i-Hop Homomorphic Encryption Schemes

Craig Gentry Shai Halevi Vinod Vaikuntanathan

March 12, 2010

Abstract

A homomorphic encryption scheme enables computing on encrypted data by means of a public evaluation procedure on ciphertexts, making it possible for anyone to transform an encryption of x into an encryption of $f(x)$ (for any function f). But evaluated ciphertexts may differ from freshly encrypted ones, which brings up the question of whether one can keep computing on evaluated ciphertexts. Namely, is it still possible for anyone to transform the encryption of $f(x)$ into an encryption of $g(f(x))$?

An *i-hop* homomorphic encryption is a scheme where the evaluation procedure can be called on its own output upto i times (while still being able to decrypt the result). A *multi-hop* homomorphic encryption is a scheme that is i-hop for all i. In this work we study i-hop and multi-hop schemes, in conjunction with the properties of circuit-privacy (i.e., the evaluation procedure hides the function) and compactness (i.e., the output of evaluation is short). We provide appropriate formal definitions for all these properties and show three constructions:

- We show a DDH-based construction, which is multi-hop and circuit private (but not compact), and where the size of the ciphertext is linear in the size of the composed circuit, but independent of the number of hops.
- More generically, for any constant i, an i-hop circuit-private homomorphic encryption can be constructed from any two-flow protocol for two-party SFE. (Conversely, a two-flow protocol for two-party SFE can be constructed from any 1-hop circuit-private homomorphic encryption.)
- For any polynomial i, an i-hop compact and circuit-private homomorphic encryption can be constructed from a 1-hop compact homomorphic encryption and a 1-hop circuit-private homomorphic encryption, where the size of the public key grows linearly with i. Moreover, a multi-hop scheme can be constructed by making an additional circular-security assumption.

For the first construction, we describe a *re-randomizable* variant of the Yao garbled-circuits. Namely, given a garbled circuit, anyone can re-garble it in such a way that even the party that generated that circuit (in collusion with the intended recipient) will not be able to recognize it. This construction may be of independent interest.

Keywords. BHHO encryption, Homomorphic Encryption, Circuit Privacy, Secure Two-party Computation, Oblivious Transfer, Yao's garbled circuits.

1 Introduction

Computing on encrypted data epitomizes the conflict between privacy and functionality, and received a great deal of attention lately. In the canonical setting of this problem there are two parties

– a client that holds an input x, and a server that holds a function f. The client wishes to learn $f(x)$ using minimal interaction with the server and without giving away information about its input. Similarly, the server may want to hide information about the function f from the client (except, of course, the value $f(x)$). This problem arises in a wide variety of practical applications such as secure cloud computing, searching encrypted e-mail and so on and so forth.

One way to achieve this goal is to have the client encrypt its input x and send the ciphertext to the server, and have the server "evaluate the function f on the encrypted input". The server returns the evaluated ciphertext to the client, who decrypts it and recover the result. An encryption scheme that supports computation on encrypted data is called a *homomorphic* encryption scheme. Namely, in addition to the usual encryption and decryption procedure, it has an *evaluation procedure*, that takes a ciphertext and a function and returns an "evaluated ciphertext", which can then be decrypted to obtain the value $f(x)$.

A trivial implementation of this procedure is for the evaluated ciphertext to include both the original ciphertext and the function f, and for the client to decrypt the original ciphertext and then evaluate f on the result. The problem with this trivial solution is that it does not hide the server's function from the client, and that it does not offload any of the work to the server. We are therefore interested also in the properties of *circuit privacy* (meaning that the evaluated ciphertext hides the function) and *compactness* (meaning roughly that the work involved with decrypting the evaluated ciphertext is less than in computing the function "from scratch"). We note that the first concern can be handled using "generic tools" (such as Yao's garbled circuits [32]), while the latter concern seem to require "special-purpose schemes" (such as the recent scheme of Gentry [12] and its variants [30, 29]).

1.1 Multi-Hop Fully Homomorphic Encryption

Now that we know what homomorphic encryption is, we ask whether it can be used also outside the two-party setting for which it was designed. For example, consider the problem of encrypted e-mail forwarding, where an e-mail encrypted under the public-key of Alice is sent to "alice@yahoo.com", and is promptly forwarded to "alice@gmail.com". Both Yahoo and Google have their own spam-tagging algorithms that they want to apply to incoming e-mails, hence we may want to use a homomorphic encryption scheme so that they can apply these algorithms to the encrypted email. In this example, Yahoo can clearly apply its spam-tagging algorithm to the encrypted incoming e-mails and produce an (encrypted and) tagged set of e-mails. The question is: can Google apply its own spam-tagging algorithm to this ciphertext? The problem is that the ciphertext received by Google is not "fresh", but is the result of a homomorphic evaluation procedure performed by Yahoo!

What we need in this application is a *multi-hop* fully homomorphic encryption scheme, where the homomorphic function evaluation can be applied not only to a fresh ciphertext, but also a ciphertext that was already subjected to another homomorphic evaluation. This multi-hop setting introduces two additional concerns, which are the focus of this work:

Extendability. The first concern is mostly syntactic: the evaluated ciphertexts may be in a different form than fresh ciphertexts, and it is not clear that the evaluation procedure can process this modified form. (Indeed, homomorphic encryption schemes that are derived from generic secure computation protocols tend to have this problem, see below.) Hence the first challenge that we face is Extendability: Can we transform one-hop schemes into multi-hop schemes?

Multi-hop Circuit Privacy. The second concern is security. Multi-hop homomorphic encryption brings up the concern of *multi-hop circuit privacy.* For example, in the mail-forwarding example above, Google may worry that Yahoo! will collude with the sender and receiver of the email, in order to learn something about Google's spam-tagging techniques. For example, in Section 1.2 we show a that although it is easy to modify Yao-based schemes to make them extendable, the result does not provide full circuit privacy.

1.2 A Folklore Construction of Single-Hop FHE

The starting point of our work is the "folklore theorem" which asserts that any two-message protocol for secure two-party computation can be used to construct a *single-hop* fully homomorphic encryption scheme (see, e.g., [16, 4]). The analogy between secure two-party computation and fully homomorphic encryption is best seen by examining the structure of a two-message protocol for secure computation of a function f. Consider the setting where a server holds a function f and a client holds an input x, and the client wishes to receive $f(x)$.

- The client sends to the server a first message that "encodes" its input x, and yet reveals no information about x to a computationally bounded server. In other words, the client's message acts as a semantically secure *encryption* of x.
- The server's response encodes the result of the computation (namely $f(x)$), and yet, reveals no more information to the client about the function f. In other words, the server essentially performs a circuit-private *evaluation* of the function f on an encrypted input.
- The client recovers the result $f(x)$ from the server's message, using her secret randomness. This is the *decryption* procedure.

There is a slight mismatch, however, between the settings of secure function evaluation and public-key encryption. In the setting above we have the client performing both encryption and decryption, whereas in public-key encryption these two operations can be performed by two separate entities. Fortunately, this mismatch is easily fixed: The solution is to have the client choose a public and private key for some semantically secure encryption scheme, and ask the encryptor to publish an encryption of its randomness (for the SFE protocol) under the client's public key. Now, the client (who knows the secret key) can recover the randomness used by the encryptor in the SFE protocol, and use it to recover the result $f(x)$ from the server's message. This construction therefore achieves a *circuit-private, single-hop fully homomorphic encryption* scheme from *any* two-message secure two-party computation protocol (and an auxiliary semantically-secure encryption scheme).

Is the folklore scheme extendable? Consider the setting where there is a client who holds an input x, two evaluators (servers) E_1 and E_2 who hold functions f_1 and f_2 respectively, and the client wishes to receive $f_2(f_1(x))$. To achieve this, the client would like to compute an encryption of x and send it to the first evaluator, who computes an encryption of $f_1(x)$ and passes it to the second evaluator. The question we ask is: Can E_2 now compute on the output of E_1? For generic two-party computation protocols, we only have a partial answer to this question: In Theorem 2 we show how to extend the scheme from above to more than one hop, but the size of the ciphertext grows by a polynomial factor for every hop (and hence we can only do so for constant many hops).

It is easy to see, however, that a protocol based on Yao's garbled circuits [31, 19] is syntactically extendable. Recall that in Yao's garbled circuit construction, the server (who has a circuit) chooses

	Number of Hops d	Size of the Ciphertext (in terms of d and the functions $f_1, \ldots, f_d$)	Assumption
Construction 1 (see Theorem 1)	any polynomial in n	$\mathsf{poly}(n) \cdot \left(\sum_{i=1}^{d} \lvert f_i \rvert\right)$	decisional Diffie-Hellman
Construction 2 (see Theorem 2)	any constant	$O(n^d) \cdot \left(\sum_{i=1}^{d} \lvert f_i \rvert\right)$	any PKC + OT
Construction 3 (see Theorem 3)	any polynomial in n	$\mathsf{poly}(n)$	any compact FHE + OT

Figure 1: Our Results. PKC=Public-key Encryption, FHE=Fully Homomorphic Encryption, OT=Oblivious Transfer.

two random labels for every wire in the circuit, and for every gate it computes a "gate gadget" that allows the client to learn one of the output labels if it knows one label on each input wire. The collection of all these gadgets is called the "garbled circuit." The server sends the garbled circuit to the client, and engages in an *oblivious transfer protocol* where it reveals to the client exactly one of the two labels on every input wire (without learning which was revealed). The client uses the gadgets to learn one label on each wire, all the way to the output wires of the circuit. The server also provides the client with a mapping between the output labels and zero/one, hence allowing the client to learn the output.

Clearly, this construction is extendable: the second sever E_2 receives the garbled circuit from the first server E_1, and it can now just use E_1's output labels for its own input labels, thus connecting these two circuits and proceeding with the protocol.

Is the Yao-based scheme multi-hop circuit-private? We note that this extendable scheme does offer a weak form of circuit-privacy: if only the client is corrupted, then the composed garbled circuit looks as if it was generated "from scratch" on the compositions of the two circuits, and thus it hides them from the client.

However, this argument breaks down completely when E_1 colludes with the client. Now, E_1 knows both the labels for each input wire of the garbled circuit that E_2 prepares. Thus, from the point of view of E_1, the output of E_2 is not garbled at all, in fact E_1 can completely recover f_2. Our work shows various ways to remedy this problem, and construct multi-hop circuit-private fully homomorphic encryption schemes.

1.3 Our Results

We formally define the notion of multi-hop circuit-privacy, and show three constructions of i-hop fully homomorphic encryption schemes, for different values of i, and under different assumptions.

Definition of Multi-Hop Fully Homomorphic Encryption. Informally, in an i-hop FHE scheme, a sequence of i functions $f_1, \ldots, f_i$ can be homomorphically evaluated one by one, on a ciphertext c produced by encrypting a message x. This is pictorially depicted as follows. (Here $E_1, \ldots, E_d$ denote the d players – evaluators – that hold the functions $f_1, \ldots, f_d$).

$$\text{Encryptor}(x) \overset{c_0 = Enc(x)}{\rightarrow} E_1(f_1, c_0) \overset{c_1}{\rightarrow} \ldots \; E_{j-1}(f_{j-1}, c_{j-2}) \overset{c_{j-1}}{\rightarrow} \boxed{E_j(f_j, c_{j-1})} \overset{c_j}{\rightarrow} \ldots \overset{c_d}{\rightarrow} \text{Decryptor}$$

A multi-hop FHE scheme is simply an i-hop scheme that works for any (polynomial) i.

