Homomorphic Secret Sharing from Paillier Encryption

Nelly Fazio[1*], Rosario Gennaro[1†], Tahereh Jafarikhah[2], and William E. Skeith III[1]

[1] The City College and Graduate Center of CUNY, New York, NY, USA
{fazio,rosario,wes}@cs.ccny.cuny.edu
[2] The Graduate Center of CUNY, New York, NY, USA jafarikhah@gmail.com

Abstract. A recent breakthrough by Boyle et al. [7] demonstrated secure function evaluation protocols for branching programs, where the communication complexity is sublinear in the size of the circuit (indeed just linear in the size of the inputs, and polynomial in the security parameter). Their result is based on the Decisional Diffie-Hellman assumption (DDH), using (variants of) the ElGamal cryptosystem. In this work, we extend their result to show a construction based on the circular security of the Paillier encryption scheme. We also offer a few optimizations to the scheme, including an alternative to the "Las Vegas"-style share conversion protocols of [7, 9] which *directly* checks the correctness of the computation. This allows us to reduce the number of required repetitions to achieve a desired overall error bound by a constant fraction for typical cases, and for large programs, reduces the total computation cost.

Keywords: Homomorphic secret sharing · Function secret sharing · Paillier encryption · Secure function evaluation · Private information retrieval.

1 Introduction

In this paper, following exciting recent results by Boyle et al. [7, 9], we present new protocols for low-communication MPC. We extend the results in [7] (proven secure under the Decisional Diffie-Hellman Assumption) by showing that they can be based on the circular security of the Paillier encryption scheme [37]. Additionally, we describe a verification technique to directly check correctness of the actual computation, rather than the absence of a potential error as in [7]. This results in fewer repetitions of the overall computation for a given error bound.

1.1 Background and Motivation

Secure MultiParty Computation (MPC) has been a vital research area in Cryptography for the last 30 years. Since the early seminal works [42, 43, 26, 4, 14],

*Author supported by NSF CAREER 1253927.
†Author supported by NSF Grant 1565403.

we know that it is possible for two or more parties to compute a joint function of individual secret inputs. It is a very powerful tool, since most, if not all of the security problems can be solved in principle using a multiparty computation protocol. Those initial results established the feasibility of the solutions, and at the same time highlighted their complexity. The research of the last 30 years has been focused on inventing increasingly powerful MPC techniques to get more efficient solutions. One of the bottleneck parameters that immediately attracted researchers' attention was communication complexity: all the early results require communication between the parties which is at least as large as the size of the circuit representing the function being computed. Ideally one would like the parties to exchange just a few messages of limited size.

Most of the research on this issue focused on types of "homomorphic" encryption (resp. secret sharing) schemes, which allow the computation of a function to be carried out non-interactively directly on the encryption (resp. shares) of the secret inputs. For example *additively homomorphic encryption* [27, 17, 34, 36, 37, 19] allows the parties to publish encryptions of their inputs, and to compute any linear function without interaction (except for a final decryption step). Similarly this can be achieved by using *linear secret sharing* such as Shamir's [39]. These techniques were applied to the concept of *Private Information Retrieval (PIR)* [16, 15, 32] which allows the secure computation of a "selection" function (Party 1 holds n values $x_1, \ldots, x_n$, Party 2 holds an index i, the output is x_i) with communication which is sublinear in the size of the circuit.

General solutions for any function had to wait for the discovery by Gentry of *Fully Homomorphic Encryption (FHE)* [21] which enables the computation of arbitrary functions over encrypted input, breaking the circuit barrier in general. The drawback of FHE is that in spite of continuous progress [40, 23, 10], even the best implementations of FHE remain quite slow [22, 24, 35]. Additionally, the set of cryptographic assumptions underlying FHE remains limited to assumptions related to the complexity of lattice based problems [20, 25, 11, 12], and do not include more classical assumptions such as factoring or discrete logarithm.

These observations motivated Boyle et al. to look for alternatives. In a very exciting recent result [7] they present a Homomorphic Secret Sharing scheme which allows the non-interactive computation of *Branching Programs* over the shares of the secret inputs. Further optimizations (as well as transporting some results to the generic group model for DDH-hard groups) are given by the same authors in [9]. Their scheme is orders of magnitude more efficient than FHE and its security is based on the DDH Assumption.

1.2 Our Results

We extend the results in [7, 9] by showing that Homomorphic Secret Sharing for Branching Programs can be based on the (circular) security of the Paillier [37] encryption schemes.

While our protocols follow the same blueprint of the Homomorphic Secret Sharing in [7] our extensions were not immediate. Below we give an overview of the main technical problems and challenges we encountered, and the techniques used to overcome them.

1.3 Techniques

To begin, we give a very high-level review of the techniques used in [7]. We then outline where new techniques are needed for our work. Informally, the construction of [7] follows these steps:

1. The scheme uses the ElGamal encryption scheme modified to be additively homomorphic by placing the plaintext in the exponent. That is, encryptions of a message x look like $[\![x]\!] = (\alpha = g^r, \beta = h^r \cdot g^x)$, where $h = g^c$ is the public key. When messages are small, decryption is feasible by performing a discrete logarithm after the usual ElGamal decryption.
2. The scheme also uses, simple 2-out-of-2 additive sharing. Given $z \in Z_q$ where q is the order of the ElGamal group, we denote $\langle z \rangle = (z_1, z_2)$ such that $z_1 + z_2 = z \bmod q$, where each party P_i holds z_i.
3. Given $[\![x]\!] = (\alpha, \beta)$, $\langle y \rangle = (y_1, y_2)$ and $\langle cy \rangle = (w_1, w_2)$ each party P_i can now locally compute a share γ_i as $\gamma_i = \beta^{y_i} \cdot \alpha^{-w_i}$, such that $\gamma_1 \cdot \gamma_2 = g^{xy}$, i.e. a *multiplicative sharing* of g^{xy}. Note how this step effectively removes the randomness from the encryption of x, using the secret key c.
4. Finally, a clever technique is used to compute a *distributed discrete logarithm*, thus recovering an *additive* sharing of xy without the need for interaction. We point out that their procedure requires the multiplicative sharing to be in a *cyclic* group.

Abstracting out from the specifics, we can see that the scheme in [7] requires the following ingredients:

- An encryption scheme which is both *message* and *key homomorphic* over $\mathbb{Z}$ (or a finite quotient), i.e., a scheme that allows the transformation in step 3 above.
- A non-interactive method for transforming a multiplicative sharing of g^z into an additive sharing of z, where these two values "live" in the ciphertext and message space (respectively) of the encryption scheme.

Our Construction. We now address the challenge of adapting these techniques to make use of the Paillier cryptosystem [37]. Recall that Paillier is naturally additively homomorphic over the integers, which works in our favor here. Additionally, we can use a version of Paillier threshold decryption [18, 30] to obtain the "key homomorphic" property which allows to perform Step 3.

