i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits

Craig Gentry Shai Halevi Vinod Vaikuntanathan
IBM T.J. Watson Research Center

May 27, 2010

Abstract

Homomorphic encryption (HE) schemes enable computing functions on encrypted data, by means of a public Eval procedure that can be applied to ciphertexts. But the evaluated ciphertexts so generated may differ from freshly encrypted ones. This brings up the question of whether one can keep computing on evaluated ciphertexts. An *i-hop* homomorphic encryption scheme is one where Eval can be called on its own output up to i times, while still being able to decrypt the result. A *multi-hop* homomorphic encryption is a scheme which is i-hop for all i. In this work we study i-hop and multi-hop schemes in conjunction with the properties of function-privacy (i.e., Eval's output hides the function) and compactness (i.e., the output of Eval is short). We provide formal definitions and describe several constructions.

First, we observe that "bootstrapping" techniques can be used to convert any (1-hop) homomorphic encryption scheme into an i-hop scheme for any i, and the result inherits the function-privacy and/or compactness of the underlying scheme. However, if the underlying scheme is not compact (such as schemes derived from Yao circuits) then the complexity of the resulting i-hop scheme can be as high as $k^{O(i)}$.

We then describe a specific DDH-based multi-hop homomorphic encryption scheme that does not suffer from this exponential blowup. Although not compact, this scheme has complexity linear in the size of the composed function, independently of the number of hops. The main technical ingredient in this solution is a *re-randomizable* variant of the Yao circuits. Namely, given a garbled circuit, anyone can re-garble it in such a way that even the party that generated the original garbled circuit cannot recognize it. This construction may be of independent interest.

Keywords. BHHO encryption, Compactness, Function Privacy, Homomorphic Encryption, Secure Two-party Computation, Oblivious Transfer, Yao's Garbled Circuits.

1 Introduction

Computing on encrypted data epitomizes the conflict between privacy and functionality, and has been receiving 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.

One way to achieve this goal is via the paradigm of "computing with encrypted data" [15]: namely, the client encrypts its input x and sends the ciphertext to the server, and the server "evaluates the function f on the encrypted input". The server returns the evaluated ciphertext to the client, who decrypts it and recovers the result. An encryption scheme that supports computation on encrypted data is called a *homomorphic* encryption (HE) 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 $f(x)$. Over the years there were many proposals for encryption schemes that support computations of *some* functions on encrypted data. In this work, however, we are only interested in schemes that allow computation of *any* function on encrypted data.

A trivial implementation of the evaluation 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 client's work to the server. We are therefore interested also in the properties of *function privacy* (meaning that the evaluated ciphertext hides the function) and *compactness* (meaning roughly that the work involved in decrypting the evaluated ciphertext is less than in computing the function "from scratch").

1.1 Homomorphic encryption vs. secure function evaluation

Cachin, Camenisch, Kilian, and Müller [5] observed that the paradigm of "computing with encrypted data" with function privacy can be instantiated using any two-message protocol for two-party secure function evaluation (SFE). Indeed, the specifications of these two primitives are very similar: we can think of the first message in a 2-message SFE protocol as "encrypting" the first party's input, and the second message is the evaluation of a function held by the second party on that encryption.

Following the observation of Cachin et al., there is a simple folklore construction of public-key homomorphic encryption scheme from any two-message SFE protocol and an auxiliary CPA-secure public key encryption (e.g., [10, 3], see also Section 1.3 below). In particular, this construction can be used to convert a protocol based on Yao's garbled circuits [19] into a public-key homomorphic encryption scheme. The resulting scheme is function private but not compact: the client complexity is linear in the circuit size of the evaluated function f.

Many other schemes for "computing with encrypted data" can be found in the literature, with client complexity that depends in various forms on the complexity of the evaluated function f (e.g., its truth-table size [11], circuit depth [16], branching-program length [10], polynomial degree [1], etc.) The new scheme of Gentry [7] and its variants [18, 17] are the first schemes where the client complexity is independent of the complexity of f.

A remark about "fully homomorphic" encryption. We note that the schemes in [7, 18, 17] are unique in that evaluated ciphertexts can be made statistically close to freshly encrypted ones. We refer to schemes with this property as "fully homomorphic" (as opposed to just "homomorphic" for schemes without this property). It is easy to see that fully homomorphic schemes are both compact and function private. Also, all the issues with multi-hop evaluation that we consider in this work are trivialized for such schemes. For that reason, fully homomorphic schemes are not the focus of the current work.