The definition of multi-hop circuit privacy requires that for every $j \in [d]$, even if all the evaluators except E_j combine their information, they still learn no information about f_j (other than its input and output). The formal definition is simulation-based, extending the (1-hop) definition of Ishai and Paskin [16]. In this work we only deal with the honest-but-curious setting, and only consider the case where all but one of the evaluators are corrupted (as opposed to an arbitrary subset of them). Treatment of the more general cases is left for future work.

Construction 1. In Section 4 we describe a multi-hop FHE scheme that can handle any polynomial number of hops, and is semantically secure under the decisional Diffie Hellman assumption. The size of the ciphertext in this scheme grows linearly with the size of the functions that are evaluated on the ciphertext, but independently of the number of hops.

Theorem 1 (Informal)**.** *Under the decisional Diffie-Hellman assumption, there is a public-key fully homomorphic encryption scheme Π_1 such that for any polynomial $p(n)$, Π_1 is circuit private for $p(n)$ hops. There is a fixed polynomial $q(n)$ such that on evaluating functions $f_1, \ldots, f_d$ on a fresh ciphertext, the resulting ciphertext has size $q(n) \times \left(\sum_{i=1}^{d} |f_i|\right)$.*

This encryption scheme essentially amends the Yao-garbled-circuit construction from previous section (which was extendable but not multi-hop circuit-private). The problem there was that the garbled circuit produced by the second evaluator E_2 contains (as a sub-circuit) the garbled circuit produced by E_1; this reveals non-trivial information about the function f_2 to the first evaluator. The solution to this problem is to come up with a way to *re-randomize Yao garbled circuits.* Roughly speaking, this is a procedure that takes a garbled circuit and constructs a *random garbled circuit* for the same function.

We show a new variant of the garbled circuit construction for which such a re-randomization can be done. For the construction, we rely on the oblivious-transfer protocol of Naor-Pinkas and Aiello-Ishai-Reingold [22, 2], and on the encryption scheme of Boneh-Halevi-Hamburg-Ostrovsky [7] (both of which are based on the decisional Diffie-Hellman assumption, and have nice additive homomorphic properties).

Construction 2. In Section 3 we show how to convert *any one-hop circuit-private homomorphic encryption scheme* into a scheme that handles i hops for any constant i. The ciphertext in this scheme grows by a factor of $n^{O(i)}$ after i hops, where n is the security parameter. Thus, the scheme is viable only for a constant number of hops. Since one-hop circuit-private homomorphic encryption can be constructed from semantically secure encryption and two-message SFE, we get the following:

Theorem 2 (Informal)**.** *Under the assumption that semantically secure public-key encryption schemes and two-message secure function evaluation protocols exist, there is a public-key fully homomorphic encryption scheme Π_2 such that for any constant d, Π_2 is circuit private for d hops. There is a fixed polynomial $q(n)$ such that on evaluating functions $f_1, \ldots, f_d$ on a fresh ciphertext, the resulting ciphertext has size at most $q(n)^d \times \left(\sum_{i=1}^{d} |f_i|\right)$.*

Construction 3. A problem with both the schemes above is that the size of the ciphertext grows with the functions that are evaluated on the ciphertext. On the other hand, the recently proposed fully homomorphic encryption scheme of Gentry [12] (as well as the subsequent constructions [30, 29]) achieve *compact* ciphertexts. Namely, the size of the ciphertext after evaluating a function is *independent* of the circuit-size of the function.

In Section 3.1 we show how to combine two single-hop FHE schemes – a *compact* (and not circuit-private) homomorphic encryption scheme, and a circuit-private (but not compact) homomorphic encryption scheme – to construct a multi-hop FHE scheme that is both compact as well as circuit-private for any polynomial number of hops.

Theorem 3 (Informal)**.** *Assume that there exist a single-hop compact (but not circuit-private) FHE scheme, and a single-hop circuit-private (but not compact) FHE scheme. Then, for every polynomial $p(n)$, there is an encryption scheme Π_3 such that Π_3 is circuit private for $p(n)$ hops. There is a fixed polynomial $q(n)$ such that on evaluating functions $f_1, \ldots, f_d$ on a fresh ciphertext, the resulting ciphertext has size* $\mathsf{q}(\mathsf{n})$ *(independent of the size of the functions f_j).*

Since a single-hop circuit private (but not compact) FHE scheme can be constructed from any two-message protocol for secure function evaluation, the result above holds under the assumption that there exists a single-hop compact (but not necessarily circuit-private) encryption scheme, and a two-message SFE protocol.

Finally, the scheme Π_3 can handle only a fixed polynomial number of hops. If we assume some circular-security assumption, then the scheme can be modified to another scheme Π_3' that handles *any polynomial* number of hops.

1.4 Related Work

Rivest et al. [27] proposed the notion of homomorphic encryption shortly after the invention of RSA. The first homomorphic encryption with a proof of semantic security based on a well-defined assumption was proposed by Goldwasser-Micali [13]. Some other additively homomorphic encryption schemes include Benaloh [5], Naccache-Stern [21], Okamoto-Uchiyama [24], Paillier [25], and Damgard-Jurik [8]. ElGamal [10] is multiplicatively homomorphic. Boneh-Goh-Nissim [6] permits computation of quadratic formulas (e.g., 2-DNFs) over ciphertexts. One can construct additively homomorphic encryption schemes from lattices or linear codes [26, 18, 1, 20, 3]. These schemes are extendable and re-randomizable. However, they cannot handle arbitrary functions f.

Sanders, Young and Yung [28] show that one can use a statistically circuit-private additively homomorphic encryption scheme to construct a statistically circuit-private scheme that can handle arbitrary fan-in-two boolean circuits, where the ciphertext size increases exponentially with the depth of the circuit. Their scheme can therefore feasibly evaluate circuits in NC1. Ishai and Paskin [16] show how to evaluate branching programs. In their scheme Eval outputs a ciphertext whose length is proportional to the *length* of the branching program, but independent of its *size.* (Of course, the complexity of evaluation is still proportional to the size of the branching program.)

The "Polly Cracker" scheme [11], and schemes by Aguilar Melchor et al. [20] and Armknecht and Sadeghi [3] also handle arbitrary functions, but with rapidly expanding ciphertext size, especially for multiplication gates. The extent to which these schemes can be made circuit-private is unclear.

Recently, several compact fully homomorphic encryption (FHE) schemes have been proposed [12, 30, 29]. These are compact and extendable, and they provide non-generic re-randomization techniques to get multi-hop circuit privacy. Also, while technically compact – i.e., the ciphertext size does not depend on the function being evaluated – ciphertexts are larger than plaintexts by a factor that is a large polynomials in the security parameter. For circuits that are not too large, our multi-hop scheme may have more compact ciphertexts under reasonable security settings.

1.5 Organization

For didactic reasons, it is easier to present the "generic" Constructions 2 and 3 before we present the DDH-based Construction 1. We begin in Section 2 with definitions, then present Constructions 2 and 3 in Section 3, and Construction 1 is presented in Section 4.

2 Definitions

Nearly all our definitions rely on a security parameter, which is usually implicit. Adversarial algorithms are always nonuniform. We use $x \leftarrow X$ and $x \in_R S$ for drawing from a distribution and choosing uniformly from a set. We call a procedure efficient if it runs in polynomial time in its input. We say that distributions are statistically close if the statistical distance between them is negligible in the security parameter. We say that two distributions are indistinguishable if any efficient distinguisher has only negligible advantage in distinguishing them.

2.1 Two-party Secure Function Evaluation

We fix a particular "universal circuit evaluator" $U(\cdot,\cdot)$, taking as input a description of a function f and an argument x, and returning $f(x)$. Using U we can view every bit-string f as describing a function (where $f(x)$ is just a shorthand for $U(f,x)$). Below we sometimes assume that U itself is implemented as a boolean circuit.

A two-flow protocol for secure two-party computation (to be run by Alice the client and Bob the server), is implemented by three polynomial-time procedures $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$, where:

- $\mathsf{SFE1}(x)$ is a randomized procedure that Alice runs, taking as input the security parameter and a string x. It outputs m_1, which is the first-flow message of the SFE protocol, as well as some state to be used later, $(m_1, r_1) \leftarrow \mathsf{SFE}(x)$. We assume that r_1 includes in particular all the randomness that was used in the computation.
- $\mathsf{SFE2}(f, m_1)$ is a randomized procedure that Bob runs, taking as input the security parameter, the first-flow message m_1, and a circuit f. The output is m_2, the second-flow message of the SFE protocol, and the randomness that was used in the computation, $(m_2, r_2) \leftarrow \mathsf{SFE2}(f, m_1)$.
- $\mathsf{SFE\text{-}Out}(r_1, m_2)$ is a procedure that takes Alice's state r_1 and Bob's second-flow message m_2, and outputs some y.

We often neglect to mention some of the outputs of these procedures, writing, e.g., $m_2 \leftarrow \mathsf{SFE2}(\cdots)$. Correctness of the SFE protocol demands that the value y thus computed is equal to $f(x)$, except with negligible probability over the randomness of Alice and Bob. The input-privacy requirements for Alice and Bob are defined next.

Definition 1 (Client privacy). *A protocol $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ enjoys client privacy if for any two inputs x, x' of the same length, the distributions $\mathsf{SFE1}(x)$ and $\mathsf{SFE1}(x')$ are indistinguishable (even given x, x').*

Definition 2 (Server privacy - honest-but-curious). *A protocol $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ enjoys server privacy in the honest-but-curious model if there exists a polynomial-time simulator Sim such that for every input x and function f, the two distributions $\mathcal{D}_R(f,x), \mathcal{D}_S(f,x)$ below are indistinguishable:*

$$\mathcal{D}_R(f,x) \overset{\text{def}}{=} \{(m_1, r_1) \leftarrow \mathsf{SFE1}(x),\ m_2 \leftarrow \mathsf{SFE2}(f, m_1),\ \ \mathsf{output}\ (f, x, r_1, m_1, m_2)\}$$
$$\mathcal{D}_S(f,x) \overset{\text{def}}{=} \{(m_1, r_1) \leftarrow \mathsf{SFE1}(x),\ m_2 \leftarrow \mathsf{Sim}(x, f(x), m_1),\ \ \mathsf{output}\ (f, x, r_1, m_1, m_2)\}$$

2.2 Homomorphic Encryption

A homomorphic encryption scheme consists of four efficient procedures, $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. KeyGen takes the security parameter and outputs a secret/public key-pair, Enc takes the public key and plaintext and outputs a ciphertext, and Dec takes the secret key and a ciphertext and outputs a plaintext. The Eval procedure takes a description of a function, the public key, and a ciphertext, and outputs another ciphertext.

Also here to we sometimes use the convention that these procedures output the randomness that they used, for example writing $(c', r') \leftarrow \mathsf{Eval}(\text{PK}, f, c)$ rather than just $c' \leftarrow \mathsf{Eval}(\text{PK}, f, c)$.