Recall that a Paillier encryption of an integer x is of the form $g^x r^n \bmod n^2$, where n is an RSA modulus, $\mathrm{ord}(g) = n$, and $\mathrm{ord}(r^n) \mid \varphi(n)$. Since it is required that $(n, \varphi(n)) = 1$, we can use the Chinese Remainder Theorem to find an integer λ such that

$$\lambda \equiv \begin{cases} 1 & \bmod n \\ 0 & \bmod \varphi(n) \end{cases} .$$

Now if $\sigma = g^x r^n$ is an encryption of x, then by raising to the λ power we get: $\sigma^\lambda = (g^x r^n)^\lambda = g^x \bmod n^2$. While there are efficient procedures for completing the decryption (recovering x from g^x), note that we have already made substantial progress in obtaining the necessary ingredients for the [7] blueprint. Given an additive sharing $\langle \lambda y \rangle = (z_1, z_2)$ of λy (so that $z_1 + z_2 = \lambda y$), then $(\sigma^{z_1}, \sigma^{z_2})$ is a multiplicative sharing of g^{xy}, i.e.

$$\sigma^{z_1}\sigma^{z_2} = \sigma^{\lambda y} = g^{xy} \bmod n^2.$$

If xy is relatively small, we might hope to then perform the distributed discrete log protocol from [7], however there are a few complications. To begin, it is not entirely obvious that the distributed discrete log protocol would work in $\mathbb{Z}_{n^2}^\times$ which is not a cyclic group. For example, while certainly g^{xy} lives in $\langle g \rangle$, each party's shares do not – the shares sit in $\mathbb{Z}_{n^2}^\times$, and furthermore in different cosets of $\langle g \rangle$. Fortunately, we can modify the protocol in [7] (as well as most of the variants from [9], sans a few optimizations) to work for any finite group in a fairly straightforward way (see Section 3 for details).

The other main issue concerns the representation of our additive shares. In the original ElGamal-based scheme, additive shares of a value y satisfy $\sum y_i \equiv y \bmod q$, where q is the order of the group. Note that q is *public* in this case. Thus, each party can perform addition modulo q without knowledge of any secret values. In Paillier, however, we need to work with additive shares of values that work modulo $n\varphi(n)$, a value that must be kept secret. Therefore we do this sharing over the integers. Without a careful implementation this step can cause the size of the shares to grow exponentially, but we are able to avoid this problem. Details can be found in Section 4.

Verifying Computations. In [7, 9], the authors describe "Las Vegas" style techniques to check for the *potential* risk of having incurred an incorrect computation during each step of the protocol. If the possibility of an error is never signaled, then the overall computation is considered correct. This method was then shown to provide efficiency improvements for several applications.

In this work, we describe a technique to *directly* check correctness which verifies the actual computation, rather than the absence of a potentially "risky" situation. This method of checking does not produce false negatives (erroneously reporting that the protocol failed), and allows us to reduce even further the number of required invocations for a desired overall error bound by a constant fraction. The price we pay for this (in addition to a negligible probability of a false positive), is some extra effort to compute the values used in the check. However, this effort depends linearly on the program size, whilst each repetition takes quadratic time in the program size. Hence, we achieve a savings in computation for large programs. Our verification method works both for the original ElGamal-based construction of [7] and for our Paillier-based construction, although the benefits are more pronounced for the latter. Details can be found in Section 5.

2 Preliminaries

2.1 Encryption

A public-key encryption system Π consists of three algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen is a key generation (randomized) algorithm that takes a security parameter k and outputs a public-secret key pair $(\mathsf{PK}, \mathsf{SK})$; $\mathsf{Enc}(\mathsf{PK}, m)$ is the encryption (randomized) algorithm that on input a message m and the public key PK outputs a ciphertext c; and $\mathsf{Dec}(\mathsf{SK}, c)$ decrypts ciphertext c with secret key SK. Obviously if $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^k)$ and $c \leftarrow \mathsf{Enc}(\mathsf{PK}, m)$ then $m = \mathsf{Dec}(\mathsf{SK}, c)$.

Semantic Security [27] says that no polynomial time adversary can distinguish between the encryption of two messages of its choice. For all PPT $\mathcal{A}$

$$\begin{aligned} Pr[b' = b :& (\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(k), (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{PK}), \\ & b \leftarrow \{0,1\}, b' \leftarrow \mathcal{A}^{\mathcal{O}_b}(\mathsf{PK})] \leq \frac{1}{2} + \nu(k) \end{aligned}$$

where oracle $\mathcal{O}_b$ takes no input and outputs $c \leftarrow \mathsf{Enc}(\mathsf{PK}, m_b)$, and $\nu(k)$ is a negligible function.

Circular Security A public-key encryption Π is circular secure if it remains secure even encrypting messages that depend on the secret keys in use. More precisely, if c with $length(c) = l(k)$ is the secret key of the public key encryption scheme Π which encrypts bits, there is a negligible function $\nu(k)$ that the following holds for all PPT $\mathcal{A}$:

$$Pr[b' = b : (\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(k), b \leftarrow \{0,1\}, b' \leftarrow \mathcal{A}^{\mathcal{O}_b}(\mathsf{PK})] \leq \frac{1}{2} + \nu(k)$$

where oracle $\mathcal{O}$ takes no input and outputs $(D_1, D_2, \dots, D_l)$ such that

$$\begin{cases} \forall i \in [l], D_i \leftarrow \mathsf{Enc}(\mathsf{PK}, 0) & if\ b = 0 \\ \forall i \in [l], D_i \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathsf{SK}^i) & if\ b = 1 \end{cases}$$

in which SK^i is the *i-th* bit of SK. Later we will see that circular security plays an important role in the construction of our homomorphic secret sharing. We remark that circular security implies semantic security.

2.2 The Paillier Encryption Scheme

Let n be an RSA modulus, i.e. $n = pq$ where p, q are primes. A number z is said to be an *n-th residue modulo* n^2 if there exists a number $y \in \mathbb{Z}_{n^2}^{\times}$ such that $z = y^n \mod n^2$. We assume that there exists no polynomial time distinguisher for n-th residues $\mod n^2$. We will refer to this hypothesis as the *Decisional Composite Residuosity Assumption* (DCRA).

More formally, we assume that there exists a randomized RSA key generation algorithm RSAGen that on input a security parameter 1^k selects two k-bit primes. Then we say that the DCRA holds (with respect to RSAGen) if for all PPT $\mathcal{A}$ there exists a negligible function $\nu(k)$, such that

$$Pr[b' = b : (p, q) \leftarrow \mathsf{RSAGen}(k), n = pq, b \leftarrow \{0, 1\}, b' \leftarrow \mathcal{A}^{\mathcal{O}_b}(n)] \leq \frac{1}{2} + \nu(k)$$

where oracle $\mathcal{O}_b$ takes no input, selects y uniformly at random in $\mathbb{Z}^{\times}_{n^2}$ and outputs z such that $z = y$ if $b = 0$, and $z = y^n$ if $b = 1$.

