4. *A family of functions $\{F_k \colon \mathcal{X} \to \mathcal{G}\}_{k \in \mathcal{D}}$ is a family of* key-homomorphic functions *if the key domain, $\mathcal{D}$, and the image, $\mathcal{G}$, are equipped with group operations, $(\mathcal{D}, \widetilde{+})$ and $(\mathcal{G}, \widetilde{\cdot})$, respectively, such that for every $k_1, k_2 \in \mathcal{D}$ and $x \in \mathcal{X}$ it holds that $F_{k_1 \widetilde{+} k_2}(x) = F_{k_1}(x) \,\widetilde{\cdot}\, F_{k_2}(x)$. A* key-homomorphic PRF (KHPRF) *(resp.,* key-homomorphic weak PRF(KHWPRF)*) is a pseudorandom (resp., weak pseudorandom) function that is key-homomorphic.*

Let $\{F_k \colon \mathcal{X} \to \mathcal{G}\}_{k \in \mathcal{D}}$ be a family of key-homomorphic functions. Then, the inverse of an element $h \in \mathcal{G}$ is denoted $(h)^{\widetilde{-1}}$. We denote $\widetilde{\Sigma}_{i=1}^{m}(k_i) \overset{\text{def}}{=} k_1 \,\widetilde{+}\, \cdots \,\widetilde{+}\, k_m$ and $\widetilde{\Pi}_{i=1}^{m}(h_i) \overset{\text{def}}{=} h_1 \,\widetilde{\cdot}\, \cdots \,\widetilde{\cdot}\, h_m$.

In general, it may be the case that $\mathcal{D} \neq \mathcal{G}$. As we shall see, this can pose some difficulty because keys from $\mathcal{D}$ are encrypted in $\mathcal{G}$. Furthermore, we also need to encrypt the bit necessary for point-and-permute. Therefore, we will assume throughout the existence of an efficiently computable function $f \colon \mathcal{D} \times \{0,1\} \to \mathcal{G}$, with an efficiently computable inverse f^{-1}. We note that for all known KHPRFs, such a function exists.

2.4 Secure Multiparty Computation

We follow the standard definition of secure multiparty computation for semi-honest adversaries, as it appears in [17]. In brief, an n-party protocol π is defined by n interactive probabilistic polynomial-time Turing machines $\mathrm{P}_1, \ldots, \mathrm{P}_n$, called parties. The parties hold the security parameter 1^κ as their joint input and each party P_i holds a private input x_i. The computation proceeds in rounds. In each round j of the protocol, each party sends a message to each of the other parties (and receives messages from all other parties). The number of rounds in the protocol is expressed as some function $r(\kappa)$ in the security parameter.

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

Similarly, for a set of parties with indices $I \subseteq [n]$, we denote by $\boldsymbol{x}_I$ the set of their inputs, by $\text{view}_I^\pi(\boldsymbol{x}, 1^\kappa)$ their joint view, and by $\text{Output}_I^\pi(\boldsymbol{x}, 1^\kappa)$ their joint output. In the setting of this work, it suffices to consider deterministic functionalities. We therefore provide the definition of security only for deterministic functionalities; see [17] for a motivating discussion regarding the definition.

Definition 5 (security for deterministic functionalities). *A protocol π t-securely computes a deterministic functionality $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ in the presence of semi-honest adversaries if the following hold:*

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

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

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

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

$$\{\mathcal{S}_A(\boldsymbol{x}_I, f_I(\boldsymbol{x}), 1^\kappa)\}_{\boldsymbol{x}\in(\{0,1\}^*)^n;\kappa\in\mathbb{N}} \overset{C}{\equiv} \{\text{view}_I^\pi(\boldsymbol{x}, 1^\kappa)\}_{\boldsymbol{x}\in(\{0,1\}^*)^n;\kappa\in\mathbb{N}}. \tag{2}$$

3 Multiparty Garbled Circuits via Key-Homomorphic PRFs

In this section, we describe a general paradigm for constructing garbled circuits for many parties via key-homomorphic PRFs. As explained above, the goal is to allow parties to efficiently construct a multiparty garbled circuit in which the number of decryptions per gate and the size of the decryption keys are independent of the number of parties.

In Sect. 3.1 we define our main offline functionality, $\mathcal{F}_{GC}$, that constructs a garbled circuit from a key-homomorphic PRF. In Sect. 3.2 we describe an online protocol for computing outputs using $\mathcal{F}_{GC}$. In Sect. 3.3 we show the correctness of the protocol and we state conditions when $\mathcal{F}_{GC}$ is secure, i.e., when a secure implementation of F_{GC}, along with the online protocol, can be used as a secure multiparty protocol for computing any boolean circuit. The proof of security will appear in the full version.

In the following, let C denote the Boolean circuit and let $|C|$ denote the number of gates in C. For every gate g in C, we let $g(\alpha, \beta)$ denote the gate operation on $\alpha, \beta \in \{0,1\}$. We further abuse notation by also using g to denote the index of the gate in a fixed topological ordering of C.

The parties are denoted $P_1, \ldots, P_n$, where n denotes the number of parties. The set of all wires in the circuit is denoted by W. For each wire $\omega \in W$ each party P_i will hold two keys $k_{\omega,0}^i, k_{\omega,1}^i \in \mathcal{D}$, which we will call the *individual keys*. The "summations" of the individual keys, $k_{w,0} \overset{\text{def}}{=} \widetilde{\Sigma}_{i=1}^n(k_{w,0}^i)$ and $k_{w,1} \overset{\text{def}}{=} \widetilde{\Sigma}_{i=1}^n(k_{w,1}^i)$, will be termed the *joint keys*.

We associate a hidden bit λ_ω to each wire $\omega \in W$, generally unknown to all parties. During the online phase, the parties reveal an external value bit e_ω for each wire ω. It will later become clear that $e_\omega = \lambda_\omega \oplus t_\omega$, where t_ω is the real bit of the wire ω in an ungarbled computation of C for the inputs provided by the parties.

3.1 The Offline Phase

In this section, we describe the main functionality of the offline phase of our protocol, Functionality $\mathcal{F}_{GC}$. This functionality constructs a garbled circuit from a family of key-homomorphic PRFs.

Functionality $\mathcal{F}_{GC}$ receives as public input a circuit C with wires W, a family of key-homomorphic functions $\{F_k\}$, and a set $x_0, \ldots, x_{4|C|-1} \in \mathcal{X}$. The functionality also receives from each party P_i two individual keys, $k^i_{\omega,0}$ and $k^i_{\omega,1}$, for each wire $\omega \in W$. The functionality associates to each wire $\omega \in W$ a hidden bit λ_ω, that is generally not revealed to any of the parties. The functionality then computes and outputs to the parties a version of Yao's garbled circuit, in which the keys for the encryption are the joint keys. More precisely, for each gate $g \in C$ with input wires u, v and output wire w and for every $\alpha, \beta \in \{0, 1\}$, the functionality secretly computes $e_{w,\alpha,\beta} \overset{\text{def}}{=} g((\lambda_u \oplus \alpha), (\lambda_v \oplus \beta)) \oplus \lambda_w$ and $KEY_{w,\alpha,\beta} \overset{\text{def}}{=} \begin{cases} k_{w,0} & e_{w,\alpha,\beta} = 0 \\ k_{w,1} & e_{w,\alpha,\beta} = 1. \end{cases}$ The value $e_{w,\alpha,\beta}$ is equal to the external value of the wire that would be revealed to the parties during evaluation, if the external values of u and v are α and β respectively. $KEY_{w,\alpha,\beta}$ is the joint key that corresponds to this external value.