Multi-hop evaluation. We extend the Eval procedure from single functions to a sequence of functions in the natural way. Below we say that an ordered sequence of functions $\vec{f} = \langle f_1, \ldots, f_t \rangle$ is *compatible* if the output length of f_j is the same as the input length of f_{j+1} for all j. For $\vec{f}$ a compatible sequence of t functions as above, we denote its j prefix by $\vec{f_j} = \langle f_1, \ldots, f_j \rangle$. The composed function $f_t(\cdots f_2(f_1(\cdot))\cdots)$ is denoted $(f_t \circ \cdots \circ f_1)$.

We define an extended procedure Eval^* that takes as input the public key, a compatible sequence $\vec{f} = \langle f_1, \ldots, f_t \rangle$, and a ciphertext c_0. For $i = 1, 2, \ldots, t$ it sets $c_i \leftarrow \mathsf{Eval}(\text{PK}, f_i, c_{i-1})$, outputting the last ciphertext c_t.

Definition 3 (i-Hop Homomorphic Encryption)**.** *Let $i = i(k)$ be a function of the security parameter. A scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is an i-hop homomorphic encryption scheme if for every compatible sequence $\vec{f} = \langle f_1, \ldots, f_t \rangle$ with $t \le i$ functions, every input x to f_1, every* (PK, SK) *in the support of* KeyGen*, and every c in the support of* $\mathsf{Enc}_{\text{PK}}(x)$*,*

$$\mathsf{Dec}_{\text{SK}}\big(\mathsf{Eval}^*(\text{PK}, \vec{f}, c)\big) = (f_t \circ \cdots \circ f_1)(x)$$

We say that $\mathcal{E}$ is multi-hop *homomorphic encryption if it is i-hop for any polynomial i.*

We note that a 1-hop homomorphic encryption is just the usual notion of homomorphic encryption, as formalized, e.g., in [16, Def 5].

2.2.1 Semantic security, circuit privacy, and compactness

Semantic security (aka CPA security) [14] is defined as usual, regardless of Eval. We provide the definition here for self containment.

Definition 4 (Semantic security)**.** *A scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is semantically secure, is any efficient adversary $\mathcal{A}$ has at most a negligible advantage in the following game: First* KeyGen *is run to produce* (PK, SK) *and $\mathcal{A}$ is given* PK*. Then $\mathcal{A}$ produces two* target messages *M_0, M_1 of the same length. Then a bit σ is chosen at random $\sigma \in_R \{0, 1\}$, and the adversary gets back the* challenge ciphertext*, which is computed as $c \leftarrow \mathsf{Enc}_{\text{PK}}(M_\sigma)$. Finally $\mathcal{A}$ outputs a guess σ' for σ. The advantage of $\mathcal{A}$ is defined as* $\Pr[\mathcal{A}$ *outputs* $\sigma' = 1 \mid \sigma = 1] - \Pr[\mathcal{A}$ *outputs* $\sigma' = 1 \mid \sigma = 0]$.

We sometimes refer to the game above as the *CPA game.*

Next we define circuit privacy. Here we view the operation of Eval^* as a multi-party protocol with one party per evaluated function, and formalizes circuit-privacy as the usual input-privacy property for these parties: Roughly, even if the recipient who holds the secret key colludes with all the parties but one, the circuit of that one party still remains hidden, except perhaps (its size and) the value that this circuit outputs on a single input.

Definition 5 (Circuit privacy - honest-but-curious)**.** *An i-hop homomorphic scheme* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is circuit-private if there exists an efficient simulator* Sim *such that for every compatible sequence of functions* $\vec{f} = \langle f_1, \ldots, f_t \rangle$ *with* $t \leq i$*, every* $j \leq t$*, every input* x *for* f_1*, every* (PK, SK) *in the support of* KeyGen*, and every ciphertext* c_{j-1} *in the support of* $\mathsf{Eval}^*\big(\text{PK}, \vec{f}_{j-1}, \mathsf{Enc}_{\text{PK}}(x)\big)$*, the following two distributions are indistinguishable (even given* $x, \vec{f}_j$ *and* SK*):*

$$\mathsf{Eval}(\text{PK}, f_j, c_{j-1}) \text{ and } \mathsf{Sim}\big(\text{PK}, (f_1 \circ \cdots \circ f_{j-1})(x), (f_1 \circ \cdots \circ f_j)(x), c_{j-1}\big)$$

We remark that Definition 5 can be extended in several different ways. One obvious extension would consider the malicious case (with or without assuming that the public key was created honestly). Another extension is to consider a more general adversarial structure, where the attacker can corrupt an arbitrary subset of the players (encryptor, evaluators, and recipient), and we still want to ensure the privacy of the non-corrupted ones. Another extension to Definitions 5 and 3 is to consider an arbitrary networks of functions (not just a single chain). Yet another extension is to strengthen the privacy guarantee, requiring that Eval^* hides not only the functions that the nodes compute but also the structure of the network itself (e.g., the number of functions in the chain). We leave all of these extensions to future work.

Definition 6 (Compactness)**.** *A scheme* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is i-hop compact homomorphic if there exists a polynomial* $p(\cdot)$ *in (only) the security parameter, such that decryption of any ciphertext w.r.t., security parameter* k *can be implemented by a circuit of size at most* $p(k)$ *(even for evaluated ciphertext).*

Namely, for every value of k *there exists a circuit* $\mathsf{Dec}^{(k)}$ *of size size at most* $p(k)$*, such that the i-Hop property from Definition 3 holds for that decryption circuit.*

The name "compactness" is justified by the fact that the evaluated ciphertexts cannot grow beyond $p(k)$ (regardless of f), if they are to be decrypted by a $p(k)$-size circuit. We comment that compactness and circuit privacy together are still formally weaker then the Ishai-Paskin notion of "privacy with size hiding" [16, Def 8].

3 From One-hop to i-hop Homomorphic Encryption

Below we show how to extend "generically" any 1-hop circuit-private homomorphic encryption schemes into an i-hop scheme for any constant $i > 0$. The price that we pay for our transformation is that the complexity of the i-hop scheme grows as $k^{O(i)}$ (for security parameter k).

The idea is that each evaluator (with function f_j) in the chain, upon receiving the "evaluated ciphertext" c_{j-1} from its predecessor, applies again the evaluation procedure but not to its original function f_j. Rather it applies it to the concatenation of f_j with the decryption function of the underlying scheme, namely to $F_{f_j, c_{j-1}}(\cdot) \overset{\text{def}}{=} f\big(\mathsf{Dec}(\cdot\,; c_j)\big)$. Correctness follows because (by induction) $\mathsf{Dec}(s; c_j) = (f_{j-1} \circ \cdots f_1)(x)$.

Construction 2: from 1-hop to constant-hops. Let $\mathcal{H} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a circuit-private one-hop homomorphic encryption scheme, and let d be a parameter, and we show how to construct a circuit-private d-Hop homomorphic encryption $\mathcal{H}_d^* = (\mathsf{KeyGen}^*, \mathsf{Enc}^*, \mathsf{Eval}^*, \mathsf{Dec}^*)$.

$\underline{\mathsf{KeyGen}^*}$**:** For $j = 0, 1, \ldots, d$, run the key generation of $\mathcal{H}$ to get $(\text{PK}_j, \text{SK}_j) \leftarrow \mathsf{KeyGen}()$. Then, for $j = 0, 1, \ldots, d-1$ sets $\alpha_j \leftarrow \mathsf{Enc}(\text{PK}_{j+1};\ \text{SK}_j)$ (and set $\alpha_d = \perp$). The public key is $\text{PK}^* = \{(\text{PK}_j, \alpha_j) : j = 0, 1, \ldots, d\}$, and the secret key is $\text{SK}^* = (\text{SK}_0, \ldots, \text{SK}_d)$.

$\underline{\mathsf{Enc}(\text{PK}^*;\ x)}$**:** Set $c_0 \leftarrow \mathsf{Enc}(\text{PK}_0; x0)$ and output $[\mathsf{level}\text{-}0, c_0]$.

$\underline{\mathsf{Eval}^*(\text{PK}^*, [\mathsf{level}\text{-}j, c_j], f_{j+1})}$**:** Check that $j < d$, then compute the description of the function:

$$F_{f_{j+1},c_j}(s) \overset{\text{def}}{=} f_{j+1}\big(\mathsf{Dec}(s;\ c_j)\big).$$

(Note that f_{j+1}, c_j are "hard wired" inside F_{f_{j+1},c_j}.) Set $c_{j+1} \leftarrow \mathsf{Eval}(\text{PK}_{j+1};\ F_{f_{j+1},c_j},\ \alpha_j)$ and output $[\mathsf{level}\text{-}(j+1), c_{j+1}]$.

$\underline{\mathsf{Dec}^*(\text{SK}^*;\ [\mathsf{level}\text{-}j, c_j])}$**:** Check that $j \le d$, then compute and output $y \leftarrow \mathsf{Dec}(\text{SK}_j, c_j)$.

Theorem 4. *The scheme $\mathcal{H}^*$ above is d-hop circuit private.*

Proof (sketch) Correctness is easy to establish by induction. Fix some compatible sequence of functions $\vec{f} = \langle f_1, \ldots, f_t \rangle$ with $t \le d$ and some input x to f_1. Clearly correctness holds for the fresh ciphertext [level-0, c_0], this is decrypted to x just by correctness of the underlying scheme. Assuming now that correctness holds for j and we prove for $j+1$. By 1-hop correctness of the underlying private scheme we have

$$\begin{aligned}
\mathsf{Dec}(\text{SK}_{j+1};\ c_{j+1}) &= \mathsf{Dec}(\text{SK}_{j+1};\ \mathsf{Eval}(\text{PK}_{j+1}; F_{f_{j+1},c_j}, \alpha_j)) \\
&= \mathsf{Dec}(\text{SK}_{j+1};\ \mathsf{Eval}(\text{PK}_{j+1}; F_{f_{j+1},c_j}, \mathsf{Enc}(\text{PK}_{j+1}; \text{SK}_j))) \overset{(a)}{=} F_{f_{j+1},c_j}(\text{SK}_j) \\
&\overset{(b)}{=} f_{j+1}(\mathsf{Dec}(\text{SK}_j;\ c_j)) \overset{(c)}{=} f_{j+1}((f_j \circ \cdots f_1)(x)) = (f_{j+1} \circ \cdots f_1)(x)
\end{aligned}$$

where equality (a) follows since $\mathcal{H}$ is one-hop homomorphic, equality (b) follows by definition of F_{f_{j+1},c_j}, and equality (c) follows from the induction hypothesis.

Semantic security of $\mathcal{H}^*$ follows trivially from that of $\mathcal{H}$ (used for both the encryption of x and the encryptions of each secret-key SK_j under the next public key PK_{j+1}). Similarly, d-hop circuit privacy follows trivially from the 1-hop privacy of $\mathcal{H}$: The simulator for node j just uses the underlying 1-hop simulator to directly generate the ciphertext c_j. ■

Complexity. For a generic (non-compact) homomorphic encryption, the size of the circuit for $\mathsf{Dec}(\cdots \mathsf{Eval}(f, c))$ could be larger than the size of the circuit for f by a *poly*(k) factor (with k the security parameter). Indeed, this is the case for schemes that are derived from generic SFE protocols. Hence the size of the circuit for $F_{f_d,c_{d-1}}(\cdot)$ in our construction is

$$k^{O(1)}(\cdots k^{O(1)} \cdot (k^{O(1)} \cdot |f_1| + |f_2|)) \cdots) + |f_t| \;=\; \sum_{j=1}^{t} |f_j| \cdot k^{O(t-j)} \;\approx\; \Big(\sum_{j=1}^{t} |f_j|\Big) \cdot k^{O(t)}$$

which means that the ciphertext c_j has size $k^{O(t)}$ times the size of the f_j's.