The Paillier encryption scheme, whose security is based on DCRA is defined as follows (where we use the modified definition of the secret key λ from [19, 30] used in their threshold variant of the scheme). The key generation algorithm $\mathsf{KeyGen}_{Paillier}(1^k)$ picks two k-bit prime numbers p and q such that $n = pq$ satisfies $(n, \varphi(n)) = 1$ (which will hold with high probability for such n), computes

$$\lambda = \begin{cases} 1 & \mod n \\ 0 & \mod \varphi(n) \end{cases} \tag{1}$$

and outputs $(\mathsf{PK}, \mathsf{SK})$ for $\mathsf{PK} = n$ and $\mathsf{SK} = \lambda$. Note that the existence of such a λ, as well as an efficient means of computing it, are given by the Chinese Remainder Theorem since $(n, \varphi(n)) = 1$. Note also that λ is unique in the range $[0, \ldots, n\varphi(n) - 1]$. The encryption algorithm for a message $x \in \mathbb{Z}_n$ is defined by

$$\mathsf{Enc}_{Paillier}(\mathsf{PK}, x) = (1 + n)^x \cdot r^n \mod n^2$$

and the decryption algorithm for $\sigma < n^2$ is defined by

$$\mathsf{Dec}_{Paillier}(\mathsf{SK}, \sigma) = \frac{L(\sigma^\lambda \mod n^2)}{L((1 + n)^\lambda \mod n^2)} \mod n \;\text{ where } L(u) = \frac{u - 1}{n}$$

Paillier is an additive homomorphic scheme; given only the public-key and $\sigma_i = \mathsf{Enc}_{Paillier}(x_i)$ then $\sigma_1 \cdot \sigma_2 \mod n^2 = \mathsf{Enc}_{Paillier}(x_1 + x_2 \mod n)$.

2.3 Homomorphic Secret Sharing

A 2-out-of-2 homomorphic secret sharing scheme (HSS) [7] deals with the scenario that a client wants to split a secret input $w \in \{0, 1\}^n$ into shares (w_0, w_1), and sends each w_i to a different server. Each server holding a representation of a function f, can locally compute additive shares of $f(w)$.
A representation for a function is a program P (a collection of bit strings). For an input $w \in \{0, 1\}^n$, the output of P is represented by $P(w)$. The symbol $\perp$ is used when the output of $P(w)$ is undefined. For simplicity we can consider the inputs and outputs of a function as binary strings. A HSS scheme consists of two algorithms: Share that splits the secret into two shares and Eval that evaluates a program P on two inputs such that the outputs are the additive shares of $P(w)$.

Definition 1. *A homomorphic secret sharing scheme with error bound δ for the collection of programs $\mathcal{P}$ consists of algorithms* $(\mathsf{Share}, \mathsf{Eval})$ *with the following properties:*

- $\mathsf{Share}(1^k, w)$*: on the security parameter 1^k and $w \in \{0,1\}^n$ outputs (w_0, w_1).*
- $\mathsf{Eval}(b \in \{0,1\}, w_b, P, \delta)$ *outputs y_b.*
- **Correctness***: For every polynomial p there exists a negligible function ν such that for every k, w, P, δ in which $|P|, 1/\delta \leq p(k)$*

$$\begin{aligned} Pr[y_0 + y_1 = P(w) :& (w_0, w_1) \leftarrow Share(1^k, w), \\ & y_b \leftarrow Eval(b, w_b, P, \delta), b \in \{0,1\}] \geq 1 - \delta - \nu(k) \end{aligned}$$

- **Security***: Each share computationally hides the secret input.*

We would like to apply a stronger version of HSS that allows homomorphic computation on encrypted inputs.

Definition 2. *A Distributed-Evaluation Homomorphic Encryption (DEHE) with error bound δ for a class of programs $\mathcal{P}$ consists of three algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$ *as follows:*

- $(\mathsf{PK}, (e_0, e_1)) \leftarrow \mathsf{KeyGen}(1^k)$*: It takes a security parameter 1^k and outputs a* PK *and a pair of evaluation keys (e_0, e_1).*
- $E_w := \mathsf{Enc}(\mathsf{PK}, w)$*: It encrypts a secret input bit w and output c.*
- $\mathsf{Eval}_b := \mathsf{Eval}(b \in \{0,1\}, e_b, \mathbf{c} = (c_1, c_2, \ldots, c_n), P, \delta)$*: Outputs y_b as party b's share of output y.*
- **Correctness***: For every polynomial p there exists a negligible function ν such that for every $k, w = (w^1, \ldots, w^n) \in \{0,1\}^n, P, \delta$ in which $|P|, 1/\delta \leq p(k)$*

$$\begin{aligned} Pr[y_0 + y_1 = P(w) :& (\mathsf{PK}, (e_0, e_1)) \leftarrow \mathsf{KeyGen}(1^k), \\ & C \leftarrow (E_{w^1}, \ldots, E_{w^n}), y_b \leftarrow \mathsf{Eval}_b] \geq 1 - \delta - \nu(k) \end{aligned}$$

- **Security***: Let D_b stand for the distribution obtained by applying the evaluation key e_b in this setting. The security of the DEHE scheme means that D_0 and D_1 are computationally indistinguishable.*

2.4 Restricted Multiplication Straight-line Programs (RMS)

Our construction will provide non-interactive evaluation of some specific collection of programs called as restricted multiplication straight-line programs (RMS). The class of RMS programs with bound 1^M (where M is an upper bound for the size of a memory location) is an arbitrary sequence of the instructions as follow:

1. Load an input $x = (x^1, x^2, \ldots, x^n) \in \{0,1\}^n$ into memory: $y_j \leftarrow x^i$.
2. Add memory locations: $y_k \leftarrow y_i + y_j$.
3. Multiply a memory location by an input: $y_k \leftarrow x^i \cdot y_j$.

4. Output a memory location: $O_j \leftarrow y_j$.

Whenever the size of a memory value exceeds M, the program aborts and outputs $\perp$. We define the size of an RMS program as the number of its instructions. As pointed out in [7] RMS programs can be used to evaluate branching programs with constant overhead.

3 Share Conversion

Here we provide our first technical contribution: A generalization of the distributed discrete log and share conversion procedures from [7] which works in any finite group G, not just a cyclic group.

Consider the setting of two party computation, where one party holds x and the other party holds y such that $xy = g^b$ where g is an element of a group G (i.e. (x, y) is a multiplicative sharing relative to g of a small value b. Suppose that both parties have access to a random function $\phi : G \longrightarrow \{0, \ldots, k-1\}$ for $k \in \mathbb{N}$ (appropriate values for k will be determined shortly).

We will prove that if each party locally runs the procedure DDLog below (where the input a is set to the share held by each party and δ, M are parameters we will determine later) then at the end, the parties output values i, j such that $i - j = b$ with sufficiently high probability. In other words the procedure simultaneously computes the discrete log[3] of g^b and turns the multiplicative sharing into an additive one.