The functionality outputs to all parties the garbled gates

$$\tilde{g}_{\alpha,\beta} = \left(F_{k_{u,\alpha} \tilde{+} k_{v,\beta}}(x_{4g+2\alpha+\beta})\right)^{\widetilde{-1}} \tilde{\cdot} f(KEY_{w,\alpha,\beta}, e_{w,\alpha,\beta}).$$

The functionality also outputs to each party the λ's associated with its input and output wires. The full details of Functionality $\mathcal{F}_{GC}$ appear in Fig. 2.

3.2 The Online Phase

In this section, we describe our online phase protocol. In this phase, the parties exchange garbled inputs and compute the output of the garbled circuit. The general flow of the online phase is the same as in the BMR protocol. Specifically, it contains two short rounds of communication, in which the parties learn the keys for input wires. From then on, all computations are done locally.

In the first step of the protocol, each party computes and broadcasts the external values $e_\omega = \lambda_\omega \oplus b_\omega$ for each of its input wires ω. This is possible, as each party knows the real value b_ω of each of its input wires, and the party knows λ_ω of its input wires from the output of functionality $\mathcal{F}_{GC}$.

In the next step, the parties broadcast their individual keys corresponding to the external values for each of the input wires and compute the joint keys

Functionality $\mathcal{F}_{GC}$ for Constructing the Garbled Circuit

Public Inputs: a circuit C with wires W, a family of key-homomorphic functions $\{F_k\}$ and values $x_0, \ldots, x_{4|C|-1} \in \mathcal{X}$.
Private Inputs: Each party P_i gives as input two keys $k^i_{w,0}, k^i_{w,1} \in \mathcal{D}$ for every wire $w \in W$.
Denote $k_{w,0} \overset{\text{def}}{=} \widetilde{\Sigma}^n_{i=1}(k^i_{w,0})$ and $k_{w,1} \overset{\text{def}}{=} \widetilde{\Sigma}^n_{i=1}(k^i_{w,1})$.
Random Input: To each wire $\omega \in W$ of the circuit, a hidden random bit $\lambda_\omega \in_R \{0,1\}$ is associated.
Output: The functionality outputs to all parties the garbled circuit GC – for every gate $g \in C$, with input wires u, v, and output wire w, and for every $\alpha, \beta \in \{0,1\}$, it outputs:

$$\tilde{g}_{\alpha,\beta} = \left(F_{k_{u,\alpha} \,\widetilde{+}\, k_{v,\beta}}(x_{4g+2\alpha+\beta})\right)^{\widetilde{-1}} \,\widetilde{\cdot}\, f(KEY_{w,\alpha,\beta}, e_{w,\alpha,\beta}) \tag{3}$$

Where

$$e_{w,\alpha,\beta} \overset{\text{def}}{=} g((\lambda_u \oplus \alpha), (\lambda_v \oplus \beta)) \oplus \lambda_w \tag{4}$$

and

$$KEY_{w,\alpha,\beta} \overset{\text{def}}{=} k_{w,e_{w,\alpha,\beta}}. \tag{5}$$

Notice that

$$\begin{aligned} F_{k_{u,\alpha} \,\widetilde{+}\, k_{v,\beta}}(x_{4g+2\alpha+\beta}) &= F_{\widetilde{\Sigma}^n_{i=1}(k^i_{u,\alpha} \,\widetilde{+}\, k^i_{v,\beta})}(x_{4g+2\alpha+\beta}) \\ &= \widetilde{\Pi}^n_{i=1}(F_{k^i_{u,\alpha} \,\widetilde{+}\, k^i_{u,\beta}}(x_{4g+2\alpha+\beta})). \end{aligned} \tag{6}$$

In addition, each party P_i receives the λ's corresponding to its input and output wires.

Fig. 2. Functionality$\mathcal{F}_{GC}$

for these wires. Then, each party locally decrypts the correct row at each gate in topological order, and recovers the key and external value for the output wire of that gate. In this way, at the end of this step the parties recover the external values of the output wires. Finally, the parties then recover the real outputs of the function by XORing the output external values with the corresponding λ's, which they know from the output of functionality $\mathcal{F}_{GC}$.

3.3 Correctness and Security

In this section we show the correctness of using the online protocol in the $\mathcal{F}_{GC}$ hybrid model, and state condition under which this results in a secure multiparty protocol for any Boolean circuit. The proof of security is deferred to the full version.

Online Protocol

Public Inputs: a circuit C with wires W, a family of key-homomorphic functions $\{F_k\}$ and values $x_0, \ldots, x_{4|C|-1} \in \mathcal{X}$.
Private Inputs: All parties hold their input to the functionality F_{GC} and the output they received from it.
In addidition, each party holds its input bit b_ω for each of its input wires ω.
Computation:

1. Each party broadcasts $e_\omega = \lambda_\omega \oplus b_\omega$ for the each of its input wires ω.
2. Each party P_i broadcasts k^i_{ω,e_ω} for each input wire of the circuit.
3. Each party does the following computation locally:
   (a) Computes $k_{\omega,e_\omega} = \widetilde{\Sigma}^n_{i=1}(k^i_{\omega,e_\omega})$ for every input wire ω of the circuit.
   (b) In topological order, for each gate $g \in C$, with input wires u, v and output wire w computes
   $$(k_{w,e_w}, e_w) = f^{-1}(F_{k_{u,e_u} \tilde{+} k_{v,e_v}}(x_{4g+2e_u+e_v}) \tilde{\cdot} \tilde{g}_{e_u,e_v}), \tag{7}$$
   and recovers the external value e_w and corresponding key k_{w,e_w} for the output wire w of the gate.
   (c) For each of its output wires ω computes the true value of the output $e_\omega \oplus \lambda_\omega$.

Output: Each party recovers the true value of the output, $e_\omega \oplus \lambda_\omega$, for each of its output wires ω.

Fig. 3. Online protocol

Correctness. We now show that the outputs received by the parties from the online phase corresponds to the correct outputs. By Step 3c of the online protocol, it follows from the following claim:

Claim 3. *For each $\omega \in W$, the external value e_ω revealed by the parties in the online protocol is equal to $\lambda_\omega \oplus t_\omega$, where t_ω is the true value of the wire ω in an ungarbled computation with the same inputs.*

The proof is by induction on the topological ordering of C. We give a sketch of the proof, omitting the proof of correctness of the decryption in Step 3b.