1.2 Multi-Hop Homomorphic Encryption

Beyond the simple client-server setting from above, computing with encrypted data is useful also in settings where several functions are computed on the same encrypted data. For example, consider an email message encrypted under the public-key of Alice, which is sent to `alice@yahoo.com` and promptly forwarded to `alice@gmail.com`. Both Yahoo and Google have their own spam-tagging algorithms that they want to apply to incoming emails, 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 apply its spam-tagging algorithm to the encrypted email and produce an (encrypted and) tagged email, and then Google needs to apply its own spam-tagging algorithm to the result.

Another application with similar requirements is the setting of "autonomous mobile agents" that was considered by Cachin et al. [5]. In this application, a software agent is originated in some node in the network, and includes within it an encryption of data from that node. The agent then roams the network, visiting one node after another, and at each visited node it computes a function that depends on its current state and on the data from the visited node. Finally, the agent returns to its originator, and the originator learns the result of the composed function from all the visited nodes, as applied to the original data.

What we need in these applications is a *multi-hop* 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. We stress that evaluated ciphertexts may be very different from fresh ciphertexts, and it is not clear that the evaluation procedure of the scheme can process this modified form. (Indeed, homomorphic encryption schemes that are derived from generic secure computation protocols tend to have this problem; see below.) Cachin et al. [5] described a solution to the multi-hop setting based on Yao circuits, and our second construction in this work is an extension of that solution.

The multi-hop setting implies a new function-privacy requirement, namely *multi-hop function privacy*. For example, in the mail-forwarding example above, Google may worry that Yahoo! will try to collude with the sender and receiver of the email, in order to learn something about Google's spam-tagging techniques. Indeed, the solution of Cachin et al., which is described in Section 1.3 below, suffers from exactly this problem. Ensuring multi-hop function privacy is the main focus of our work.

1.3 Homomorphic encryption from Yao circuits

For the sake of concreteness, we now describe the folklore construction of (1-hop) homomorphic encryption from any two-message SFE protocol, and the extension of Cachin et al. to the multi-hop setting based on Yao circuits. Consider the structure of a two-message SFE protocol where a client holds an input x, a server holds a function f, and the client wishes to receive $f(x)$.

- The client sends to the server a message that "encodes" its input x, and yet does not reveal x to a computationally bounded server. In other words, the client's message acts as an *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 function-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.

The above is still not quite a public-key encryption scheme: in particular, there is no public key involved, and the same party (the client) is doing both the encryption and the decryption. In contrast, a public key homomorphic encryption should be thought of as a three-player game: first a recipient publishes a public key, then a sender (client) encrypts the data x under that public key, next an evaluator (server) computes a function f on the encrypted data, and finally the recipient decrypts the result and recovers $f(x)$.

Fortunately, we can get a public key HE scheme from a two-message SFE protocol by using an auxiliary standard public-key encryption scheme: The recipient chooses a public/secret key pair for some semantically secure encryption scheme, the sender sends the first-message SFE message and in addition also the encryption of the SFE randomness under the public key, and the evaluator forwards the encrypted randomness to the recipient together with the second-message SFE message. The recipient uses its secret key to decrypt and recover the SFE randomness, and then uses the SFE procedure with this randomness to recover $f(x)$.

Extending to more than one hop. Consider next the setting where there is a sender who holds an input x, two evaluators E_1 and E_2 who hold functions f_1 and f_2 respectively, and the recipient 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 1-hop homomorphic encryption (such as the construction above from a generic 2-message SFE protocol), we only offer a partial answer to this question: In Theorem 1 we show that "bootstrapping" techniques [7] can be used to transform a 1-hop HE scheme into an i-Hop scheme for any i, but the size of the ciphertext could grow by a polynomial factor for every hop (and hence we can only carry out this procedure for a constant number of hops).

On the other hand, a scheme based on Yao's garbled circuits [19] is easy to extend to many hops without the exponential blowup in complexity. Recall that in Yao's garbled circuit construction, the server (who has a function) chooses two random labels for every wire in the circuit that computes that function, 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 gate 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.

Cachin et al. [5] noted that this construction is extendable to more than one hop: the second evaluator E_2 receives the garbled circuit from the first evaluator 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. Moreover this extension offers a weak form of function 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 recipient.

However, privacy breaks down completely when E_1 colludes with the recipient. 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 main technical contribution is a re-randomizable variant of Yao circuits, allowing E_2 to re-randomize the labels of E_1's garbled circuit, thus obtaining privacy even against a collusion of E_1 and the recipient.