Algorithm 1 $\mathsf{DDLog}_{G,g}(a, \delta, M, \phi)$

```
1: $i = 0; h = a; T = 2M\ln(2/\delta)/\delta$
2: while $\phi(h) \neq 0$ and $i < T$ do
3:     $h = gh$
4:     $i = i + 1$
5: end while
6: return $i$
```

Let G be a finite group, and $g \in G$. Note that if two elements $x, y \in G$ have a product in $\langle g \rangle$, this is of course equivalent to saying that x and *the inverse* of y live in the same coset of $\langle g \rangle$, or put another way, x and y^{-1} differ by some number of "g-steps":

$$x = g^b y^{-1}. \tag{2}$$

Define $S = \phi^{-1}(\{0\})$. Then the parties will be able to "synchronize" by counting g-steps to the next value in S, recovering an additive sharing of b. The parameter k can be used to balance the running time of the process with its success probability. The basic idea is depicted in Figure 1. Note that the domain

[3]There is no contradiction here with the hardness of discrete log, since this works only for small values of b.

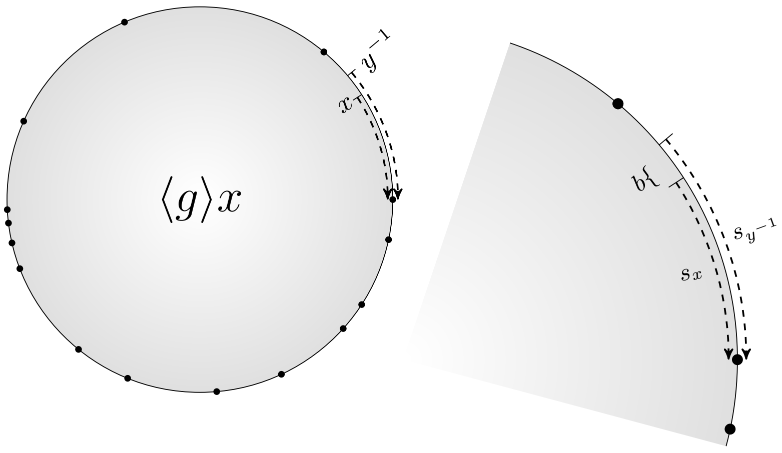

Fig. 1. Illustration of DDLog procedure on $xy = g^b$. Here, x, y are *multiplicative shares* of a small value b, which are inputs to DDLog. Both x and y^{-1} sit in the same coset $\langle g\rangle x$ of $\langle g\rangle$. The dots represent the elements of a random δ-sparse subset S in $\langle g\rangle x$. Note that (with good probability) the difference in the number of steps taken is $b = s_{y^{-1}} - s_x$, so that $(-s_x, s_{y^{-1}})$ is an additive sharing of b.

of ϕ must be the *entire group* G not just the particular coset where x, y^{-1} reside. Indeed, finding a useful representation of that coset (in order to instantiate ϕ) might be difficult.[4]

Fortunately this is not much of a complication – equation (2) combined with the fact that ϕ gives random labels to each element allows the same analysis to proceed for the restriction of ϕ to any coset. Indeed, Algorithm 1 is a proper generalization of the corresponding algorithm from [7] (where the group is cyclic $\langle g\rangle = G$ so there is only one coset), and a very similar argument suffices to show its correctness in our application. We provide a few details for completeness. Following the notation of [7], we set M to be an upper bound on the value being shared and T will be a "timeout" value.

Proposition 1 ([7, Prop. 3.2]). *Let G be any finite group, $g \in G$, $\delta > 0$, and $M \in \mathbb{N}$. If $M, T < \mathrm{ord}(g)$, then for any $x, y \in G$ such that $xy = g^b$ with $b < M$, we have*

$$\Pr_{\phi}\left[\mathsf{DDLog}(y^{-1}, \delta, M, \phi) - \mathsf{DDLog}(x, \delta, M, \phi) = b\right] \geq 1 - \delta$$

where ϕ is sampled uniformly from all functions from $G \longrightarrow \{0, \ldots, \lfloor 2M/\delta \rfloor\}$.

[4]For example in our DCRA-based construction, this would be equivalent to decryption.

Proof sketch: Modeling ϕ as a random function from $G \longrightarrow \{0, \dots, \lfloor 2M/\delta \rfloor\}$, note that for any $a \in G$ we have $\Pr_{\phi \xleftarrow{\$} \mathcal{R}} [\phi(a) = 0] \approx \frac{\delta}{2M}$, and in particular the same is true of ϕ restricted to the coset $\langle g \rangle x$. With this in hand, the rest of the proof proceeds as that of [7, Prop. 3.2], using a few straightforward applications of the well known inequality $1 + x \leq e^x$. $\square$

Lastly, we remark that when the random function ϕ is replaced by a pseudorandom function (PRF), an analogous proposition holds, stating that no efficient adversary can find a sequence of instructions that would cause the probability to deviate substantially below $1 - \delta$. The important observation (present in [7]) is that by modularizing DDLog (in particular, this procedure accesses ϕ as an oracle, and does not need to know the seed), we can use any adversary that finds an input which is "bad" for a PRF ϕ to construct an adversary that distinguishes ϕ from random, thus breaking the security guarantee of the PRF.

4 Construction from DCRA

Using DDLog introduced in previous section as one of our sub-procedures, we will present an *HSS* scheme based on the circular security of Paillier's encryption which evaluates RMS programs (see section 2.4). We make use of the following convenient notation, borrowed from [7]:

1. For input $x \in \mathbb{Z}_n$, $[\![x]\!]_\lambda$ is a Paillier encryption of x with respect to the secret key λ. That is, $[\![x]\!] := E(x) = \sigma = (1+n)^x \cdot r^n \bmod n^2$ where $r \xleftarrow{\$} \mathbb{Z}_n^\times$, and λ is the unique integer in $[0, \dots, n\varphi(n) - 1]$ satisfying equation (1). Note that $\sigma^\lambda = (1+n)^x \in \mathbb{Z}_{n^2}^\times$ in this case.
2. $\langle y \rangle$ refers to *additive secret shares* of y, i.e., two values y_0, y_1 such that $y = y_0 + y_1$ *over the integers.*
3. Lastly, $\langle\!\langle y \rangle\!\rangle$ refers to *multiplicative secret shares* of $(1+n)^y$ i.e., two values $h_0, h_1 \in \mathbb{Z}_{n^2}$ such that $h_0 \cdot h_1 = (1+n)^y \bmod n^2$. These are intermediate values that arise during multiplication instructions, and will be converted back to $\langle y \rangle$ by the sub-routine DDLog.

Note that $[\![x]\!]_\lambda$ is a *global value* meaning that both parties receive the same value, in contrast to $\langle y \rangle$ and $\langle\!\langle y \rangle\!\rangle$, where each party has a different share. In the following we denote with $\lambda^{(i)}$ is the i-th bit of the binary representation of λ; that is, $\lambda = \sum_{i=0}^{\ell-1} 2^i \lambda^{(i)}$.