Remark 1. *We note that the parameter d can be larger than a constant, thus permitting the evaluation of any constant number of hops (as opposed to having a particular constant parameter d).*

We also note that we can shorten the public key by essentially putting it in the ciphertext. Specifically, we add to the construction an auxiliary semantically secure encryption scheme $\mathcal{E}$, and the key generation is just the key-generation of $\mathcal{E}$. Then the encryption routine runs first the KeyGen^* *routine from above and then the* Enc^* *routine, adding to the ciphertext the entire public key* PK^**, as well as an encryption of* SK^* *under the public key of the auxiliary scheme $\mathcal{E}$. On decryption, the receiver first recovers* SK^* *(using the auxiliary secret key that it knows), and then runs the decryption* Dec^* *from above.*

3.1 Compact and Circuit-Private Homomorphic Encryption

From the construction above we know that it is possible to construct an i-hop circuit-private scheme from a 1-hop circuit-private scheme (which implies a two-flow private SFE protocol). The complexity of the result grows with n^i, where n is the blowup factor of the one-hop scheme (i.e., the amount by which $\mathsf{Dec}(\cdots \mathsf{Eval}(f, \cdots))$ is larger than f). We note that this means in particular that if we have a *compact 1-hop circuit-private scheme*, then we could use it to get a compact i-hop circuit-private scheme, for any polynomial i (since the circuits that are involved do not grow).

Below we show that given a 1-hop scheme which is compact but not private, and another 1-hop scheme which is private but not compact, we can combine them to get a 1-hop scheme which is both compact and private (and thus also i-hop compact and private schemes for all i). The idea is simply to iterate the two schemes at every hop. First we apply the private scheme to the function f that we want to evaluate, thus getting a "private ciphertext" which is large but does not reveal information about f. Then we apply the compact scheme to the decryption function of the private scheme, in essence "compressing" the large ciphertext into a compact one which is still decrypted to the same value. The result is clearly compact (since it outputs the "compact ciphertext"). It is also private since the compact ciphertext only depends on the function f is via the value of the intermediate "private ciphertext", and even if we were to give the adversary the "private ciphertext" itself, it would still not violate the privacy of f.[1]

We note that when using this technique, we again get a "hard-wired" parameter d that limits the number of hops that we can handle: to get a d-hop scheme, the public key (or ciphertext) must have size linear in d. (Differently from Construction 2, however, now d can be any polynomial.) This limitation can be circumvented by relying on circular security of the resulting 1-hop scheme, see Remark 2.

Construction 3: from compact 1-hop to multi-hop. Let $p\mathcal{H} = (p\mathsf{KeyGen}, p\mathsf{Enc}, p\mathsf{Eval}, p\mathsf{Dec})$ be a circuit-private homomorphic 1-hop encryption scheme (that need not be compact), and let $c\mathcal{H} = (c\mathsf{KeyGen}, c\mathsf{Enc}, c\mathsf{Eval}, c\mathsf{Dec})$ be a compact homomorphic 1-hop encryption scheme (that need not be private).

Let d (=$\mathsf{poly}(n)$) be a parameter of the system (that represents the number of hops that we are shooting for). We construct a *compact and circuit-private* d-hop homomorphic encryption scheme $\mathcal{H}_d^* = (\mathsf{KeyGen}^*, \mathsf{Enc}^*, \mathsf{Eval}^*, \mathsf{Dec}^*)$ as follows.

$\underline{\mathsf{KeyGen}^*}$**:** Run each of $p\mathsf{KeyGen}, c\mathsf{KeyGen}$ for $d+1$ times, to get for $j = 0, 1, \ldots, d$:

$$\begin{array}{rll} & (p\textsc{pk}_j, p\textsc{sk}_j) \leftarrow p\mathsf{KeyGen}, & (c\textsc{pk}_j, c\textsc{sk}_j) \leftarrow c\mathsf{KeyGen}, \\ \text{and for } j < d \text{ also:} & \alpha_j \leftarrow p\mathsf{Enc}\big(\underbrace{p\textsc{pk}_j}_{\text{key}};\ \underbrace{c\textsc{sk}_j}_{\text{ptxt}}\big), & \beta_j \leftarrow c\mathsf{Enc}\big(\underbrace{c\textsc{pk}_{j+1}}_{\text{key}};\ \underbrace{p\textsc{sk}_j}_{\text{ptxt}}\big) \end{array}$$

Defining $\alpha_d = \beta_d = \perp$, the public key is the set $\textsc{pk}^* = \{(p\textsc{pk}_j c\textsc{pk}_j, \alpha_j, \beta_j) : j = 0, 1, \ldots, d\}$, and the secret key is $\textsc{sk}^* = (c\textsc{sk}_0, c\textsc{sk}_1, \ldots, c\textsc{sk}_d)$.

$\underline{\mathsf{Enc}^*(\textsc{pk}^*;\ x)}$**:** Set $c_0 \leftarrow c\mathsf{Enc}(c\textsc{pk}_0;\ x)$ and output $\big[\text{level-0},\ c_0\big]$.

[1]We comment that iterating the two systems in the opposite order also works: we can apply the compact scheme to the function f and the private scheme to the decryption of the compact one.

$\underline{\mathsf{Eval}^*(\text{PK}^*, [\mathsf{level}\text{-}j, c_j], f_{j+1})}$**:** Check that $j < d$, then compute the description of the function $F_{f_{j+1},c_j}(s) \overset{\text{def}}{=} f(c\mathsf{Dec}(s;\ c_j))$, and set

$$c'_j \leftarrow p\mathsf{Eval}(p\text{PK}_j;\ F_{f_{j+1},c_j},\ \alpha_j).$$

Then compute the description of the function $G_{c'_j}(s) \overset{\text{def}}{=} p\mathsf{Dec}(s;\ c'_j))$, and set

$$c_{j+1} \leftarrow c\mathsf{Eval}(c\text{PK}_{j+1};\ G_{c'_j},\ \beta_j).$$

Output $\big[\mathsf{level}\text{-}(j+1),\ c_{j+1}\big]$.

$\underline{\mathsf{Dec}^*(\text{SK}^*;\ [\mathsf{level}\text{-}j, c_j])}$**:** Check that $j \le d$, then compute and output $y \leftarrow c\mathsf{Dec}(c\text{SK}_j;\ c_j)$.

Theorem 5. *For any $d = \mathsf{poly}(n)$, the scheme $\mathcal{H}^*_d$ above is a compact circuit private d-hop homomorphic encryption scheme.*

Proof (sketch) Correctness is again proved by easy induction. Fix some compatible sequence of functions $\vec{f} = \langle f_1, \ldots, f_t\rangle$ with $t \le d$ and some input x to f_1. Clearly correctness holds for the fresh ciphertext [level-0, c_0], this is decrypted to x just by correctness of the underlying compact scheme. Assuming now that correctness holds for j and we prove for $j+1$. By 1-hop correctness of the underlying private scheme we have

$$\begin{aligned} p\mathsf{Dec}(p\text{SK}_j;\ c'_j) &= p\mathsf{Dec}(p\text{SK}_j;\ p\mathsf{Eval}(p\text{PK}_j; F_{f_{j+1},c_j}, \alpha_j)) \\ &= p\mathsf{Dec}(p\text{SK}_j;\ p\mathsf{Eval}(p\text{PK}_j; F_{f_{j+1},c_j}, p\mathsf{Enc}(p\text{PK}_j; c\text{SK}_j))) \\ &= F_{f_{j+1},c_j}(c\text{SK}_j) \overset{(a)}{=} f_{j+1}(c\mathsf{Dec}(c\text{SK}_j;\ c_j)) \overset{(b)}{=} f_{j+1}(\vec{f_j}(x)) = \vec{f}_{j+1}(x) \end{aligned}$$

where equality (a) follows by definition of F_{f_{j+1},c_j} and equality (b) follows from the induction hypothesis. Now, by 1-hop correctness of the underlying compact scheme we have

$$\begin{aligned} c\mathsf{Dec}(c\text{SK}_{j+1};\ c_{j+1}) &= c\mathsf{Dec}(c\text{SK}_{j+1};\ c\mathsf{Eval}(c\text{PK}_{j+1}; G_{c'_j}, \beta_j)) \\ &= c\mathsf{Dec}(c\text{SK}_{j+1};\ c\mathsf{Eval}(c\text{PK}_{j+1}; G_{c'_j}, c\mathsf{Enc}(c\text{PK}_{j+1}; p\text{SK}_j))) \\ &= G_{c'_j}(p\text{SK}_j) = p\mathsf{Dec}(p\text{SK}_j;\ c'_j) = \vec{f}_{j+1}(x) \end{aligned}$$

Hence [level-$(j+1)$, c_{j+1}] will indeed be decrypted to $\vec{f}_{j+1}(x)$, as needed.

Semantic security of $\mathcal{H}^*$ follows trivially from that of the two underlying schemes (where we only need the semantic security of the private scheme due to the chain of key-encryptions in the public key of $\mathcal{H}^*$). Also, compactness follows trivially since the decryption algorithm is the same as that of the underlying compact scheme.

Similarly, d-hop circuit privacy follows easily from the 1-hop privacy of the underlying private scheme. The simulator for node j uses the underlying 1-hop simulator to generate the intermediate ciphertext c'_{j-1}, and then proceeds just as in the scheme to compute the description of $G_{c'_{j-1}}(\cdot)$ and compute c_j. ■

Remark 2. *To get a multi-hop scheme (without the parameter d), we can replace the chain of α_j's and β_j's by a two-circle $\alpha \leftarrow p\mathsf{Enc}(p\text{PK};\ c\text{SK})$, and $\beta \leftarrow c\mathsf{Enc}(c\text{PK};\ p\text{SK})$. If the result is still semantically secure and 1-hop circuit private, then we get a multi-hop compact and private scheme.*

4 Multi-hop homomorphic encryption from DDH

In this section, we devise a new tool for achieving multi-hop homomorphic encryption, namely "extendable and re-randomizable SFE", and show that this tool can be implemented under the decision Diffie-Hellman assumption. We begin with definitions.

Definition 7 (Extendable SFE – honest but curious)**.** *A two-flow SFE protocol* $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ *is (statistically) extendable, if there exists an efficient procedure* Extend *such that for every two circuits* f, f' *with the input length of* f' *equal the output length of* f*, every input* x *for* f*, and every first-flow message* m_1 *in the support of* $\mathsf{SFE1}(x)$*, the distributions* $\mathsf{Extend}(\mathsf{SFE2}(m_1, f),\ f')$ *and* $\mathsf{SFE2}(m_1,\ f' \circ f)$ *are statistically close.* Π *is computationally extendable if these two distributions are computationally indistinguishable (even given* m_1*,* f *and* f'*).*

Note that extendable SFE by itself already yields multi-hop homomorphic encryption with a weak form of circuit-privacy: to a recipient that *does not know the intermediate values* from $\mathsf{SFE2}(m_1, f)$, the output of Extend looks just as if it was generated "from scratch" by $\mathsf{SFE2}$, so Extend hides the circuit if $\mathsf{SFE2}$ does. This means that when Π is used for multi-hops, then as long as all the intermediate hops are "trusted" not to reveal their intermediate results, using Extend would maintain the privacy of everyone's circuits.