Proof Sketch. For the input wires, $e_\omega = \lambda_\omega \oplus t_\omega$ follows from Step 1 of the online protocol. For any other wire $\omega \in W$ that is the output wire of gate g, with input wires u, v, we have from Step 3b of the online protocol, Eqs. (3) and (4) in functionality $\mathcal{F}_{GC}$, and the induction assumption, that

$$e_\omega = e_{\omega,e_u,e_v} = g((\lambda_u \oplus e_u), (\lambda_v \oplus e_v)) \oplus \lambda_\omega = g(t_u, t_v) \oplus \lambda_\omega = t_\omega \oplus \lambda_\omega. \qquad \square$$

Security. We state conditions when a secure implementation of $\mathcal{F}_{GC}$, along with the online protocol, can be used as a secure multiparty protocol to compute any Boolean circuit C. We first give two definitions. Let $\{F_k \colon \mathcal{X} \to \mathcal{G}\}_{k \in \mathcal{D}}$ be a family of functions, and let $R \colon \mathcal{X} \to \mathcal{G}$ be random functions.

Definition 6. *For $x_1, \ldots, x_n \in \mathcal{X}$, we say that* Property $I(x_1, \ldots, x_n)$ *holds for* $\{F_k\}$ *if for a randomly chosen $k \in \mathcal{D}$, it holds that*

$$\{(x_i, F_k(x_i))\}_{i=1}^{n} \overset{c}{\equiv} \{(x_i, R(x_i))\}_{i=1}^{n} . \tag{8}$$

Note that if $\{F_k\}$ is a PRF then Property $I(x_1, \ldots, x_n)$ holds for *any* choice of $x_1, \ldots, x_n$. If $x_1, \ldots, x_n$ are *random*, then Property $I(x_1, \ldots, x_n)$ holds also if $\{F_k\}$ is a weak PRF.

Let C be a Boolean circuit with set of wires W.

Definition 7. *For $x_0, \ldots, x_{4|C|-1} \in \mathcal{X}$ (possibly with $x_i = x_j$ for $i \neq j$), we say that* Property $J(x_0, \ldots, x_{4|C|-1})$ *holds for C if for every gate $g \in C$ it holds that $\{x_{4g}, x_{4g+1}, x_{4g+2}, x_{4g+3}\}$ are all distinct and for every wire $\omega \in W$ that is an input wire to two different gates $g_1, g_2 \in C$, it holds that*

$$\{x_i\}_{i=4g_1}^{4g_1+3} \cap \{x_i\}_{i=4g_2}^{4g_2+3} = \emptyset. \tag{9}$$

The idea of Definition 7 is that if two gates share a common input wire, they do not share the same x's. This is to ensure that the same PRF is not queried twice with the same input. We are now ready to state our main security theorem.

Theorem 4. *If Property $I(x_0, \ldots, x_{4|C|-1})$ holds for $\{F_k\}$ and Property $J(x_0, \ldots, x_{4|C|-1})$ holds for C, then the online protocol in Fig. 3 securely computes C in the semi-honest $\mathcal{F}_{GC}$-hybrid model with up to $n-1$ corrupt parties.*

Theorem 4 can be proved in a standard way, similar to the proofs of [26,27]. We will give a slightly different proof in the full version.

Remark 1. An important observation from the proof is that the security of the protocol relies on the individual keys, and not the joint keys. Intuitively, this is because the adversary knows many of the individual keys, and could thus, in certain circumstances, learn partial information on the joint key. For example, this observation is important in our DDH instantiation, where we restrict the individual keys. It is also important in our LWE instantiation, as we shall see in Sect. 5.

4 Explicit Instantiation Based on DDH

In this section we describe an explicit instantiation of our key-homomorphic PRF based MPC protocol that relies on the DDH assumption. The resulting encryption scheme used in the garbling can be seen as a variant of the ElGamal scheme. However, using the ElGamal scheme naïvely can be insecure. We show a simple example demonstrating this in the full version.

In Sect. 4.1, we prove some properties of the DDH problem and the DLSE (Discrete Log over Short Exponents) problem. In Sect. 4.2, we explain our instantiation relying on DDH and a Random Oracle. In Sect. 4.3, we explain how to remove the Random Oracle and also describe some optimizations. Some of these optimizations explicitly rely on the hardness of DLSE. In Sect. 4.4, we describe two possible offline protocols for our DDH instantiation: in Sect. 4.4.1 we describe a protocol based on BGW that is secure when assuming an honest majority. In Sect. 4.4.2 we describe an OT based protocol that is secure against up to $n-1$ corrupt parties.

4.1 Some Properties of DDH and DLSE

In this section we state a couple of properties of DDH and DLSE that we use in our instantiation. The proofs use theorems from [24,31], and will be supplied in the full version.

Notation 5. *For* $0 \leq c \leq \kappa$, *let* $\widehat{R}_{\kappa-c} = \{u \mid 0 \leq 2^{\kappa-c}u < q\}$.

Notice that this notation differs from Notation 1 because here the most significant bits are zeros. However, it turns out that in prime order groups, the hardness of DDH with short exponenents is equivalent using Notation 5 or Notation 1. This enables us to prove the following variations of a theorem from [31].

Theorem 6. *Assuming the* DDH *problem is hard,*

1. *If* $c = O(\log \kappa)$ *then* $\left\{F_k(x) = x^k\right\}_{k \in \widehat{R}_{\kappa-c}}$ *is a key-homomorphic weak* PRF.
2. *If the* DLSE$_c$ *problem is hard then* $\left\{F_k(x) = x^k\right\}_{k \in \widehat{R}_{\kappa-c}}$ *is a key-homomorphic weak* PRF.

Furthermore, if H is a random oracle whose images are generators of the group, then $\left\{F_k(x) = H(x)^k\right\}_{k \in \widehat{R}_{\kappa-c}}$ *is a key-homomorphic* PRF *in the above two cases.*

4.2 Concrete Instantiation

In this section we describe a concrete instantiation of our protocol, based on the hardness of DDH. We first assume also that using SHA256 as described below is a random oracle. In Sect. 4.3, we explain how to avoid the random oracle assumption.

Our concrete implementation of our protocol is as follows. Let $p = 2q+1$ be a safe prime with κ bits, and denote by $\mathbb{G}_q$ the subgroup of order q of the multiplicative group $\mathbb{Z}_p{}^*$, i.e., the group of quadratic residues. We let our PRF family $\{F_k\}$ be $F_k(x) = (H(x))^k \mod p$, where H is a random oracle modeled by $H(x) \overset{\text{def}}{=} (\text{SHA}(r||x\ell)|| \cdots ||\text{SHA}(r||(x\ell + (\ell-1))))^2 \mod p$ with $\ell = \left\lceil \frac{\kappa}{256} \right\rceil$, and r an agreed random nonce. Notice that $H(x)$ is a generator of $\mathbb{G}_q$.[2] The domain of the individual keys is initially $\mathbb{Z}_q$, but we slightly modify this below. We define $x_i = i$, $a \,\tilde{\cdot}\, b = a \cdot b \mod p$, $a \,\tilde{+}\, b = a + b \mod q$, and $f(a) = a^2 \mod p$, which is in $\mathbb{G}_q$ for any non-zero element $a \in \mathbb{Z}_p$.