When evaluating an RMS program a dealer will share each input $x \in \mathbb{Z}_n$, in the following way $[\![x]\!]_\lambda, \{([\![x\lambda^{(i)}]\!]_\lambda\}_{i=0}^{\ell-1}, \langle x \rangle, \langle \lambda x \rangle$. Note that this will typically include encryptions of many bits of λ which is why we need the circular security assumption for Paillier.

Values y in memory locations will instead be stored as $\langle y \rangle, \langle \lambda y \rangle$. The original shares of all additive sharing are chosen randomly in $[-n^3, n^3]$ which result in a distribution that is statistically close to uniform for any shared value.

We first notice that additions are easily computed due to the homomorphic properties of Paillier's encryption and the additive secret sharing. One thing to note is that the size of the additive sharing increases by at most one bit after each addition since each player locally adds shares over the integers. This is not a major problem (since the size of the shares will still be polynomial by the end of the execution of the program). Furthermore, upon each multiplication step we will again have *small* additive shares for the product, as these shares are produced by DDLog (which outputs shares of *logarithmic* size in its polynomial running time). We discuss this further in what follows.

We now turn our attention to the computation of multiplication between an input x and a memory location value y. Since this value will be stored in a memory location (and so that it may be used again in subsequent multiplications) we need to compute $\langle xy \rangle$ and $\langle \lambda xy \rangle$.

The computation of $\langle xy \rangle$ uses $[\![x]\!]_\lambda$ and $\langle \lambda y \rangle$ via the following steps[5]

$$([\![x]\!]_\lambda, \langle \lambda y \rangle) \xrightarrow{(a)} \langle\!\langle xy \rangle\!\rangle \xrightarrow{(b)} \langle xy \rangle. \tag{3}$$

A description of steps (a) and (b) follows:

(a) Let $z_1 + z_2 = \lambda y$ and $\sigma = [\![x]\!]_\lambda$. Then each player computes $\gamma_i = \sigma^{z_i} \bmod n^2$. Note that $\gamma_1 \cdot \gamma_2 = \sigma^{\lambda y} = (1+n)^{xy} \bmod n^2$. In other words $(\gamma_1, \gamma_2) = \langle\!\langle xy \rangle\!\rangle$. We denote with $(\gamma_1, \gamma_2) = \mathsf{MultShares}([\![x]\!]_\lambda, \langle \lambda y \rangle)$.

(b) Use the DDLog procedure on (γ_1, γ_2) with parameters δ, M (which will be specified by the RMS program being run on the shares) and random function ϕ. We denote with $\mathsf{ConvertShares}(\langle\!\langle xy \rangle\!\rangle, \delta, M, \phi)$ the pair

$$\langle xy \rangle = (-\mathsf{DDLog}(\gamma_1, \delta, M, \phi), \mathsf{DDLog}(\gamma_2^{-1}, \delta, M, \phi)).$$

Note that the first party negates the result of DDLog to maintain the invariant that the shares *add* to the shared value (DDLog output shares whose difference is the shared value), and that the second party must invert her share before invoking DDLog (see Figure 1).

Then, to compute $\langle \lambda xy \rangle$ we use $\left\{ ([\![x\lambda^{(i)}]\!]_\lambda \right\}_{i=0}^{\ell-1}$ and $\langle \lambda y \rangle$ as follows

$$\left\{ \left(\left[\!\left[x\lambda^{(i)} \right]\!\right]_\lambda, \langle \lambda y \rangle \right) \right\}_{i=0}^{\ell-1} \xrightarrow{(c)} \left\{ \langle \lambda^{(i)} xy \rangle \right\}_{i=0}^{\ell-1} \xrightarrow{(d)} \langle \lambda xy \rangle. \tag{4}$$

A description of steps (c) and (d) follows:

(c) ℓ invocations of step (a,b) above to compute each $\langle xy\lambda^{(i)} \rangle$

(d) Each party will locally multiply the i-th share by the value 2^i and sum these shares together.

Note that if the shares in $\langle \lambda y \rangle$ are of size t at the beginning of this step, at the end they are of size at most $3t$ ($2t + \ell$ to be precise[6]). However these shares do

[5]Differently than in [7] we do not use $\langle y \rangle$ in the multiplication step – The additive sharing of y however needs to be stored so that we can compute the output at the end.

[6]$\ell \leq t$ since additive shares start of size ℓ and then they can grow as the result of addition operations.

not grow further since at the next step they are used "in the exponent", and the result of additive shares coming out of the DDLog procedure is always ℓ.

The following figures will present our homomorphic secret sharing scheme (Share, Eval).

Homomorohic Secret Sharing Scheme-Share$(1^k, x_1, \ldots, x_n)$

The inputs are security parameter 1^k and bits x_i for $i \in \{0, \ldots, n\}$.

- Sample k-bit prime numbers p, q, set $n = pq$ and $\varphi(n) = (p-1)(q-1)$.
- Compute $\lambda \in \mathbb{Z}_{n \cdot \varphi(n)}$ according to Equation 1. Let $\ell = \log(n^2)$.
- Sample a PRF $\phi : \mathbb{Z}_{n^2}^{\times} \longrightarrow \{0, \ldots, \lfloor 2M/\delta \rfloor\}$.
- For each input x_i sample the following;
    - $[\![x_i]\!]_\lambda$: A Paillier encryption of the integer x_i with public key n^2.
    - $\left[\!\left[\lambda^{(t)} x_i\right]\!\right]_\lambda$, $\forall t \in [l] := \{0, ..., l-1\}$: A Paillier encryption of the integer $\lambda^{(t)} x_i$ with public key n^2.
    - $\langle x_i \rangle \leftarrow \mathsf{AdditiveShare}(x_i)$.
    - $\langle \lambda x_i \rangle \leftarrow \mathsf{AdditiveShare}(\lambda x_i)$.
- Party b receives

$$\mathsf{Share}_b = \{\phi, ([\![x_i]\!]_\lambda, \{\left[\!\left[\lambda^{(t)} x_i\right]\!\right]_\lambda\}_{t \in [l]}, \langle x_i \rangle_b, \langle \lambda x_i \rangle_b)_{i \in [n]}\}.$$

Notation: The AdditiveShare operator on input x selects $\alpha \in [-n^3, n^3]$ uniformly at random and computes $\beta = x - \alpha$ over the integers. It sets $\langle x \rangle = (\alpha, \beta)$.

Fig. 2. Share for secret sharing an input x via the HSS scheme

Theorem 1. *Assuming that Paillier is circular secure, the scheme* (Share, Eval) *as described in figures* 2*and* 3 *is a secure homomorphic secret sharing with error* δ *for the class of RMS programs.*

The proof follows the same structure of the proof in [7] and we refer the reader to that proof. The only difference is that our additive sharings are *statistically* secure rather than *perfectly* secure as in [7]. This comes into play only in the proof of Lemma 3.11 in [7], specifically in the proof of the indistinguishability of Hybrid 0 versus Hybrid 1. In our simulation the shares of each player in Hybrid 1 are chosen uniformly at random in $[-n^3, n^3]$. For player P_1 this distribution is identical to the distribution in the real protocol (Hybrid 0). For player P_2 that's not the case, indeed the distribution of the shares of this player in the real protocol is uniform in $[-n^3 + x, n^3 + x]$ where x is the value being shared. It's not hard to see that the statistical distance between the two distributions is $\frac{2x}{2n^3}$ which is $O(\frac{1}{n})$ i.e. negligible in the worst case when $x = \lambda = O(n^2)$.