However, this solution still falls short of our circuit-privacy goal, since a collusion between the recipient and the node that computed $\mathsf{SFE2}(m_1, f)$ can reveal the circuit f'. In other words, the output of Extend may not be (pseudo)random given the intermediate results from $\mathsf{SFE2}(m_1, f)$. To overcome this problem, we introduce the notion of re-randomizable SFE: In a nutshell, we want to transform the second flow message m_2 into m_2' such that even a collusion of the party that computed m_2 with the recipient cannot distinguish m_2' from random. Then a node can re-randomize the circuit of its predecessor, thus rendering the intermediate results held by the predecessor irrelevant. This is formalized as follows:

Definition 8 (Re-randomizable SFE – honest but curious)**.** *A two-flow SFE protocol* Π *is (statistically) re-randomizable if there exists an efficient procedure* reRand *such that for every circuit* f *and input* x *and every two messages* m_1, m_2 *in the support of* $\mathsf{SFE1}(x)$*,* $\mathsf{SFE2}(m_1, f)$*, respectively, the distributions* $\mathsf{reRand}(m_1, m_2)$ *and* $\mathsf{SFE2}(m_1, f)$ *are statistically close.* Π *is computationally extendable if these two distributions are indistinguishable, even given* m_1, m_2 *(and the randomness that was used to generate them).*

Theorem 6 (Extendable+Re-randomizable $\Rightarrow$ Multi-hop)**.** *An extendable and re-randomizable two-flow SFE protocol with client and server privacy, in conjunction with a semantically secure public-key encryption scheme, implies a multi-hop homomorphic encryption scheme. Moreover, the size of an evaluated ciphertext in this scheme does not depend on the number of hops, but only on the composed circuit.*

Proof (sketch) Let $(\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ be an extendable and re-randomizable two flow SFE protocol with client and server privacy, and let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically secure public key encryption scheme. We now describe the construction of the homomorphic encryption scheme.

The key generation of the homomorphic scheme is the same as KeyGen for the underlying encryption. To encrypt a plaintext x under PK, first generate $(m_1, r) \leftarrow \mathsf{SFE1}(x)$, encrypt r under the public key PK to get $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(r)$, and finally, compute $m_2 \leftarrow \mathsf{SFE2}(m_1, f_{ID})$ (where f_{ID} is the identity function). The ciphertext is (c, m_1, m_2).

To evaluating a function f on a ciphertext (c, m_1, m_2), first set $m_2' \leftarrow \mathsf{reRand}(m_1, m_2)$, and then set $m_2^* \leftarrow \mathsf{Extend}(m_2', f)$. The evaluated ciphertext is (c, m_1, m_2^*).

Decrypting a ciphertext (c, m_1, m_2) consists of using the decryption of the underlying encryption scheme to recover the randomness r from c, then outputting $y \leftarrow \mathsf{SFE\text{-}Out}(r, m_2)$.

Correctness of the scheme follows from the the correctness of Π, and its extendability and re-randomizability: we know that $\mathsf{SFE\text{-}Out}$ would recover the right y when given the second-flow message from $\mathsf{SFE2}$, and by extendability the output of Extend is the same as that of $\mathsf{SFE2}$, no matter how many hops were used.[2] Semantic security follows from semantic security of the underlying encryption and from the client-privacy of Π.

Circuit privacy follows from the re-randomizability and extendability of the SFE protocol. Consider a player that gets the ciphertext $(c, m_1, m_2^{(j-1)})$, obtained after evaluating $f_1, \ldots, f_{j-1}$ on an encryption of x. This player begins by re-randomizing $m_2^{(j-1)}$, such that the distribution of the resulting ciphertext can be simulated using only the knowledge of the input $f_{j-1}(\ldots f_1(x))$. (This follows from the re-randomizability as well as the server privacy of the SFE). Finally, from the extendability of the SFE protocol, the final garbled circuit that player-j outputs is indistinguishable from a fresh garbled circuit for the combined function $f_j(\ldots f_1(\cdot))$ which, by the server privacy of the SFE protocol, can be simulated using only the knowledge of the output $f_j(\ldots f_1(x))$. In summary, the view of the honest-but-curious adversary can be sampled using the knowledge of only the input and the output of the j^{th} player (in the clear), which shows circuit privacy. ■

Given Theorem 6, we now focus on constructing an extendable and re-randomizable SFE protocol. Our starting point is Yao's garbled circuit construction [31, 19], which is extendable but not re-randomizable. We seek a re-randomizable implementation of this scheme by using building blocks that are "sufficiently homomorphic" to support the randomization that we need. Specifically, we rely on the oblivious-transfer protocol of Naor-Pinkas and Aiello-Ishai-Reingold [22, 2], and on the encryption scheme of Boneh-Halevi-Hamburg-Ostrovsky [7]. We recall our tools in Sections 4.1-4.3, and then present our construction in Sections 4.4-4.5.

4.1 Additively-homomorphic oblivious transfer

Definition 9 (Oblivious Transfer – honest but curious). *A two-flow oblivious transfer protocol is a two party protocol between a sender and a receiver, where the sender gets as input two bits $\gamma_0, \gamma_1 \in \{0,1\}$, the receiver gets as input a choice bit $\sigma \in \{0,1\}$, and the following conditions are satisfied:*

- Functionality: *For any sender input bits γ_0, γ_1 and choice bit σ, the receiver outputs γ_σ at the end of the protocol.*
- Receiver's security: *Denote by $OT1(1^k, \sigma)$ the message sent by the honest receiver with choice bit input σ (and security parameter k). Then the distribution $OT1(1^k, 0)$ and $OT1(1^k, 1)$ are indistinguishable.*
- Sender's security: *Denote by $OT2(1^k, \gamma_0, \gamma_1, m_1)$ the response of the honest sender with input (γ_0, γ_1) and security parameter k when the receiver's first message is m_1. Then there exists an efficient simulator* Sim *such that for any three bits $\sigma, \gamma_0, \gamma_1 \in \{0,1\}$, and any*

[2] For computationally extendable Π, we use the fact that outputting the wrong y would be a distinguishing test.

first-flow message m_1 in the support of $OT1(1^k, \sigma)$, the distributions $OT2(1^k, \gamma_0, \gamma_1, m_1)$ and $\mathsf{Sim}(1^k, b, m_1, \gamma_\sigma)$ are statistically close.

Definition 10 (Blindable OT). *A two-flow oblivious transfer protocol is blindable if there exists an efficient algorithm* Blind *such that for every three bits $\sigma, \gamma_0, \gamma_1$, every first-flow message m_1 in the support of $OT1(1^k, \sigma)$ and every second-flow message m_2 in the support of $OT2(1^k, \gamma_0, \gamma_1, m_1)$, the distributions $\mathsf{Blind}(m_1, m_2)$ and $OT2(1^k, \gamma_0, \gamma_1, m_1)$ are statistically close.*

Naor-Pinkas and Aiello et al. [22, 2] proved that the following protocol meets Definition 9.[3] The protocol operates in a prime-order group where the decision Diffie-Hellman problem is believed hard. Denote the group order by q. On input a choice-bit σ, the receiver chooses two arbitrary distinct order-q elements $g, h \in G$ and two random distinct exponents $r, r' \in_R Z_q^*$. The receiver computes $x := g^r$, $y_\sigma := h^r$, and $y_{\bar{\sigma}} := h^{r'}$, and sends to the sender the elements (g, h, x, y_0, y_1). Note that (g, h, x, y_σ) is a Diffie-Hellman tuple, while $(g, h, x, y_{\bar{\sigma}})$ is a non-Diffie-Hellman tuple.

The sender, given two input bits γ_0, γ_1 and the receiver's message (g, h, x, y_0, y_1), chooses four random exponents $s_0, t_0, s_1, t_1 \in_R Z_q$, and for $i \in \{0, 1\}$ it sets $a_i := g^{s_i} h^{t_i}$ and $b_i := x^{s_i} y_i^{t_i} \cdot g^{\gamma_i}$. The sender sends (a_0, b_0, a_1, b_1) back to the receiver. When the sender inputs are longer than one bit, the same construction can be repeated for every bit (but they all can share the same elements g, h, x, y_0, y_1).

The receiver can recover the bit γ_b by outputting zero when $b_\sigma = a_\sigma^r$ and outputting one otherwise. At the same time, the bit $\gamma_{\bar{\sigma}}$ is statistically hidden from the receiver, since $(g, h, x, y_{\bar{\sigma}})$ is a non-Diffie-Hellman tuple. (This scheme was later extended to work with other smooth projective hashing schemes, cf. [17, 15].)

This scheme is also blindable: On input $m_1 = (g, h, x, y_0, y_1)$ and $m_2 = (a_0, b_0, a_1, b_1)$, the Blind algorithm chooses four random exponents $s'_0, t'_0, s'_1, t'_1 \in_R Z_q$, and for $i \in \{0, 1\}$ it sets: $a'_i := g^{s'_i} h^{t'_i} \cdot a_i$ $(= g^{s'_i + s_i} h^{t'_i + t_i})$, and $b'_i := x^{s'_i} y_i^{t'_i} \cdot b_i$ $(= x^{s'_i + s_i} y_i^{t'_i + t_i} \cdot g^{\gamma_i})$.

4.2 The BHHO encryption scheme

Boneh et al. described in [7] a "circular secure" encryption scheme, with security based on the hardness of DDH. Below we refer to this scheme as the BHHO scheme. The BHHO scheme is a public-key encryption scheme, but here we describe it as a secret-key scheme (since we only use the public key for re-randomization, not for encryption). The scheme works in a prime-order group G where the Decision Diffie-Hellman problem is believed hard. Denote the order of G by q, let g be some "canonical" generator of G, and denote $\ell \overset{\text{def}}{=} \lceil 3 \log q \rceil$.

The secret key is a random vector $\vec{s} \in \{0, 1\}^\ell$. An encryption of a bit $b \in \{0, 1\}$ is an $(\ell+1)$-vector of elements $\vec{u} \in G^{\ell+1}$, with the first ℓ elements chosen at random in G and the last one computed as $u_{\ell+1} := g^b / \prod_{i=1}^{\ell} u_i^{s_i}$. Decryption works by computing $y := u_{\ell+1} \cdot \prod_{i=1}^{\ell} u_i^{s_i} = 1$, outputting zero if $y = 1$, one if $y = g$, and $\perp$ otherwise. The public key for this scheme is a random encryptions of zero, and here we consider the public key to be a part of every ciphertext. Encrypting a vector of bits is done bit-by-bit.

It was shown in [7] that this scheme is semantically secure, and it also enjoys strong homomorphic properties for both plaintext and secret-key. In particular, given a BHHO public key PK for some secret key $\vec{s} \in \{0, 1\}^\ell$ and a ciphertext $\vec{u} \in G^{\ell+1}$ that encrypts a bit b w.r.t. $\vec{s}$, and given any affine transformation from Z_q^ℓ to itself, $T(\vec{x}) = A\vec{x} + \vec{b}$, one can transform PK, $\vec{u}$ into PK$'$, $\vec{u}'$ such that

[3] In fact, they proved that this protocol is even secure in the malicious model.