We note some difficulties for using the instantiation above.

[2] The only possible exceptions are the values 0 and 1, but this happens with negligible probability.

1. The input of the function f should contain both the key and the external value.
2. The squaring function has 2 inverses for every entry. Therefore, as f needs to be invertible, there must be some way for the parties to decide which of the 2 inverses is correct.
3. The joint key is not well defined. Initially, the individual keys are drawn from $\mathbb{Z}_q$. The summation of the keys in the message to be encrypted will be modulo p, while the summation of the keys in the exponent will be modulo q (i.e., the size of the subgroup of the generator).
   Since the keys are chosen randomly, the summation of the individual keys of the parties in the message, which is the joint key revealed during the online phase, and the joint key in the exponent, which is the real key used for encryption/decryption, will, in general, not be equal. This will cause errors in the computation.

We solve the above problems by letting the individual keys be drawn from $\{0, \ldots, \frac{q-1}{4n}\} \subset \mathbb{Z}_q$. First, notice that the summation of the keys is now well defined as $k_{w,0} + k_{w,1} = \Sigma_{i=1}^{n} k_{w,0}^{i} + \Sigma_{i=1}^{n} k_{w,1}^{i} < q < p$ so it is equal modulo q and modulo p. Therefore, the key used for the decryption is equal to the sum of the keys that are revealed from the decryption of the input wires. By Theorem 6, Item 1, this choice of keys still renders our protocol secure.

To insert the external value bit, we multiply the key by two and add the external value bit, i.e., $f(KEY_{w,\alpha,\beta}, e_{w,\alpha,\beta}) = (2KEY_{w,\alpha,\beta} + e_{w,\alpha,\beta})^2$. Notice that $2KEY_{w,\alpha,\beta} + e_{w,\alpha,\beta} < \frac{q-1}{2}$. Restricted to this domain, the squaring function has a unique inverse in $\mathbb{Z}_p$, which can be computed in polynomial time.

The instantiated functionality $\mathcal{F}_{GC}$ and the online protocol are simply applying the above concrete instantiation, i.e., the key-homomorphic PRF based on DDH, to the respective figures in Sect. 3.

4.3 Removing the Random Oracle and Optimizations

In this section we first show how to remove the Random Oracle assumption. Then, we describe optimizations, some of which need to further assume DLSE$_c$ for sufficiently small c.

Removing the random oracle. Recall that the use of the random oracle was necessary to show that $\{F_k\}$ is a PRF family. However, Theorem 4 does not explicitly require that $\{F_k\}$ be a PRF. If $x_i = h_i$ are agreed *random distinct* generators of $\mathbb{G}_q$, then defining $F_k(a) = a^k$ is sufficient: $\{F_k\}$ is a weak PRF, and thus Property $I(x_1, \ldots, x_{4|C|-1})$ holds because the generators are random. Property $J(x_1, \ldots, x_{4|C|-1})$ also holds because all the generators are distinct.

Furthermore, it is possible to require less than $4|C|$ distinct random generators – we can reuse the same random generators (i.e., $x_i = h_{r_i}$ where $r_i = r_j$ may happen for $i \neq j$), as long as Property $J(x_1, \ldots, x_{4|C|-1})$ also holds. I.e., as long as for every random generator h and individual key k, the pseudorandom value $F_k(h)$ is used only once for encryption (Eq. (3)) in all gates.

We note that the same individual key is always used more than once, both in two rows of each gate, and also in any other gate with the same input wire. However, we can bound the number of necessary random generators we need by 8 times the maximal fan-out of the circuit. The proof will be given in the full version. The proof gives a simple deterministic (with respect to the circuit and topological ordering) algorithm for deciding which generators to use at each gate. Furthermore, these agreed generators can be chosen once and used repeatedly for many different circuits.

We now move on to describing some optimizations to the above protocol. There are two main bottlenecks for the online computation time – computing the exponentiations h^k, and computing square root modulo p to recover the key and external value. We next show how to optimize these steps.

Precomputation. We point out that the set of generators used is already known in the offline phase. Thus, to speed up the exponentiations, the parties can precompute $h^2, h^4, , ..., h^{2^{\lfloor \log q \rfloor}}$ for each generator h. If the number of generators is small, (e.g., in the version without the random oracle described above), this will also significantly speed up the offline phase. If the same generators are used for multiple circuits then this precomputation can be done only once for all the circuits.

Optimizations based on hardness of DLSE. We now show how, by further assuming the DLSE$_c$ assumption for sufficiently small c, we can significantly improve the computation time. Clearly, assuming the keys are short significantly improves the time of exponentiations – if the domain of the keys is R_{n-c} then the exponentiations are approx. $\frac{\kappa}{c}$ times faster.

A second and less obvious optimization is that assuming DLSE$_c$ with $c < \frac{\kappa}{2} - \log n - 2$ significantly shortens the time of modular square root. This follows from the following observation.

Observation 7. *If $m \in \mathbb{N}$ is d bits long with $d < \frac{\kappa}{2}$, then $m^2 \mod p = m^2$.*

Following Observation 7, if the keys are short enough, we can replace taking square root modulo p by taking regular square root. The time of computing square root is several orders of magnitude faster than computing square root modulo p. Thus, this practically removes the time it takes to compute modular square root.

Remark 2. If we assume both DLSE and that the set of generators is small, then for computing $(h^k)^{-1}$ for a generator h and key k (such as needed in our offline protocols in Sect. 4.4), it is more efficient to compute exponentiation by a short key (k). This can be done via the equality $(h^k)^{-1} = (h^{-1})^k$, i.e., precomputing the tables also for the inverse of every generator h (of course, this makes sense only if the number of generators is small, as in the version without the random oracle).

4.4 Offline Phase Protocols for DDH Based Implementation

We now describe two different secure protocols for computing the offline phase of the instantiation based on DDH, one based on BGW [9] that requires an honest majority and another based on oblivious transfer that is secure up to $n-1$ corrupt parties.

The offline protocol for computing the version without the random oracle is identical except that in all locations $H(x_i)$ is replaced by h_{r_i} as explained in Sect. 4.3. If one assumes DLSE$_c$, then the individual keys should be drawn from $\{0, \dots, \frac{q}{2^{\kappa-c}}\}$.