From Private to Public-key. In the construction above, secret shares of an input x consisted of Paillier encryptions $[\![x]\!]_\lambda$, $\{[\![\lambda^{(i)} x]\!]_\lambda\}_{t \in [l]}$ and additive secret shares

$\langle x \rangle, \langle \lambda x \rangle$. It is not immediately clear how one would generate those values without knowing the secret λ. However, by leveraging the homomorphic property of Paillier, we can generate these values for a secret sharing of x given only public key information which is independent of the input x. We can set up an initiative algorithm that samples a Paillier key pair (n, λ), encryptions of $\{ \llbracket \lambda^i \rrbracket_\lambda \}_{t \in [l]}$, and evaluation key corresponding to additive secret shares of $\langle \lambda \rangle$. A user without any knowledge of the secret key can then compute $\llbracket x \rrbracket_\lambda$ and $\{ \llbracket \lambda^i x \rrbracket_\lambda \}_{t \in [l]}$ using the public parameters and homomorphic property of the underlying encryption scheme. Values $\langle x \rangle$ and $\langle \lambda x \rangle$ can be computed by running Eval.

Optimizing the generator. For protocols based on DDH, considerable practical performance improvements have been demonstrated in [9]. For example, by using the quadratic reciprocity theorem to choose pseudo-Mersenne primes p for which large prime order subgroups of $\mathbb{Z}_p^\times$ are generated by the integer 2, impressive speed-ups for DDLog are shown. Unfortunately, these techniques do not seem to transfer well to Paillier, as the analogous subgroups (for which 2 is a generator) would naturally be contained in the subgroup of n-th powers, rather than $\langle 1+n \rangle$. While it might be the case that rejection sampling safe primes until $\langle 2 \rangle = \langle 1+n \rangle$ is plausible,[7] and moreover such that the modulus n is close to a power of 2, it is not clear how this would affect security. However, we note that the "standard" generator $(1+n)$ of the subgroup of order n actually admits a small optimization, which is as follows. Let h denote the share of one of the parties, which will be input into DDLog. First, write $h = an+b$, where $a, b < n$. Then notice that $h(1+n) \equiv (a+b)n + b \bmod n^2$. Also, note that since the two inputs to corresponding DDLog invocations will be in the same coset of $\langle 1+n \rangle$, the values b will also be identical for each share. So not only can we define the PRF ϕ to have domain $\mathbb{Z}_n$, more importantly we can substitute a multiplication (by $(1+n) \bmod n^2$) with a simple *addition* of two values in $\mathbb{Z}_n$ (we only need to keep track of $(a+b) \bmod n$ for each step). Since performing the group multiplications was the most costly part of DDLog, this may yield considerable savings in computation.

[7]At least the test is efficient if the factorization of the order of the group is known, as is the case if n was a product of safe primes.

Homomorphic Share Evaluation of RMS Programs-$\mathsf{Eval}_{G,g}(b, \mathsf{Share}_b, P, \delta)$

Each party P_b runs on its secret share value Share_b, the RMS program P of size $\leq S$ with magnitude bound 1^M, error bound δ. Set $\delta' := \delta/((l+1)MS)$.

```
– Load inputs into memory:
    • $\langle y_j \rangle \leftarrow \langle x_i \rangle$.
    • $\langle \lambda y_j \rangle \leftarrow \langle \lambda x_i \rangle$.
  In which $\langle x_i \rangle$ and $\langle \lambda x_i \rangle$ are as in Share.
– Addition over memory values:
    • Compute $\langle y_k \rangle \leftarrow \langle y_i \rangle + \langle y_j \rangle$.
    • Compute $\langle \lambda y_k \rangle \leftarrow \langle \lambda y_i \rangle + \langle \lambda y_j \rangle$.
  Each party locally adds its shares over the integers.
– Multiplication of an input $x_i$ and a memory value $y_j$:
    • For each $t \in [l]$,
        ∗ Execute MultShares($\left[\!\left[\lambda^{(t)} x_i\right]\!\right]_\lambda, \langle \lambda y_j \rangle$) and output $\langle\!\langle \lambda^{(t)} x_i y_j \rangle\!\rangle$.
        ∗ Run ConvertShares($\langle\!\langle \lambda^{(t)} x_i y_j \rangle\!\rangle, \delta', M, \phi$) and output $\langle \lambda^{(t)} x_i y_j \rangle$.
        ∗ Set $\langle \lambda^{(t)} y_k \rangle \leftarrow \langle \lambda^{(t)} x_i y_j \rangle$.
    • Compute $\langle \lambda x_i y_j \rangle = \sum_{t \in [l]} 2^t \langle \lambda^{(t)} x_i y_j \rangle$.
    • Run MultShares($[\![x_i]\!]_\lambda, \langle \lambda y_j \rangle$) and output $\langle\!\langle x_i y_j \rangle\!\rangle$.
    • Execute ConvertShares($\langle\!\langle x_i y_j \rangle\!\rangle, \delta', M, \phi$) and output $\langle x_i y_j \rangle$.
    • Set a new memory location $k$ to value $y_k = x_i y_j$ by storing
        ∗ Set $\langle \lambda y_k \rangle \leftarrow \langle \lambda x_i y_j \rangle$.
        ∗ Set $\langle y_k \rangle \leftarrow \langle x_i y_j \rangle$.
– Output memory values:
    • If $b = 0$, set $\langle z \rangle \leftarrow \langle y_i \rangle$ otherwise let $\langle z \rangle$ be the additive inverse i.e., $\langle z \rangle \leftarrow -\langle y_i \rangle$.
    • Call the PRF $\phi$ on $(1+n)$ and shift the additive secret by its output meaning: $\langle z \rangle \leftarrow \langle z \rangle + \phi(1+n)$.
    • $\langle O_j \rangle \leftarrow \langle z \rangle$.
    • Output $\langle O_j \rangle$.
```

Fig. 3. Procedures for performing homomorphic operations on secret shares

5 Verifying Computations

The work of [7] mentions a "Las Vegas" style version of HSS in which one of the parties checks for the *potential* of the ConvertShares / DDLog procedure failing at each step. If there was never a chance of failure, then a special flag is set by this party to indicate that the results of the computation are guaranteed to be correct. This method was then shown to provide efficiency improvements for several applications. In particular, for *function secret sharing* applications (denoted "FSS" henceforth; see [6, 8]) in which neither evaluator learns the output (e.g., PIR), this method can be used to reduce the number of parallel invocations required to attain a desired bound on the error probability of the protocol. In this section, we briefly describe a technique to *directly* check correctness which verifies the actual computation, rather than the absence of a potentially "risky" situation arising during DDLog. Since this method of checking does not produce false negatives (erroneously reporting that the protocol failed), we can reduce even further the number of required invocations for a desired overall error bound by a constant fraction. The price we pay for this (in addition to a negligible probability of a false positive), is some extra effort to compute the values used in the check. However, this effort depends *linearly* on the program size, whilst each repetition takes quadratic time in the program size. Hence, we achieve a savings in computation for large programs. We suspect this technique will be most useful in the case of Paillier-based constructions where some of the optimizations of [9] which reduce computation are not readily available. We nevertheless describe the method for both cases, as the ElGamal-based version has a simpler description.