IF $\vec{s'} = T(\vec{s})$ is a 0-1 vector, THEN PK$'$ is a random public key for $\vec{s'}$ and $\vec{u'}$ is a random encryption of the same bit b under $\vec{s'}$. [4] This means in particular that we can implement a bitwise XOR of a known mask with $\vec{s}$, and a permutation of the bits of $\vec{s}$, since both are affine functions that map 0-1 vectors to 0-1 vectors. Also, BHHO has the same homomorphic properties with respect to the plaintext.

4.3 Yao's garbled circuit construction

Recall that in Yao's garbled circuit construction, the server (who has a circuit of fan-in-2 gates) chooses two random ℓ-bit labels for every wire w in the circuit: label $L_{w,0}$ representing the value zero and label $L_{w,1}$ representing the value one. For every gate in the circuit, the server prepares a small gadget as follows: Denote the gate operation by $\star$ and let the labels on the input wires be $L_{w_1,0}$, $L_{w_1,1}$, $L_{w_2,0}$, $L_{w_2,1}$, and the labels on the output wires be $L_{w_3,0}$, $L_{w_3,1}$. The gate gadget consists of the four ciphertexts

$$\{\mathsf{Enc}_{L_{w_1,i}}(\mathsf{Enc}_{L_{w_2,j}}(L_{w_3,k}))\ :\ i,j \in \{0,1\},\ k = i \star j\} \tag{1}$$

The server sends all these gadgets to the client, and also uses an oblivious transfer protocol to reveal exactly one of the two labels on every input wire to the client, depending on the clients input. The server also provides the mapping of the two labels on every output wire to zero and one. The client can then decrypt its way through the circuit, arriving at the output labels and learning the output bits.

We comment that this construction does not hide the structure of the circuit (i.e., what gate output is used in what other gate input), but this is easily dealt with by canonicalizing the circuit. Recall that we consider a "universal circuit evaluator" $U(\cdot,\cdot)$ with $f(x)$ being a shorthand for $U(f,x)$. The server can therefore always prepare a garbled version of U itself, and for the bits of f it just sends the corresponding label to the client in the clear.

Clearly, this construction is extendable: an intermediate node can take the output labels from its predecessor and just use them as input labels for its own garbled circuit.[5] As we mentioned above, this means that it offers a weak form of circuit-privacy. However, it is not quite circuit private, since the predecessor of a node knows both input labels to the circuit of this node, so from the predecessor's perspective the circuit of this node is not garbled at all.

To allow re-randomization, we rely on the homomorphic properties of our building blocks. In a nutshell, we implement the double-encryption from Eq. (1) by choosing a random mask δ and then encrypting δ under $L_{w_1,i}$ and encrypting $(L_{w_3,k} \oplus \delta)$ under $L_{w_2,j}$, using the BHHO scheme. We note that re-randomization turns out to be nontrivial. In particular XOR-homomorphism seems insufficient, and we need to use the full power of the "affine function homomorphism" of BHHO. See Section 4.5.

[4] Strictly speaking, to get new public key and ciphertext that are random and independent of the original PK and $\vec{u}$, one needs to use the "extended public key" for the scheme $\mathcal{E}_1$ from [7]. It is easy to see, however, that using the non-extended public key we get a new public key and ciphertext that are pseudorandom under DDH. We ignore this fine point in the rest of this writeup.

[5] This description assumes that the "circuit canonicalization" still works, i.e., that canonicalizing first f and then f' gives the same circuit as canonicalizing $f' \circ f$.

4.4 Our Construction

The construction closely follows Yao's original garbled circuit construction. The client (Alice) on input $x_1, \ldots, x_n$, sends n first-flow messages of the OT protocol from above, using her input bit x_i as the choice bit for the i'th message, $m_1[i] \leftarrow OT1(x_i)$.

The server (Bob) has a boolean circuit with fan-in-2 gates. Bob's circuit has n input ports, some number of output ports, and some number of internal gates. Each wire in the circuit is therefore either an input wire (connecting an input port to some internal gates and/or output ports), or a gate-output wire (connecting the output of one internal gate to some other internal gates and/or output ports). We stress that we allow the same wire to be used as input to several internal gates or output ports.[6]

Bob receives from Alice the n OT first-flow messages, $m_1[1], \ldots, m_1[n]$. He begins by choosing at random two ℓ-bit labels $L_{w,0}, L_{w,1}$ for every wire w, each having exactly $\lceil \ell/2 \rceil$ 1's. (Here ℓ is the length of the BHHO secret key.) For each input wire w_i, corresponding to Alice's first-flow message $m_1[i]$, Bob computes the second-flow OT message for the two labels on the corresponding input wire, $m_2[i] \leftarrow OT2(m_1[i]; L_{w_i,0}, L_{w_i,1})$.

Then, for an internal fan-in-2 gate (computing the binary operation $\star$), Bob computes four pairs of ciphertexts as follows: Let w_1, w_2 be the two input wires for this gate and w_3 be the output wire. Bob chooses four fresh random 2ℓ-bit masks $\delta_{i,j}$ for $i, j \in \{0, 1\}$ and computes the four pairs:

$$\Big\{ \Big(\mathsf{Enc}_{L_{w_1,i}}(\delta_{i,j}),\ \mathsf{Enc}_{L_{w_2,j}}((L_{w_3,k}|0^\ell) \oplus \delta_{i,j}) \Big) \ :\ i, j \in \{0,1\},\ k = i \star j \Big\} \tag{2}$$

Namely, Bob uses the secret key $L_{w_1,i}$ to encrypt the mask $\delta_{i,j}$ itself, and the other secret key $L_{w_2,j}$ to encrypt the masked label, concatenated with ℓ zeros).[7] The "gadget" for this gate consists of the four pairs of ciphertexts from Eq. (2) in random order. The gabled circuit that Bob sends back to Alice consists of the n second-flow OT messages $m_2[1], \ldots, m_2[n]$, and the gadgets for all the gates in the circuit (which we assume include an indication of which wire connects what gates). In addition, for each output wire w with labels $L_{w,0}$ and $L_{w,1}$, Bob sends an ordered pair of public keys, the first corresponding to $L_{w,0}$ and the second to $L_{w,1}$. (We chose this particular mapping to enable re-randomization.)

Upon receiving this garbled circuit, Alice first uses the recovery procedure of the OT protocol to recover one of the labels for each input wire. Then she goes over the garbled circuit gate by gate as follows: For a fan-in-2 gate where she knows the labels L_1, L_2 for the two inputs, she uses the key L_1 to decrypt the first component in each of the four pairs and uses the key L_2 to decrypt the second component of the four pairs. Then she XORs the two decrypted strings from each pair, and if any of the resulting strings is of the form $L^*|0^\ell$ then she takes L^* to be the label of the output wire. (If more than one string has the form $L^*|0$ then Alice takes the first one, and if none has this form then she sets $L^* = 0^\ell$.) Upon recovering a label on an output port, she checks if this label corresponds to the first or the second public keys that were provided for this port, outputting zero or one accordingly. (Or $\perp$ if it does not correspond to any of them.)

[6]We assume that the two input wires at each gate are always distinct. This can be enforced, e.g., by implementing a fan-in-1 gate (i.e., NOT) via a fan-in-2 XOR-with-one gate.

[7]Recall that encryption under a secret key $\vec{s}$ includes also the BHHO-public-key for this secret key. In our case we encrypt many bits under the same secret key, and we can use just one public key for all of these encryptions to save on space. This does not really effect security, however.

Remark: balanced secret keys. We note that the BHHO scheme is used here with secret keys that have exactly $\ell/2$ 1's in them, rather than with completely uniform secret keys. This is used for the purpose of re-randomization, as described in Section 4.5. We note that this variant of BHHO is also semantically secure: In fact, Naor and Segev proved that under DDH, the BHHO scheme is semantically-secure for *every secret-key distribution with sufficient min-entropy* (cf. [23, Sec 5.2]). We will use this stronger result in our proof of the re-randomization property in Section 4.5.

Theorem 7. *The protocol from above, using the BHHO encryption scheme, enjoys both client and server privacy, under the DDH assumption.*

Proof (sketch) The proof is essentially the same as the Lindell-Pinkas proof of the Yao protocol [19, Thm 5]. We note that privacy holds in whatever model in which the underlying OT protocol is secure. That is, if the OT protocol is only secure in the honest-but-curious model then the above protocol will be private only in the honest-but-curious model, and if the OT is secure in the malicious model then this protocol will also be private in the malicious model.

The client-privacy part is completely identical to [19], and is omitted here. The high-level structure of the server-privacy proof is also similar to [19], in that we use roughly the same simulator, and a similar high-level argument about why the simulator's output is indistinguishable from the real scheme. Given Alice's first-flow message $(m_1[1], \ldots, m_1[n])$ and the value $f(x)$ (for Bob's function f and Alice's effective input x), the simulator proceeds as follows: First it chooses two random labels, each with $\ell/2$ 1's, for each wire in the circuit. Then it chooses at random one of these two labels, and designates it as the "active label" for that wire. Throughout this proof we always denote the active label on wire w by L_w, and the other label by L'_w.

Next, the simulator uses the OT simulator to generate second-flow OT messages that would yield the active value for each input wire, setting $m_2[i] \leftarrow \text{OT-Sim}(x_i, m_1[i], L_{w_i})$.

Next, for each internal fan-in-2 gate in the circuit with input wires w_1, w_2 and output wire w_3, the simulator generates four ciphertext-pairs under the same keys as Bob would have done, but it encrypts only the active label for the output wire in these four pairs. Namely, denote the active labels on these wires by $L_{w_1}, L_{w_2}, L_{w_3}$, and the inactive labels by $L'_{w_1}, L'_{w_2}, L'_{w_3}$, respectively. Bob chooses four fresh random masks $\delta_1, \delta_2, \delta_3, \delta_4$ for this gate, and computes the four ciphertext pairs:

$$\begin{array}{ccc} \Big(\mathsf{Enc}_{L_{w_1}}(\delta_1),\ \mathsf{Enc}_{L_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_1)\Big) & , & \Big(\mathsf{Enc}_{L_{w_1}}(\delta_2),\ \mathsf{Enc}_{L'_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_2)\Big) \\ \Big(\mathsf{Enc}_{L'_{w_1}}(\delta_3),\ \mathsf{Enc}_{L_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_3)\Big) & , & \Big(\mathsf{Enc}_{L'_{w_1}}(\delta_4),\ \mathsf{Enc}_{L'_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_4)\Big) \end{array} \tag{3}$$

The simulated gadget for this circuit consists of the four pairs in random order. Finally, for each output port the simulator provide an ordered pair of the public keys for both labels, where the public key of the active label is either the first or the second in the pair, depending on whether the output bit for that port is zero or one.

Proving that the view generated by this simulator is indistinguishable from the real execution follows an approach similar to Lindell-Pinkas. We consider a sequence of games, with the first game producing a distribution identical to Bob's message and the last game producing a distribution identical to simulator's output, and prove that any two successive games have indistinguishable output. These games are all played by a "challenger" that knows Bob's function f and Alice's effective input x.

The first game just follows Bob's procedure for generating his reply, without any changes. In addition, for every wire w in the circuit, the challenger designates the label that the honest Alice (with input x) would learn during evaluation as the "active" label, and the other label is the "inactive" one.