1.4 Summary of our results

Definition of multi-hop homomorphic encryption. Informally, in an i-hop HE 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 $E_1, \ldots, E_i$ denote the i players – evaluators – that hold the functions $f_1, \ldots, f_i$).

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

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

The definition of multi-hop function 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 [10]. 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 I: 1-hop → i-hop. In Section 3, we show how to convert a 1-hop HE scheme into an i-hop HE scheme for any i. This construction uses a bootstrapping technique, similar to [7]: given a function f and an evaluated ciphertext c that decrypts to some value x, we can express the value $f(x)$ as a function of the secret key, $F_{f,c}(\mathrm{SK}) \stackrel{\text{def}}{=} f(\mathsf{Dec}(\mathrm{SK}, c)) = f(x)$. If we add to the public key a fresh encryption of the secret key, we can then use the evaluation procedure of the scheme to evaluate $F_{f,c}$ on this fresh encryption, thus obtaining a ciphertext that decrypts to $f(x)$. As described, this construction relies on circular security of the underlying scheme (since we publish an encryption of the secret key). Just as in [7], we can avoid relying on circular security and still support up to i hops, by having i public/secret key pairs and encrypting the j'th secret key under the $j+1$'st public key.

We note, however, that for non-compact HE schemes, the size of the evaluated ciphertext can be polynomially larger than the size of the evaluated function. Hence the ciphertext in the resulting i-hop scheme could grow by a factor of up to $k^{O(i)}$ after i hops, where k is the security parameter. Thus, this construction is viable only for a constant number of hops. Since by the folklore construction (described in section 1.3), the existence of 1-hop HE schemes is equivalent to the existence of two-message SFE protocols, we get:

Theorem 1 (Informal) *If two-message secure function evaluation protocols exist, then for any constant i there is a public key encryption scheme $\mathcal{H}^{(i)}$ which is i-hop homomorphic and i-hop function-private. There is a fixed polynomial $q(k)$ in the security parameter k such that on evaluating functions $f_1, \ldots, f_i$ on a fresh ciphertext of $\mathcal{H}^{(i)}$, the resulting evaluated ciphertext has size at most $\left(\sum_{j=1}^{i} |f_j|\right) \cdot q(k)^i$.*

We also note that if the underlying 1-hop HE scheme is compact, then the construction above can be carried out without the exponential blowup, hence we can extend it to an i-hop scheme for any polynomial i. Moreover, similar bootstrapping techniques can be used to combine two 1-hop HE schemes – one compact but not private and the other private but not compact – into a single 1-hop scheme which is both private and compact. Using the construction above we can then extend it to a *compact and private* i-hop scheme for any polynomial i.

Theorem 2 (Informal) *Assume that there exist a 1-hop compact HE scheme, and a (possibly different) 1-hop function-private HE scheme. Then, for every polynomial $p(k)$ there is an encryption scheme $\mathcal{H}^{(p)}$, which is $p(k)$-hop homomorphic and $p(k)$-hop private. There is a fixed polynomial $q(k)$ such that on evaluating functions $f_1, \ldots, f_{p(k)}$ on a fresh ciphertext of $\mathcal{H}^{(p)}$, the resulting ciphertext has size $q(k)$ (independent of the size of the functions f_j).*

Construction II: Re-randomizable Yao → multi-hop. In Section 5, we describe a scheme that can handle any polynomial number of hops, and is semantically secure and function private 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.

This encryption scheme essentially amends the Yao-garbled-circuit construction from the previous section, which only offered a weak form of function privacy. 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 describe a variant of the garbled circuit construction that allows such re-randomization. For the construction, we rely on the encryption scheme of Boneh-Halevi-Hamburg-Ostrovsky [4], and on the oblivious-transfer protocol of Naor-Pinkas and Aiello-Ishai-Reingold [13, 2] (both of which are based on the decisional Diffie-Hellman assumption, and have "nice" additive homomorphic properties).

Theorem 3 (Informal) *Under the decisional Diffie-Hellman assumption, there is a public-key multi-hop homomorphic encryption scheme $\mathcal{H}^*$ which is function-private for any number of hops. Moreover, there is a fixed polynomial $q(k)$ in the security parameter such that on evaluating functions $f_1, \ldots, f_d$ on a fresh ciphertext, the resulting ciphertext has size $\left(\sum_{i=1}^{d} |f_i|\right) \cdot q(k)$.*

2 Definitions of Homomorphic Encryption

Nearly all our definitions rely on a security parameter, which is usually implicit. By $x \leftarrow X$ and $x \in_R S$ we denote drawing from a distribution and choosing uniformly from a set. We call a procedure efficient if it runs in time polynomial in the length of its input. We say that two distributions are computationally indistinguishable if any *efficient* distinguisher has only a negligible advantage in distinguishing them. Throughout the writeup, adversarial algorithms are always nonuniform.