The method works by constructing a sort of "hash" of the intermediate states of the computation in two ways – the states prescribed by multiplicative shares, and the states given by the additive shares after performing DDLog. We first consider the original case of ElGamal-encrypted inputs. Let $\mathbb{G}$ be a group of prime order q, and let $\langle g \rangle = \mathbb{G}$. Let m be the number of multiplication steps in the program being evaluated. Then we denote by $z_i = z_i^0 + z_i^1$ the exponents of the multiplicative sharing of the i-th multiplication step. That is, the players hold $g^{z_i^0}, g^{z_i^1}$. After running DDLog, the players will hold $\overline{z}_i^0, \overline{z}_i^1$, respectively. If the DDLog protocol was successful, it should be the case that $z_i = \overline{z}_i$ for $i = 0, \ldots, m-1$, where $\overline{z}_i = \overline{z}_i^0 + \overline{z}_i^1$. We now define polynomials $P, \overline{P} \in \mathbb{F}_q[X]$ for each of the two potential transcripts:

$$P(X) = \sum_{i=0}^{m-1} z_i X^i, \ \ \overline{P}(X) = \sum_{i=0}^{m-1} \overline{z}_i X^i. \tag{5}$$

Note that each player ($j \in \{0,1\}$) can compute shares of these polynomials $P^j(X) = \sum z_i^j X^i$ and $\overline{P}^j(X) = \sum \overline{z}_i^j X^i$, so that $P = P^0 + P^1$ and $\overline{P} = \overline{P}^0 + \overline{P}^1$. Now consider the polynomial $(P - \overline{P}) \in \mathbb{F}_q[X]$. If DDLog succeeded at each multiplication step, then this polynomial is identically 0. On the other hand, if at any point DDLog failed, this polynomial will be non-zero, and of course will have degree at most $m-1$. Since q is prime, $(P - \overline{P})$ can have at most $m-1$ roots so that

$$\Pr_{\alpha \xleftarrow{\$} \mathbb{Z}_q} \left[(P - \overline{P})(\alpha) = 0 \right] \leq \frac{m-1}{q} = \mathsf{negl}. \tag{6}$$

Thus, with high probability, $[(P - \overline{P})(\alpha) = 0] \iff$ [no errors occurred in DDLog]. For applications like PIR, this observation alone will suffice: we can modify the protocol to send a random α along with the query, and the servers will compute their shares of $(P_j - \overline{P}_j)(\alpha)$, which will be returned with the answers to the query. Note that the shares of $P(\alpha)$ must be computed in the exponent (which can nevertheless be done using Horner's rule), and the shares of $\overline{P}(\alpha)$ are computed directly in $\mathbb{Z}_q$. Hence the total additional cost is m exponentiations and m multiplications. We also mention a few optimizations. First, since each exponentiation will be to the same exponent α, we can pre-compute an addition chain for α and reuse this for all the exponentiations. Second, we note that it is not necessary to choose $\alpha \xleftarrow{\$} \mathbb{Z}_q$. We could for example choose $\alpha \xleftarrow{\$} \left\{1, \ldots, (m-1)2^{80}\right\}$ instead and still achieve the same effect as equation (6), meanwhile reducing the number of multiplications for exponentiations by a factor of two to four (for common choices of $\mathbb{G}$, as of this writing).

From Paillier Encryption. We can also adapt the above to work with Paillier. In this case, $(1+n)$ will serve as our generator g, but since we now work in the larger, composite order group $\mathbb{Z}_{n^2}^{\times}$ (rather than $\langle g \rangle$), a few remarks are in order. First note that if $n = pq$ is an RSA modulus, then for $f \in \mathbb{Z}_n[X]$ with $\deg(f) = d$, f has at most d^2 roots. This follows at once from the Chinese Remainder Theorem: the roots $\alpha \in \mathbb{Z}_n$ of f are in bijective correspondence with the respective pairs of roots (α_p, α_q) of $f_p = (f \bmod p) \in \mathbb{Z}_p[X]$ and $f_q = (f \bmod q) \in \mathbb{Z}_q[X]$. Since $\deg(f_p), \deg(f_q) \leq d$ and since $\mathbb{Z}_p, \mathbb{Z}_q$ are fields, it follows that there can be at most d^2 roots of f in $\mathbb{Z}_n$, and thus the main point of (6) still holds (that is, $\Pr\left[(P - \overline{P})(\alpha) = 0\right] = \mathsf{negl}$).

We also remark on the importance of using Horner's rule in computing $(1+n)^{P_j(\alpha)}$. Before, we were working in a cyclic group, and so the multiplicative shares were of the form $g^{P_j(\alpha)}$ for $j \in \{0, 1\}$. In this case, each player has a sequence of group elements γ_i^0, γ_i^1 such that $\gamma_i^0 \gamma_i^1 = (1+n)^{z_i}$. Naturally we have

$$\left[\prod_{i=0}^{m-1} (\gamma_i^0)^{\alpha^i}\right] \left[\prod_{i=0}^{m-1} (\gamma_i^1)^{\alpha^i}\right] = (1+n)^{P(\alpha)}$$

but at first glance, it seems that it might be somewhat expensive to raise the shares γ_i^j to the (large) exponents α^i: since the order of the group $(n\varphi(n))$ is *not public*, it might seem that this would take work proportional to the length of α^i, which is proportional to the multiplicative depth of the program. Fortunately using Horner's rule prevents us from having to compute or store α^i directly, and instead we can simply exponentiate by α repeatedly.[8] Lastly, since current values of n may be 2048 bits in length, choosing $\alpha \xleftarrow{\$} \left\{1, \ldots, (m-1)^2 \cdot 2^{80}\right\}$ will provide substantial savings. At this point, the protocol follows identically to the above version for ElGamal.