The current bottleneck in both suggested protocols is computing unbounded fan-in multiplication, i.e., computing Shamir or additive shares of $\Pi_{i=1}^{n} m_i$, where n is the number of parties and m_i is known only to party i. Thus, any improvement to protocols computing unbounded fan-in multiplication immediately implies an improvement to our offline protocols. Currently, the best constant round protocol for computing unbounded fan-in multiplication is the protocol given by Bar-Ilan and Beaver [4].

4.4.1 BGW Based Offline Protocol for the DDH Based F_{GC}

In this section we describe an offline protocol for our DDH instantiation that is based on the BGW protocol [9], which requires an honest majority. The running time of the described protocol is comparable to the running time of BGW based protocols for the offline phase of the BMR circuit, e.g., [7,8], when the number of parties is *large*.

Our BGW offline protocol is achieved by secret-sharing both the individual keys k^i and the exponentiations g^{-k_i} in Shamir secret-sharing, and then using the BGW protocol to compute the garbled gates

$$\tilde{g}_{\alpha,\beta} = \left(\left((H(4g+2\alpha+\beta))^{k_{u,\alpha}+k_{v,\beta}}\right)^{-1} \cdot (2KEY_{w,\alpha,\beta} + e_{w,\alpha,\beta})^2\right). \tag{10}$$

The main protocol is given in Fig. 4. The subprotocols are standard using the BGW protocol. Note that $\Pi_{i=1}^{n} m_i$ is computed in constant rounds using [4]. The protocol in Fig. 4 is based entirely on BGW, and therefore securely computes the functionality F_{GC} in the semi-honest model, assuming an honest majority.

4.4.2 OT Based Offline Protocol for the DDH Based F_{GC}

In this Section we describe a protocol for computing the DDH based functionality F_{GC} that is secure up to $n-1$ corrupt parties in the OT-hybrid model.

The basic observation for the OT protocol is that if two (not necessarily disjoint) sets of parties $\mathcal{P}_1$ and $\mathcal{P}_2$ hold additive shares (in $\mathbb{Z}_p$) of secrets $s_1, s_2 \in \mathbb{Z}_p$ respectively, then using one OT round, the parties can compute an additive sharing of $s_1 \cdot s_2$ amongst the set of parties $\mathcal{P}_1 \cup \mathcal{P}_2$.

Offline Protocol for DDH Instantiation Based on BGW

Inputs: All parties hold the circuit C, the number of parties n, and the prime field $\mathbb{Z}_p$.
Computation: The parties perform the following computations, where all shares and computations, including those in the sub-protocols, are done over $\mathbb{Z}_p$.

1. For every $\omega \in W$
   (a) Each party P_i randomly selects its individual keys $k^i_{w,0}, k^i_{w,1} \in \left\{0, \ldots, \frac{q-1}{4n}\right\} \subset \mathbb{Z}_q \subset \mathbb{Z}_p$. We denote the joint keys by $k_{w,0}, k_{w,1} \in \mathbb{Z}_q$.
   (b) Using two rounds of interaction, the parties run a coin-tossing protocol and receive Shamir shares of the hidden permutation bit $\lambda_w \in \{0,1\}$[7].
2. For every gate $g \in C$ with input wires u, v and output wire w, and for every $\alpha, \beta \in \{0,1\}$, do:
   (a) Locally compute $m_i = (H(4g+2\alpha+\beta))^{-\left(k^i_{u,\alpha}+k^i_{v,\beta}\right)}$, and secret share m_i in a t-out-of-n Shamir secret-sharing scheme..
   (b) Secret share $k^i_{w,0}, k^i_{w,1}$ in a t-out-of-n Shamir secret-sharing scheme. By summing the received shares, each party recovers a share of $k_{w,0}$ and $k_{w,1}$.
   (c) Using standard sub-protocols, compute Shamir shares of $\Pi^n_{i=1} m_i = \left((H(4g+2\alpha+\beta))^{k_{u,\alpha}+k_{v,\beta}}\right)^{-1}$ and of $(2KEY_{w,\alpha,\beta} + e_{w,\alpha,\beta})^2$.
   (d) Using a single BGW round, compute shares of
   $$\tilde{g}_{\alpha,\beta} = \left((H(4g+2\alpha+\beta))^{k_{u,\alpha}+k_{v,\beta}}\right)^{-1} \cdot (2KEY_{w,\alpha,\beta} + e_{w,\alpha,\beta})^2 .$$
   (e) Reconstruct $\tilde{g}_{\alpha,\beta}$, i.e., broadcast shares of $\tilde{g}_{\alpha,\beta}$ and interpolate.
3. Reconstruct λ_ω for every output wire ω of the circuit.

Outputs: $\tilde{g}_{\alpha,\beta}$ for each gate $g \in C$ and every $\alpha, \beta \in \{0,1\}$ and λ_ω for every output wire ω of the circuit.

Fig. 4. BGW protocol

To show this, we denote $s_1 = s_1^{i_1} + \cdots + s_1^{i_{|\mathcal{P}_1|}}$ and $s_2 = s_2^{j_1} + \cdots + s_2^{j_{|\mathcal{P}_2|}}$. The observation then follows by noticing that

$$\begin{aligned} s_1 \cdot s_2 &= (s_1^{i_1} + \cdots + s_1^{i_{|\mathcal{P}_1|}})(s_2^{j_1} + \cdots + s_2^{j_{|\mathcal{P}_2|}}) \\ &= s_1^{i_1} \cdot s_2^{j_1} + \cdots + s_1^{i_1} \cdot s_2^{j_{|\mathcal{P}_2|}} + \cdots + s_1^{i_{|\mathcal{P}_1|}} \cdot s_2^{j_{|\mathcal{P}_2|}}. \end{aligned}$$

Any multiplication of the form $s_1^i \cdot s_2^j$ can be performed using $\log p$ string OTs between parties P_i and P_j, as explained in [22].

The OT protocol for the offline phase is similar to the BGW based protocol, except that shares are additive, and the multiplications are computed using the above observation. As in the BGW based protocol, the main bottleneck of the protocol is to compute unbounded fan-in multiplication.

5 Instantiation Based on Ring-LWE

Boneh et al. [10] and Banerjee and Peikert [3] constructed almost key-homomorphic PRFs from LWE and ring-LWE respectively. It seems quite possible that one can use these almost key-homomorphic PRFs in our construction.

We go in a slightly different route – we build a protocol based directly on decisional ring-LWE hardness assumption. The function we use is not a real PRF, as it is not deterministic. However, by the decision ring-LWE assumption it is indistinguishable from random, which we show is sufficient for our construction.

Let $p = 2N + 1$ be a prime where N is a power of two, and denote $\mathcal{R}_p = \mathbb{Z}_p[X]/(X^N + 1)$. We define $\mathcal{F} = \{f_k \colon \mathcal{R}_p \rightarrow \mathcal{R}_p | f_k(a) = a \cdot k + e\}$, where a, k, and e are polynomials in the ring and the coefficients of e come from a gaussian distribuition $\mathcal{D}$. Assuming decision ring-LWE, for a *bounded constant* number of *distinct random* inputs, the output of f_k is indistinguishable from random. Furthermore, the above holds also if the keys themselves come from Gaussian distribution. Notice that the output of f_k is not deterministic due to the error.