The second game proceeds just as in the first game, except that the OT second-flow messages are generated by the OT simulator instead of the OT protocol. This game is indistinguishable from the first game by the sender-security of the OT protocol.

Next come a sequence of games, one for each wire in the circuit. In each game, some ciphertexts are modified from encrypting the "correct label" for a gate-output wire (as Bob does) to encrypting just the active label for the same wire (as the simulator does). Specifically, the wires are ordered in the order that Alice learns their labels during evaluation. Then, in the i'th game we change all the encryptions under the inactive label of the i'th wire. That is, for every gate that use wire w_i for input, two of the four ciphertext-pairs include a ciphertext that is encrypted under the inactive label L'_{w_i}. In the i'th game, we change the value that is encrypted in these ciphertexts, so that when XORed with the value in the other ciphertext in the pair, they result in the corresponding gate-output wire (with ℓ trailing 0's). (This may or may not be the same value that was encrypted in these ciphertexts in game $i-1$.)

We note that the only ciphertext-pairs that are not modified in this sequence of games are those where both encryptions are under the active labels of the input wires. By definition, this means that the label encrypted by this pair must also be active: If Alice knows both labels L_{w_i} and L_{w_j} then she learns also the label that is encrypted by the pair $(\mathsf{Enc}_{L_{w_i}}(\cdot), \mathsf{Enc}_{L_{w_j}}(\cdot))$, hence that label is active. It follows that at the last game in the sequence, only the active labels are encrypted everywhere. Hence that last game produces a distribution identical to simulator's output.

Proving that each game is indistinguishable from the next is done by reduction to the semantic security of the BHHO scheme. Assume that (for particular x, f) we have a distinguisher $\mathcal{D}$ with advantage ϵ between games $i-1$ and i, and we show a (nonuniform) CPA attacker $\mathcal{A}$ against BHHO with the same advantage.

The attacker $\mathcal{A}$ gets x, f and i as its nonuniform advice. Then it gets a BHHO public key, corresponding to some unknown secret key that we denote by $\vec{s} \in \{0,1\}^\ell$. The attacker $\mathcal{A}$ now needs to produce the two target messages of the CPA game. $\mathcal{A}$ runs the challenger, producing all the values as in the game $i-1$. Then it replaces the the ciphertexts that were encrypted under the inactive label of the i'th wire w_i, as described next.

Intuitively, the attacker re-generates these ciphertexts by implicitly setting the inactive label of the i'th wire to be $\vec{s}$, the unknown key corresponding to its input public key. The attacker uses its CPA target ciphertext to get ciphertexts under this unknown key, hence getting either encryptions as in game $i-1$ or encryptions of the active labels, depending on which of the two target messages was encrypted in the challenge ciphertext.

In more details, denote by L_{w_i}, L'_{w_i} the active and inactive labels on the i'th wire w_i, respectively. Also consider all the gates that uses wire w_i for input (say that there are m of them), denote the other input wires for these gates by $v_1, \ldots, v_m$ (these need not be distinct, but they are different than w_i) and the output wires of these gates by $u_1, \ldots, u_m$ (these are distinct). For a gate-input wire v_j, denote the active and inactive label on that wire by L_{v_j}, L'_{v_j}, respectively, and similarly L_{u_j}, L'_{u_j} are the active and inactive labels for the gate-output wire u_j. We assume for concreteness that w_i is the second input wire for this gate (the case where it is the first input wire is symmetric).

The four ciphertext pairs for this gate in game $i-1$ were computed as

$$\begin{array}{cc}\Big(\mathsf{Enc}_{L_{v_j}}(\delta_{j,1}),\ \mathsf{Enc}_{L_{w_i}}((L_{u_j}|0^\ell)\oplus\delta_{j,1})\Big) \quad , & \Big(\mathsf{Enc}_{L_{v_j}}(\delta_{j,2}),\ \mathsf{Enc}_{L'_{w_i}}((X_j|0^\ell)\oplus\delta_{j,2})\Big) \\ \Big(\mathsf{Enc}_{L'_{v_j}}(\delta_{j,3}),\ \mathsf{Enc}_{L_{w_i}}((Y_j|0^\ell)\oplus\delta_{j,3})\Big) \quad , & \Big(\mathsf{Enc}_{L'_{v_j}}(\delta_{j4}),\ \mathsf{Enc}_{L'_{w_i}}((Z_j|0^\ell)\oplus\delta_{j,4})\Big)\end{array} \tag{4}$$

In Eq. (4), the $\delta_{j,*}$'s are the fresh masks that were chosen for the j'th gate, and each of X_j, Y_j, Z_j is either L_{u_j} or L'_{u_j}.

We denote $\delta'_{j,2} \overset{\text{def}}{=} \delta_{j,2} \oplus ((L_{u_j} \oplus X_j)|0^\ell)$ and $\delta'_{j,4} \overset{\text{def}}{=} \delta_{j,4} \oplus ((L_{u_j} \oplus Z_j)|0^\ell)$ (i.e., the string that should be encrypted under L'_{w_i} to get the 2'nd and 4'th pairs to be decrypted as $L_{u_j}|0^\ell$). The attacker $\mathcal{A}$ sets the target messages for the CPA game as:

$$M_0 = (\delta_{1,2}|\delta_{1,4} \mid \cdots \mid \delta_{m,2}|\delta_{m,4}) \quad \text{and} \quad M_1 = (\delta'_{1,2}|\delta'_{1,4} \mid \cdots \mid \delta'_{m,2}|\delta'_{m,4})$$

(If w_i is the first input wire to the j'th gate then we use $\delta_{j,3}$, $\delta'_{j,3}$ instead of $\delta_{j,2}$, $\delta'_{j,2}$ above.) By construction, M_0 includes all the strings that were encrypted under L'_{w_i} in game $i-1$, while M_1 contains all the strings that were encrypted under the same key in game i.

Upon receipt of the CPA challenge ciphertext c^* (which was computed with respect to the unknown secret key $\vec{s}$), $\mathcal{A}$ extracts for each wire u_j the portion of c^* corresponding to the δ_j's (or δ'_j's), and use them in the gadget for j'th gate. Finally $\mathcal{A}$ sends the garbled circuit to the distinguisher $\mathcal{D}$ and outputs whatever $\mathcal{D}$ does.

By construction, the output of $\mathcal{A}$ is consistent with a run of the challenger in which the inactive label on the wire w_i is chosen as the random unknown secret key $\vec{s}$. Depending on what's encrypted in the challenge ciphertext, the values encrypted under this key are either the values that were encrypted in game $i-1$, or these from game i. Hence the advantage of $\mathcal{A}$ in the CPA game equals the advantage of $\mathcal{D}$ in distinguishing game i from game $i-1$. ■

Theorem 8. *The BHHO-based protocol from above is extendable.*

Proof (sketch) Recall that for each output wire, the garbled circuit includes an ordered pair of the two public keys corresponding to the labels for this wire, with the zero-label public-key first and the one-label public-key second. To extend this circuit, we need to use the output labels of the given garbled circuit as input labels for the higher-level circuit. We note, however, that in our construction we only use input labels as keys that encrypt some values. This use does not require that we know the actual label, we can generate random ciphertexts under some label just by knowing the corresponding public key. ■

4.5 Re-randomizing garbled circuits

We next proceed to show how garbled circuits from above can be re-randomized. We begin by observing that a simple re-randomization method that only XORs random masks into the labels does not work: Although one can transform a ciphertext $\mathsf{Enc}_L(L')$ into $\mathsf{Enc}_{L\oplus\Delta}(L'\oplus\Delta')$ for any Δ, Δ', the re-randomizer cannot choose enough random masks to effectively hide the circuit. To see why, observe that the re-randomizer does not know which of the two labels on a wire was used as key (or input) in what ciphertext, so it cannot use two different masks to randomize the two different labels on a wire. Rather, it can only apply the *same mask* Δ_w to both labels on a wire. But this is clearly not sufficient for randomization, since it leaves the XOR of the two labels on each wire as it was before.

Moreover, such "partial randomization" is clearly insecure in our application: Note that the predecessor of a node knows the two "old labels" for every wire in its circuit, including the labels for the output wires (which are the current node's input wires). Also, the receiver (Alice) would learn one of the "new labels" on these wire upon evaluation. Hence between the predecessor and Alice, they will be able to reconstruct both new labels for every input, thus un-garbling the circuit of the current node.

To overcome this problem, we rely on stronger homomorphic properties of BHHO: Namely, viewing keys and plaintexts as vectors, it is homomorphic with respect to any affine function over Z_q. This means, in particular, that it is homomorphic with respect to permutations (i.e., multiplications by permutation matrices). Namely, we can transform a ciphertext $\mathsf{Enc}_L(L')$ into $\mathsf{Enc}_{\pi(L)}(\pi'(L'))$ for any two permutations π, π' of the bits. We therefore work with balanced secret keys that have exactly $\ell/2$ 1's, and use permutations to randomize them.

Note that in the attack scenario from above, where a predecessor colludes with the recipient, they will now know the old labels L, L', and also one new label, computed as $\pi(L)$. In Lemma 10 we show that given these three values, the other new label $\pi(L')$ still has a lot of min-entropy, provided that the Hamming distance between L, L' is not too small. In the honest-but-curious model, L and L' will be about $\ell/2$ apart, hence $\pi(L')$ will have min-entropy close to ℓ. [8] The Naor-Segev result [23] then implies that it is safe to use $\pi(L')$ as a secret key, which is indeed the way that it is used in the re-garbled circuit. Putting all these arguments together, we have the following theorem:

Theorem 9. *The BHHO-based protocol from above is computationally re-randomizable, under DDH.*

Proof (sketch) We describe the re-randomization algorithm. Given a garbled circuit, the re-randomizer chooses a permutation π_w (over $[1, \ell]$) for every wire w in the circuit, and applies that permutation to both labels on this wire. For the OT portion, since we are using a bit-by-bit OT protocol then the re-randomizer just permutes the OT responses and then blinds them to hide the permutation. Namely, the second-flow OT message for each of Alice's input bits is a vector of ℓ OT responses (one for each bit in the labels of the wire w_i). The re-randomizer permutes these ℓ OT responses according to π_{w_i} and then blinds them all.

For the garbled circuit portion, consider one particular gate in the circuit, which is represented by four ciphertexts as in Eq. (2)

$$\left\{ \left(\mathsf{Enc}_{L_{w_1,i}}(\delta_{i,j}),\ \mathsf{Enc}_{L_{w_2,j}}((L_{w_3,k}|0^\ell) \oplus \delta_{i,j}) \right) \ :\ i,j \in \{0,1\},\ k = i \star j \right\}$$

Of course, the re-randomizer only sees the ciphertexts, not the labels that were used to generate them. Still, using the BHHO homomorphic properties and the permutations $\pi_{w_1}, \pi_{w_2}, \pi_{w_3}$ that is chose, it can transform these ciphertexts first into

$$\left\{ \left(\mathsf{Enc}_{\pi_{w_1}(L_{w_1,i})}(\tilde{\pi}_{w_3}(\delta_{i,j})),\ \mathsf{Enc}_{\pi_{w_2}(L_{w_2,j})}(\tilde{\pi}_{w_3}((L_{w_3,k}|0^\ell) \oplus \delta_{i,j}) \right) \ :\ i,j \in \{0,1\},\ k = i \star j \right\}$$

[8]One can view Lemma 10 as saying that a random bit permutation gives a weak notion of universal hashing: although it is not true that $\pi(L')$ has high entropy given $\pi(L)$ for every $L \neq L'$, it does hold w.h.p. when x, x' are chosen at random.

where by $\tilde{\pi}_{w_3}(\cdot)$ above we mean applying π_{w_3} to the first ℓ bits of the 2ℓ-bit argument, leaving the last bits unchanged. Then the re-randomizer chooses one more random mask for every pair and XORs it into the values encrypted in both ciphertexts. The result is four pairs of ciphertexts, each of them a random encryption under the permuted key, such that each pair encrypts the corresponding permuted output label.