Applications. Applications of the above Las Vegas versions of $\mathsf{ConvertShares}$ include situations where it is unimportant to keep the intermediate states of the computation hidden from the receiver of the output. For example (as noted by [7]), using the scheme as an FSS to perform two-server PIR protocols. The benefit of this approach is that, for a target overall error bound, it further reduces the number of parallel repetitions of the protocol that must be performed to achieve it. Under the (generally wrong) assumption that the intermediate values of the computation are uniform in their domain, it is not hard to show that the

[8]We note that naive polynomial evaluation could also be made reasonable by raising to $\alpha^i \bmod n$, since in any abelian group, if $\prod h_i \in H < G$ with $|H| = n$, then for any $k \in \mathbb{Z}$, $(\prod h_i)^k = (\prod h_i)^{k \bmod n} = \prod (h_i^{k \bmod n})$.

probability of failure for a single round decreases by a factor of $\approx 1/2$. However, as noted this assumption is generally not true. What can be said, is that the smaller the intermediate values are (relative to their domain), the more of an advantage this method provides. For concreteness, an example: assuming half of the intermediate values are 0 and half are 1 (as would hold in expectation for the random case), then if the target error bound was 2^{-80} and the error for a single invocation *of the original* protocol was set to be $1/4$, then our protocol (assuming random intermediate values) would reduce this failure rate to $1/8$ and thus the required number of invocations would decrease from 40 to 27. Again, we note that while the computation cost increases, this increase is *linear* in the multiplicative depth of the program (and polynomial in a security parameter) which provides an advantage for large programs, especially for Paillier-based constructions where many of the speed-ups for DDLog from [9] do not seem available.

6 Conclusions and Future Work

We extend recent breakthrough results by Boyle et al. [7, 9], which under the DDH Assumption, present homomorphic secret sharing and secure function evaluation protocols for branching programs with low communication complexity. We show how to construct similar protocols based on the circular security of the Paillier encryption scheme. In the process we extended their "distributed discrete log" procedures to work over any finite group, and in particular when the discrete log is being sought in a subgroup of unknown order. This technical contribution could be of independent interest and may lead to techniques for proving the security of such protocols under larger classes of computational assumptions.

Our result leaves several interesting open problems:

1. Analyze the circular security assumption on the Paillier encryption scheme and/or come up with alternative schemes with the same functionality that can be proven to be circular secure. This seems a non-trivial question, and as shown by [38], there is no chance for proving a "blanket" result for bit encryption, as there is no *black-box* reduction of circular security to semantic security. Indeed, there have been many results in the recent literature showing separations between the two notions under various assumptions [1, 13, 5, 33, 2, 31, 28, 41].
2. Construct Homomorphic Secret Sharing based on other assumptions. One interesting question here is if we can have HSS based *directly* on LWE which results in more efficient protocols than those based on FHE.
3. Extend the class of functions for which we can break the "circuit barrier" for communication complexity in secure MPC.
4. Explore further optimizations to the Paillier-based protocol. In particular, the work of [9] makes use of PKI for the setup phase, in place of performing general purpose MPC. Their technique seems to leverage heavily a sort of

symmetry that is present in ElGamal, which is not shared by the Paillier encryption scheme: in particular, they make use of the fact that many different secret keys can exist for a single set of common parameters (the group $\mathbb{G}$ and generator g). With Paillier, the modulus n uniquely determines secret information, so it would seem new ideas are required.

5. Empirical data regarding implementations may also be of interest to have a better idea of at what point various trade-offs make sense (for example, making use of the verification process from Section 5 to reduce the number of repetitions vs trying to squash the degree using randomizing polynomials [29, 3]).

References

1. Acar, T., Belenkiy, M., Bellare, M., Cash, D.: Cryptographic agility and its relation to circular encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 403–422. Springer (2010)
2. Alamati, N., Peikert, C.: Three's compromised too: Circular insecurity for any cycle length from (ring-) lwe. In: Annual Cryptology Conference. pp. 659–680. Springer (2016)
3. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. Computational Complexity 15(2), 115–162 (2006)
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC. pp. 1–10 (1988)
5. Bishop, A., Hohenberger, S., Waters, B.: New circular security counterexamples from decision linear and learning with errors. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 776–800. Springer (2014)
6. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 337–367. Springer (2015)
7. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under ddh. In: Annual Cryptology Conference. pp. 509–539. Springer (2016)
8. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303. ACM (2016)
9. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: Optimizing rounds, communication, and computation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 163–193. Springer (2017)
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6(3), 13 (2014)
11. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing 43(2), 831–871 (2014)
12. Brakerski, Z., Vaikuntanathan, V.: Lattice-based fhe as secure as pke. In: Proceedings of the 5th conference on Innovations in theoretical computer science. pp. 1–12. ACM (2014)

13. Cash, D., Green, M., Hohenberger, S.: New definitions and separations for circular security. In: International Workshop on Public Key Cryptography. pp. 540–557. Springer (2012)
14. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 11–19. ACM (1988)
15. Chor, B., Gilboa, N.: Computationally private information retrieval. In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. pp. 304–313. ACM (1997)
16. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)
17. Cohen, J.D., Fischer, M.J.: A robust and verifiable cryptographically secure election scheme (extended abstract). In: 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985. pp. 372–382 (1985)
18. Damgård, I., Jurik, M.: A length-flexible threshold cryptosystem with applications. In: ACISP. pp. 350–364 (2003)
19. Damgrd, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: LNCS. pp. 119–136. Springer-Verlag (2001)
20. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: EUROCRYPT. pp. 24–43 (2010)
21. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing. pp. 169–178. ACM, New York, NY, USA (2009)
22. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520 (2010)
23. Gentry, C., Halevi, S.: Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In: Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. pp. 107–109. IEEE (2011)
24. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 465–482. Springer (2012)
25. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology–CRYPTO 2013, pp. 75–92. Springer (2013)
26. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. pp. 218–229 (1987)
27. Goldwasser, S., Micali, S.: Probabilistic encryption. JCSS 28(2), 270–299 (1984)
28. Goyal, R., Koppula, V., Waters, B.: Separating ind-cpa and circular security for unbounded length key cycles. In: IACR International Workshop on Public Key Cryptography. pp. 232–246. Springer (2017)
29. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on. pp. 294–304. IEEE (2000)
30. Jurik, M.J.: Extensions to the paillier cryptosystem with applications to cryptological protocols. BRICS (2003)
31. Koppula, V., Waters, B.: Circular security separations for arbitrary length cycles from lwe. In: Annual Cryptology Conference. pp. 681–700. Springer (2016)

32. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS. pp. 364–373 (1997)
33. Marcedone, A., Orlandi, C.: Obfuscation(ind-cpa security\ not\ rightarrow circular security). In: International Conference on Security and Cryptography for Networks. pp. 77–90. Springer (2014)
34. Naccache, D., Stern, J.: A new public key cryptosystem based on higher residues. In: Proceedings of the 5th ACM conference on Computer and communications security. pp. 59–66. ACM (1998)
35. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. pp. 113–124. ACM (2011)
36. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. Advances in CryptologyEUROCRYPT'98 pp. 308–318 (1998)
37. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238. Springer (1999)
38. Rothblum, R.D.: On the circular security of bit-encryption. In: Theory of Cryptography, pp. 579–598. Springer (2013)
39. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
40. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Public Key Cryptography. pp. 420–443 (2010)
41. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under lwe. Tech. rep., Cryptology ePrint Archive, Report 2017/276, 2017. http://eprint. iacr. org/2017/276 (2017)
42. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. pp. 160–164 (1982)
43. Yao, A.C.C.: How to generate and exchange secrets. In: Foundations of Computer Science, 1986., 27th Annual Symposium on. pp. 162–167. IEEE (1986)