Since in this protocol the key domain is a subset of the image of f_k, it might seem at first that the function f used to map the keys into the image of f_k can be the identity. However, this would be problematic due to the error. To avoid the error, the function f multiplies the coefficients of the key by $\lceil\sqrt{p}\rceil$, see Sect. 5.1.

The proof of security for the LWE instantiation is similar to the proof of the DDH instantiation without the random oracle. Notice that the proof did not require the PRF to be deterministic, only that the function is indistiguishable from random, and that the decrypted messages can be recovered correctly. For the latter point, by using correct parameters, this happens with overwhelming probability. It is important to note that for the encryption, the function is never queried twice on the same input.

In order to encrypt also the external values, we set the last coordinate of the key to be 0. Thus, we lose one dimension of the key, which slightly reduces security. We give a more detailed explanation on the security in the full version.

In the following protocols, using the ideas explained in Sect. 4.3 for removing the random-oracle, we let $a_1, \ldots, a_{8 \cdot f_{out}}$ be public random elements of the ring (which is also public), where f_{out} is the maximal fan-out of the circuit. We denote by $A(g, \alpha, \beta)$ the random element associated with row (α, β) of gate g, such that any two gates that share an input wire do not share any of the random elements (cf. Definition 7). The full description of functionality F_{GC}, instantiated as described above, appears in Fig. 5. Notice that for security, it must hold that $8 \cdot f_{out}$ is less than the bound on the number of samples.

5.1 Online Phase for Ring-LWE Based Instantiation

The online phase of our LWE based instantiation follows the general online phase, except that after each decryption, the error needs to be eliminated before the hidden key and external value can be recovered. The main idea for eliminating the error is that both the error and the key come from a Gaussian distribution. Thus, they will be far from the mean with only negligible probability.

Instantiated Functionality $\mathcal{F}_{GC}$ Based on LWE

Private Inputs: Each party P_i gives as input two keys $k^i_{w,0}, k^i_{w,1} \in \mathcal{D}^{N-1} \times \{0\}$ and four noise vectors $E^i_{g,\alpha,\beta} \leftarrow \mathcal{D}^N$ for every wire $w \in W$, where $\alpha, \beta \in \{0,1\}$.

Denote $k_{w,0} \stackrel{\text{def}}{=} \Sigma^n_{i=1} k^i_{w,0}$ and $k_{w,1} \stackrel{\text{def}}{=} \Sigma^n_{i=1} k^i_{w,1}$.

Random Input: To each wire $\omega \in W$ of the circuit, a hidden random bit $\lambda_\omega \in_R \{0,1\}$ is associated.

Output: The functionality outputs to all parties the garbled circuit GC – for every gate $g \in C$ with input wires u, v, and output wire w, and for every $\alpha, \beta \in \{0,1\}$, it outputs:

$$\tilde{g}_{\alpha,\beta} = A(g,\alpha,\beta) \cdot (k_{u,\alpha} + k_{v,\beta}) + E_{g,\alpha,\beta} + (\lceil\sqrt{p}\rceil \cdot (KEY_{w,\alpha,\beta} || e_{w,\alpha,\beta}))\,, \quad (11)$$

where

$$e_{w,\alpha,\beta} \stackrel{\text{def}}{=} g((\lambda_u \oplus \alpha), (\lambda_v \oplus \beta)) \oplus \lambda_w \quad (12)$$

$$KEY_{w,\alpha,\beta} \stackrel{\text{def}}{=} k_{w,0} + ((k_{w,1} - k_{w,0}) \cdot e_{w,\alpha,\beta})\,. \quad (13)$$

and $E_{g,\alpha,\beta}$ is the cumulative error, i.e.,

$$E_{g,\alpha,\beta} = \Sigma^n_{i=1} E^i_{g,\alpha,\beta}. \quad (14)$$

In addition, each party P_i receives the λ's corresponding to its input and output wires.

Fig. 5. Instantiated functionality $\mathcal{F}_{GC}$

If the mean is $\frac{\sqrt{p}}{2}$ and the standard deviation is sufficiently small, then with overwhelming probability the error will be in the range $[0, \sqrt{p}]$. Therefore, if the message is multiplied by $\sqrt{p}$ before encrypting, then dividing by $\sqrt{p}$ gets rid of the error and recovers the message.

Using this method, the probability that the protocol will output correctly is the probability that both the cumulative error and encrypted message are in the range $[0, \sqrt{p}]$ in all coordinates of all decrypted rows. If the parameters are correctly chosen, this happens with overwhelming probability. The online phase protocol for our ring-LWE based instantiation is given in Fig. 6.

5.2 Tailored Offline Phase Protocol for Ring-LWE Based Implementation

We now describe a "tailored" secure protocol for computing the offline phase for our ring-LWE based instantiation. The protocol is based on oblivious transfer and is secure up to $n-1$ corrupt parties. Other protocols for computing this functionality are also possible, e.g., using BGW if one assumes an honest majority.

The protocol we describe here is asymptotically better – the amount of work done by each party grows only quasilinearly in the number of parties. Furthermore,

Online Protocol for LWE Based Instantiation

Private Inputs: All parties hold their input to the functionality F_{GC} and the output they received from it.
In addidition, each party holds its input bit b_ω for each of its input wires ω.
Computation:

1. Each party broadcasts $e_\omega = \lambda_\omega \oplus b_\omega$ for the each of its input wires ω.
2. Each party P_i broadcasts k^i_{ω,e_ω} for each input wire of the circuit.
3. Each party does the following computation locally:
   (a) Computes $k_{\omega,e_\omega} = \Sigma_{i=1}^n k^i_{\omega,e_\omega}$ for every input wire ω of the circuit.
   (b) In topological order, for each gate $g \in C$, input wires u, v and output wire w computes

$$-A(g,\alpha,\beta) \cdot (k_{u,e_u} + k_{v,e_v}) + \tilde{g}_{\alpha,\beta} = E_{g,\alpha,\beta} + (\lceil\sqrt{p}\rceil \cdot (k_{w,e_w} || e_w)) \, .$$

   Dividing by $\sqrt{p}$ to get rid of the error, the party recovers the external value e_w and corresponding key k_{w,e_w} for the output wire w of the gate.
   (c) For each of its output wires ω computes the true value of the output $e_\omega \oplus \lambda_\omega$.

Output: Each party recovers the true value of the output, $e_\omega \oplus \lambda_\omega$, for each of its output wires ω.

Fig. 6. Online protocol for LWE instantiation

the PRFs are only computed *locally* and not in MPC, and therefore the circuit for computing the garbled circuit is independent of the complexity of the PRF.