Similarly for an output wire w, the re-randomizer uses the homomorphism of BHHO to transform the pair of public keys for $L_{w,0}, L_{w,1}$ into a pair of public-keys with respect to $\pi_w(L_{w,0}), \pi_w(L_{w,1})$.

The proof that this procedure achieves computational re-randomization is nearly identical to the proof that it achieves server privacy. Namely, we show that even given the original garbled circuit and all the randomness that was used to generate it, the re-randomized circuit is still indistinguishable from the output of the simulator from Theorem 7. Computational re-randomization follows since we already proved that the simulator's output is indistinguishable from a fresh random garbled circuit (and the extra quantities that the distinguisher knows are no longer used anywhere, so indistinguishability in not effected).

The only difference between this proof and the one from Theorem 7 is in the reduction to semantic-security of BHHO, when moving from game $i-1$ to game i. In the case of re-randomization, the distinguisher $\mathcal{D}$ also knows for each wire w the two "old labels" that were used previously on this wire. That is, if the current labels on this wire are L_w and L'_w, then the distinguisher knows also $\pi_w^{-1}(L_w)$ and $\pi_w^{-1}(L'_w)$. In the reduction, therefore, the attacker $\mathcal{A}$ (who wants to implicitly define $L'_w = \vec{s}$ for the unknown secret key $\vec{s}$) must be able to supply these quantities to the distinguisher $\mathcal{D}$.

Here we appeal to the Naor-Segev result about the leakage resilience of BHHO [23]. We define a randomized leakage function that given a secret key $\vec{s}$ (with $\ell/2$ 1's), chooses at random another balanced string L_w and a bit permutation π, and returns to the adversary $\pi^{-1}(L_w), \pi^{-1}(\vec{s})$, and L_w. Lemma 10 says that this leakage function leaks only $O(\log \ell)$ bits of entropy about $\vec{s}$, and the result of Naor-Segev says that BHHO is still semantically-secure with respect to such leakage functions. ■

The permutation lemma. Let $HW_{\ell,k} \subseteq \{0,1\}^\ell$ denote the set of all ℓ-bit strings with Hamming weight exactly k, and also let S_ℓ denote the set of all permutations over ℓ elements. Assume that ℓ is even from now on. The lemma below shows that for two strings L_1 and L_2, chosen uniformly at random from $HW_{\ell,\ell/2}$, and a random permutation $\pi : [\ell] \to [\ell]$, the string $\pi(L_2)$ has large residual min-entropy *even given* L_1, L_2 *and* $\pi(L_1)$. For the lemma below, let $\widetilde{H}_\infty(X|Y)$ be the average min-entropy of X given Y (cf. [9]), that is

$$\widetilde{H}_\infty(X|Y) \overset{\text{def}}{=} -\log \mathop{\mathbb{E}}_{y \leftarrow Y}\left(\max_x \Pr[X = x|Y = y]\right) = -\log \mathop{\mathbb{E}}_{y \leftarrow Y}\left(2^{-H_\infty(X|Y=y)}\right)$$

Lemma 10. *Let $L_1, L_2 \in_R HW_{\ell,\ell/2}$, and $\pi \in_R S_\ell$ be uniformly random. Then:*

$$\widetilde{H}_\infty\big(\pi(L_2) \mid L_1, L_2, \pi(L_1)\big) \geq \ell - \frac{3}{2}\log \ell$$

The proof is in Appendix A. It follows easily from the observation that given L_1, L_2 and $\pi(L_1)$, the string $\pi(L_2)$ is distributed uniformly from among all strings in $HW_{\ell,\ell/2}$ whose Hamming distance from $\pi(L_1)$ equals the Hamming distance between L_1 and L_2. ■

References

[1] C. Aguilar Melchor, G. Castagnos, and G. Gaborit. Lattice-based homomorphic encryption of vector spaces. In *IEEE International Symposium on Information Theory, ISIT'2008*, pages 1858–1862, 2008.

[2] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001.

[3] F. Armknecht and A.-R. Sadeghi. A new approach for algebraically homomorphic encryption. Cryptology ePrint Archive, Report 2008/422, 2008. `http://eprint.iacr.org/`.

[4] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. Cryptology ePrint Archive, Report 2009/511, 2009. `http://eprint.iacr.org/`.

[5] Josh Benaloh. *Verifiable secret-ballot elections.* PhD thesis, Yale University, 1988.

[6] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography - TCC'05*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[7] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.

[8] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. pages 119–136, 2001.

[9] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540, 2004.

[10] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. pages 10–18, 1985.

[11] M. Fellows and N. Koblitz. Combinatorial cryptosystems galore! *Contemporary Mathematics*, 168(2):51–61, 1993.

[12] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pages 169–178. ACM, 2009.

[13] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[14] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[15] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. Cryptology ePrint Archive, Report 2007/118, 2007. `http://eprint.iacr.org/`.

[16] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.

[17] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In *EUROCRYPT*, pages 78–95, 2005.

[18] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Multi-bit cryptosystems based on lattice problems. pages 315–329, 2007.

[19] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2), 2009.

[20] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additive Homomorphic Encryption with t-Operand Multiplications. Technical Report 2008/378, IACR ePrint archive, 2008.

[21] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. pages 59–66, 1998.

[22] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[23] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[24] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. pages 308–318, 1998.

[25] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. pages 223–238, 1999.

[26] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. pages 187–196, 2008.

[27] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[28] T. Sander, A. Young, and M. Yung. Non-interactive CryptoComputing for NC1. In *40th Annual Symposium on Foundations of Computer Science*, pages 554–567. IEEE, 1999.

[29] N.P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. Cryptology ePrint Archive, Report 2009/571, 2009. `http://eprint.iacr.org/`.

[30] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009. `http://eprint.iacr.org/`.

[31] Andrew C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science – FOCS '82*, pages 160–164. IEEE, 1982.

[32] Andrew C. Yao. Theory and application of trapdoor functions. *Symposium on Foundations of Computer Science*, 0:80–91, 1982.

A Proof of Lemma 10

We show that for *any two fixed strings* $x, y \in HW_{\ell,\ell/2}$ whose Hamming distance is d (d must be even), the residual min-entropy

$$\widetilde{H}_\infty(\pi(y) \mid x, y, \pi(x)) = 2\log\binom{n/2}{d/2} \tag{5}$$

where $\pi \leftarrow S_\ell$ is uniformly random. This immediately implies Lemma 10, since

$$\begin{aligned}
\widetilde{H}_\infty(\pi(L_2) \mid L_1, L_2, \pi(L_1)) &= -\log \mathop{\mathbb{E}}_{x,y\leftarrow HW_{n,n/2}}\left(2^{-\widetilde{H}_\infty(\pi(y)\mid x,y,\pi(x))}\right) && \text{(by definition of } \widetilde{H}_\infty\text{)}\\
&= -\log\left(\sum_{\text{even } d}\Pr[HD(\ell_1,\ell_2)=d]\cdot\frac{1}{\binom{\ell/2}{d/2}^2}\right) && \text{(by Equation 5)}\\
&= -\log\left(\frac{1}{\binom{\ell}{\ell/2}}\cdot\sum_{\text{even } d}\binom{\ell/2}{d/2}^2\cdot\frac{1}{\binom{\ell/2}{d/2}^2}\right) && \text{(by prob. calculation)}\\
&= \log\left(\binom{\ell}{\ell/2}/(\tfrac{\ell}{2}+1)\right) \;\geq\; \log\left(2^{\ell-1}/(\tfrac{\ell}{2})^{3/2}\right) \;\geq\; \ell - \tfrac{3}{2}\log\ell
\end{aligned}$$

■

It remains to prove Equation 5. Fix $x, x' \in HW_{\ell,\ell/2}$, and define $S_{x,x'} \overset{\text{def}}{=} \{\pi : \pi(x) = x'\}$. It is not hard to see that $|S_{x,x'}| = ((\ell/2)!)^2$ for every $x, x' \in HW_{\ell,\ell/2}$: Let I_0, I_1 a partition of the bit positions $[\ell]$ according to whether $x_i = 0$ or $x_i = 1$, and similarly let I'_0, I'_1 be such a partition for x'. (Note that $|I_0| = |I_1| = |I'_0| = |I'_1| = \ell/2$.)

$$\begin{aligned}
I_0 &= \{i \in [\ell] : x_i = 0\} \quad , \quad I'_0 = \{i \in [\ell] : x'_i = 0\}\\
I_1 &= \{i \in [\ell] : x_i = 1\} \quad , \quad I'_1 = \{i \in [\ell] : x'_i = 1\}
\end{aligned}$$

Also let δ be a fixed "canonical" permutation mapping I_0 to I'_0 and I_1 to I'_1. Then every permutation mapping x to x' is a product $\pi = \rho_{I_0} \circ \rho_{I_1} \circ \delta$, with ρ_{I_0} a permutation only on the indexes in I_0 and ρ_{I_1} a permutation only on the indexes in I_1. Moreover the mapping $(\rho_{I_0}, \rho_{I_1}) \Leftrightarrow \pi$ is a bijection between $S_{x,x'}$ and $S_{\ell/2} \times S_{\ell/2}$.

Similarly, fix four strings $x, y, x'y' \in HW_{\ell,\ell/2}$, and define $T_{x,y,x',y'} \overset{\text{def}}{=} \{\pi : \pi(x) = x',\ \pi(x) = y'\}$. Let us also denote by d the Hamming distance between x, y. A similar argument to above shows that the size of $T_{x,y,x',y'}$ is either zero (if the Hamming distance between x', y' is anything other than d), or else it is exactly $((d/2)!(\ell/2-d/2)!)^2$. (In this case we partition $[\ell]$ to four sets, depending on the values of both x_i and y_i, and any $\pi \in T_{x,y,x',y'}$ corresponds to individually permuting each of these four sets.)

It follows that for every $x, y \in HW_{\ell,\ell/2}$ that are d apart and any $x', y' \in HW_{\ell,\ell/2}$, if x', y' are also d apart then

$$\Pr_\pi\left[\pi(y) = y' \mid \pi(x) = x'\right] \;=\; \frac{((d/2)!(\ell/2-d/2)!)^2}{((\ell/2)!)^2} \;=\; \frac{1}{\binom{\ell/2}{d/2}^2},$$

and otherwise $\Pr_\pi[\pi(y) = y' \mid \pi(x) = x'] = 0$. Hence given any x, y that are d apart and $x' = \pi(x)$, the string $y' = \pi(y)$ is uniformly distributed over a set of size $\binom{\ell/2}{d/2}^2$. ■