The main idea of the protocol is that each party will encrypt its share, by adding the PRF, and broadcast. The sum of the received encryptions will be the garbled gate. This works due to the fact that both the shares and the (randomized) PRFs are additively homomorphic. Thus, it is reasonable to assume that a similar protocol exists for all key-homomorphic PRFs in which the operation on the keys is the same as the operation on the image of the PRFs.

To give more detail, the parties compute additive shares of the keys to encrypt using a sub-protocol. Then, each party encrypts its share and broadcasts. The parties sum the received broadcasts and recover the garbled gates. Intuitively, the security follows from Remark 1 and the following:

– For rows that are not decrypted during the online phase, at least one of the individual keys is unknown to the adversary. Thus, the encryption hides the share of the party.
– For rows that are decrypted during the online phase, the shares that are revealed (if the adversary is able to recover them, e.g., if the adversary controls $n-1$ parties) can be simulated by what the adversary can already learn from his shares and the decrypted key.

Offline Protocol for LWE Instantiation

Inputs: All parties hold the circuit C, the number of parties n, and the prime field $\mathbb{Z}_p$.

Computation: The parties perform the following computations, where all shares and computations are done in $\mathcal{R}_p$[a].

1. For every $\omega \in W$
   (a) Each party P_i selects its individual keys $k^i_{w,0}, k^i_{w,1} \leftarrow \mathcal{D}^N$, and sets the last coordinate of each key to 0. We denote the joint keys by $k_{w,0}, k_{w,1}$. Note that the last coordinate of the joint keys is also zero.
   (b) The parties run a coin-tossing protocol and receive additive shares of the hidden permutation bit $\lambda_w \in \{0, 1\}$ for every wire w.
2. For every gate $g \in C$ with input wires u, v and output wire w, and for every $\alpha, \beta \in \{0, 1\}$, the parties do the following:
   (a) Using a standard sub-protocol, compute additive shares of $k_{w,e_{w,\alpha,\beta}} || e_{w,\alpha,\beta}$. Denote the share of party P_i by $(k_{w,e_{w,\alpha,\beta}} || e_{w,\alpha,\beta})^i$.
   (b) Choose a Gaussian noise vector $E^i_{g,\alpha,\beta} \leftarrow \mathcal{D}^N$ and locally compute the encryption

$$A(g,\alpha,\beta) \cdot (k^i_{u,\alpha} + k^i_{v,\beta}) + E^i_{g,\alpha,\beta} + \left(\lceil \sqrt{p} \rceil \cdot (k_{w,e_{w,\alpha,\beta}} || e_{w,\alpha,\beta})^i\right). \quad (15)$$

   (c) Broadcast the encryption and sum, thus recovering

$$\tilde{g}_{\alpha,\beta} = A(g,\alpha,\beta) \cdot (k_{u,\alpha} + k_{v,\beta}) + E_{g,\alpha,\beta} + \left(\lceil \sqrt{p} \rceil \cdot (k_{w,e_{w,\alpha,\beta}} || e_{w,\alpha,\beta})\right). \quad (16)$$

3. Reconstruct λ_ω for every output wire ω of the circuit.

Outputs: $\tilde{g}_{\alpha,\beta}$ for each gate $g \in C$ and every $\alpha, \beta \in \{0, 1\}$ and λ_ω for every output wire ω of the circuit.

[a] As vector spaces, $\mathcal{R}_p \cong \mathbb{Z}_p^N$, but the multiplication is different

Fig. 7. LWE offline protocol

The offline protocol is described in detail in Fig. 7. The sub-protocol for computing additive shares of $KEY_{w,\alpha,\beta} || e_{w,\alpha,\beta}$ is straightforward using the observation in Sect. 4.4.2.

6 Implementation Details and Experimental Results

In this section we give details on our implementations and report our experimental results for online computation time with and without our various optimizations above.

We wrote our code for the online computation in C++ using the NTL library [35] for modular arithmetic and OpenSSL [33] for SHA. The code for our Ring-LWE based instantiation used the building blocks of Gaussian sampling, NTT transform, and arithmetic operations from [15]. Our code is publically available at https://github.com/cryptobiu/Protocols.

We tested our code on a circuit with 100,000 AND gates and 0 XOR gates (see Remark 3 below). Times are the average of 5 runs and reported in seconds. The experiments were run on Ubuntu 14.04.4 operating system using a single core of Intel(R) Core(TM) i7-5600U CPU @ 2.60 GHz processor. Clearly, times of all versions can be improved using parallelization, but we leave this to further research.

DDH Instantiation. The unoptimized version refers to the initial instantiation described in Sect. 4.2. The precomputation version refers to version without the random oracle that utilizes the precomputation of the exponent tables, described in Sect. 4.3. The DLSE_{500}, DLSE_{256} and DLSE_{160} refer to the optimizations relying on DLSE described in Sect. 4.3, with the individual keys having 500, 256, and 160 bits respectively. Since the security relies on the number of bits of the individual keys, and the size of the joint keys has $\log n$ more bits, we fixed the number of these bits to be 10, corresponding to up to 1023 parties. I.e., in DLSE_{500} the joint keys were 510 bits in DLSE_{256} 266 bits and in DLSE_{160} 170 bits[3]. All versions were used with a 1024 bit safe prime. Thus, $c \approx 500$ is the maximal $DLSE_c$ that can still use our sqrt. optimization. We provide a table to show the efficiency of the improvements, and they are also depicated in Fig. 1. The real security of $DLSE_c$ is unclear as far as we know. But there are known attacks that need only $\sim O(2^{\frac{c}{2}})$ exponentiations. Thus, if one aims for 80 bits of security, then one should use at least the DLSE_{160} version.

Version	Unoptimized	Precomp.	DLSE_{500}	DLSE_{256}	DLSE_{160}
Online Computation Time (sec.)	102.3	83.9	16.15	8.4	5.4

We also ran tests using larger safe primes with $1536, 2048$, and 3072 bits under the DLSE_{256} assumption. The results are given in the following table and in Fig. 8.

Number of bits in prime (DLSE_{256})	$\kappa = 1024$	$\kappa = 1536$	$\kappa = 2048$	$\kappa = 3072$
Online Computation Time (sec.)	8.4	14.9	21.65	43.2

LWE Instantiation. Our code for the ring-LWE instantiation used the following parameters, suggested in [16]: $p = 1051649, N = 512, \sigma = \frac{8}{\sqrt{2}}$. We chose these parameter to allow enough room for error, thus allowing a larger number of parties to participate. Note that for this choice of parameters, the total probability

[3] For the smaller number of parties the joint keys will have a few less bits, e.g., for <128 parties and DLSE_{160} the joint key should have only 167 instead of 170 bits. Thus, the time could possibly be very slightly better for the smaller number of parties.

the entire protocol errs on our chosen circuit is $<2^{-40}$ for up to 300 parties. For 500 parties, the total probability for error is $\approx 2^{-15.5}$, so changing the parameters should be considered for this number of parties.

The online computation time of our ring-LWE instantiation beat even the $DLSE_{160}$ version using a 1024 bit prime, with an average online computation time of approx. 4.45 s. For comparison, the result is depicated in Figs. 1 and 8.

Comparison with BMR. For comparison, we also measured the online computation time of a state of the art BMR implementation of [8] on the same circuit and hardware, with a varying number of parties. The code of [8] is highly optimized and uses AES-NI with pipelining for the encryption/decryption. The results are depicated in Figs. 1 and 8 for comparison.

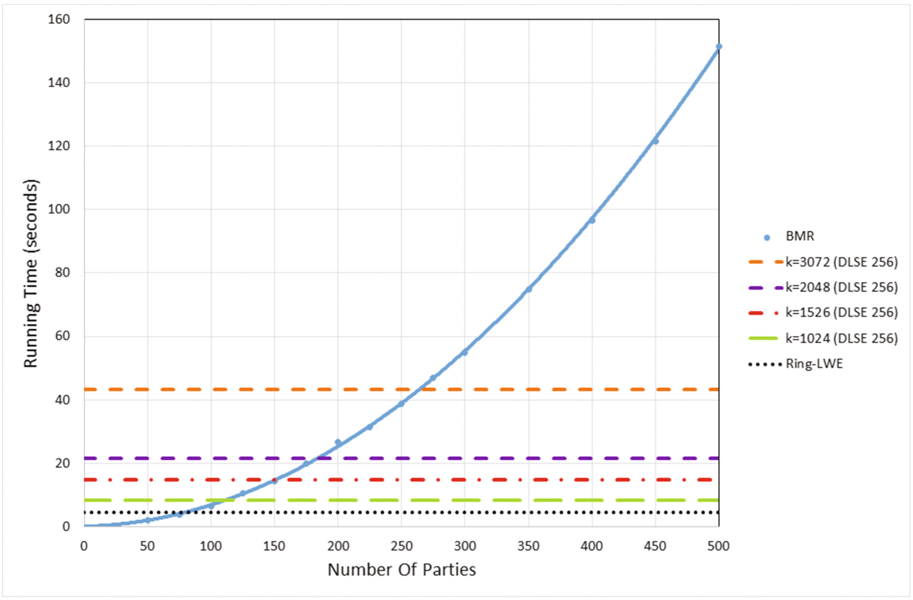

Fig. 8. Online Computation Time with DLSE_{256} and Varying Primes

As expected, for a large enough number of parties, our protocols' online computation time is faster than the respective time of the BMR protocol. With a 1024 bit prime: for the unoptimized version, the cutoff point is between 400 and 450 parties. For the precomputation version the cutoff is between 350 and 400 parties. For the DLSE_{500} and DLSE_{256} versions the cutoff points are between 150 and 175 and between 100 and 125 respectively. The cutoff points for the $DLSE_{160}$ and the ring-LWE version are near 75 parties. Cutoff points for larger primes are at a slightly higher number of parties.

We expect that on different hardware the times and cutoff points will differ, but the overall conclusion should remain the same. We also note that we did not use any dedicated hardware optimizations, while the BMR code of [8] uses pipelined fixed-key AES-NI.

Remark 3. The BMR protocol of [8] contains a free-XOR optimization, while our code computes XOR gates similarly to AND gates. Thus, for a true comparison on a specific circuit C with X XOR & XNOR gates and A non-XOR[4] gates, the time t of the BMR protocol should be adjusted to $t \cdot \frac{A}{A+X}$. However, for any fixed circuit, this will only change the exact location of the cutoff points, and not the overall conclusion.

Acknowledgements. We would like to thank Shalev Keren, Moria Farbstein and Lior Koskas for helping with the code, and to thank Shai Halevi and Vadim Lyubashevsky for helpful discussions on LWE.

References

1. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security, pp. 535–548. ACM (2013)
3. Banerjee, A., Peikert, C.: New and improved key-homomorphic pseudorandom functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 353–370. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_20
4. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: PODC (1989)
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, STOC 1990, pp. 503–513. ACM, New York (1990). https://doi.org/10.1145/100216.100287. ISBN 0-89791-361-2
6. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, 19–22 May 2013, pp. 478–492 (2013)
7. 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. ACM (2008)
8. Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: 23rd ACM Conference on Computer and Communications Security (ACM CCS) 2016 (2016). To appear
9. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for noncryptographic fault-tolerant distributed computations. In: Proceedings of the 20th ACM Symposium on the Theory of Computing, pp. 1–10 (1988)

[4] NOT gates can be eliminated even without free-XOR optimization, by modifying the circuit.

10. Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-secure encryption from decision Diffie-Hellman. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 108–125. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_7
11. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, 8–10 January 2012, pp. 309–325 (2012)
12. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29
13. Choi, S.G., Hwang, K.-W., Katz, J., Malkin, T., Rubenstein, D.: Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 416–432. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27954-6_26
14. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1
15. de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-LWE encryption. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 339–344, March 2015. https://doi.org/10.7873/DATE.2015.0378
16. Ghosh, S., Kate, A.: Post-quantum forward-secure onion routing. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 263–286. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28166-7_13
17. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. II. Cambridge University Press, Cambridge (2004)
18. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the 19th ACM Symposium on the Theory of Computing, pp. 218–229 (1987)
19. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. Cryptology ePrint Archive, Report 2017/214 (2017). http://eprint.iacr.org/2017/214
20. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proceedings of the 20th USENIX Security Symposium, San Francisco, CA, USA, 8–12 August 2011
21. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
22. Kiltz, E., Leander, G., Malone-Lee, J.: Secure computation of the mean and related statistics. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 283–302. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_16
23. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
24. Koshiba, T., Kurosawa, K.: Short exponent Diffie-Hellman problems. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 173–186. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24632-9_13

25. Larraia, E., Orsini, E., Smart, N.P.: Dishonest majority multi-party computation for binary circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 495–512. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_28
26. Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. J. Cryptology **22**(2), 161–188 (2009)
27. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant round multi-party computation combining BMR and SPDZ. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 319–338. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_16
28. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
29. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
30. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., et al.: Fairplay-secure two-party computation system. In: USENIX Security Symposium, San Diego, CA, USA, vol. 4. (2004)
31. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and KDCs. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 327–346. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_23
32. van Oorschot, P.C., Wiener, M.J.: On Diffie-Hellman key agreement with short exponents. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 332–343. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_29
33. OpenSSL Project. OpenSSL project (2006). http://www.openssl.org/
34. Schneider, T., Zohner, M.: GMW vs. Yao? efficient secure two-party computation with low depth circuits. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 275–292. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_23
35. Shoup, V.: NTL: A library for doing number theory (2003). http://www.shoup.net/ntl
36. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: 24th ACM Conference on Computer and Communications Security (ACM CCS) 2017 (2017, to appear)
37. Yao, A.C.: How to generate and exchange secrets. In: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, pp. 162–167 (1986)
38. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8