A homomorphic encryption scheme consists of four procedures, $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. KeyGen takes as input the security parameter and outputs a public/secret key-pair, Enc takes the public key and a 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.

We sometimes use the convention that these procedures output the randomness that they used, for example writing $(c', r') \leftarrow \mathsf{Eval}(\textsc{pk}, f, c)$ rather than just $c' \leftarrow \mathsf{Eval}(\textsc{pk}, f, c)$.

Multi-hop evaluation. We extend the Eval procedure from a single function 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. If $\vec{f}$ is a compatible sequence of t functions, we denote its j^{th} 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 $c_i \leftarrow \mathsf{Eval}(\textsc{pk}, f_i, c_{i-1})$, outputting the last ciphertext c_t.

Definition 1 (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 $t \leq i$ functions, every input x to f_1, every $(\textsc{pk}, \textsc{sk})$ in the support of KeyGen, and every c in the support of $\mathsf{Enc}(\textsc{pk}; x)$,*

$$\mathsf{Dec}\big(\textsc{sk}, \mathsf{Eval}^*(\textsc{pk}, \vec{f}, c)\big) = (f_t \circ \cdots \circ f_1)(x)$$

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

We note that 1-hop homomorphic encryption is just the usual notion of homomorphic encryption, as formalized, e.g., in [10, Def 5].

2.0.1 Semantic security, function privacy, and compactness

Semantic security (aka CPA security) [9] is defined exactly as in regular public-key encryption schemes (without regard to Eval). We provide the definition here for self containment.

Definition 2 (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 $(\textsc{pk}, \textsc{sk})$ and $\mathcal{A}$ is given $\textsc{pk}$. 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 $c \leftarrow \mathsf{Enc}(\textsc{pk}; M_\sigma)$. 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 same above as the *CPA game.*

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

Definition 3 (function privacy - honest-but-curious) *An i-hop homomorphic encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is function-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 $t \le i$, every $j \le t$, 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*):*

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

We remark that Definition 3 can be extended in several different ways. An obvious extension would be to consider the malicious case (with or without assuming that the public key and the initial ciphertext were created honestly). A second possible extension is to consider a more general adversarial structure, where the attacker can corrupt an arbitrary subset of the players (the encryptor, the evaluators, and the decryptor), and we still want to ensure the privacy of the non-corrupted ones. Yet another extension to Definitions 1 and 3 is to consider an arbitrary network of functions (and not just a single chain). Finally, one could 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 4 (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 k, such that decryption of any ciphertext (even one that is the output of* Eval^**) w.r.t. the security parameter k can be implemented by a circuit of size at most $p(k)$.*

Namely, for every value of k, there exists a circuit $\mathsf{Dec}^{(k)}$ of size at most $p(k)$, such that the i-Hop property from Definition 1 holds for that decryption circuit.

The name "compactness" is justified by the fact that the length of 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 function privacy together are still formally weaker than the Ishai-Paskin notion of "privacy with size hiding" [10, Def 8].

3 From 1-Hop to i-Hop Homomorphic Encryption

Below we show how to transform a 1-hop HE scheme to an i-hop scheme for any constant $i > 0$. The price that we pay, however, is that the complexity of the i-hop scheme (and in particular, the length of the evaluated ciphertexts) may grow as large as $k^{O(i)}$ (for security parameter k).

The idea is that each evaluator (with function f) in the chain, upon receiving the "evaluated ciphertext" c from its predecessor, applies again the evaluation procedure, but not to its original function f. Rather, it applies the evaluation procedure to the concatenation of f with the decryption function, namely to the function $F_{f,c}(\text{SK}) \stackrel{\text{def}}{=} f\big(\mathsf{Dec}(\text{SK}; c)\big)$. This technique, which is reminiscent of Gentry's "bootstrapping" technique [7], works because (by induction) applying $\mathsf{Dec}(\text{SK}, c)$ on the previous evaluated ciphertext outputs the value $(f_{j-1} \circ \cdots \circ f_1)(x)$.

The Construction. Let $\mathcal{H} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a function-private homomorphic 1-hop encryption scheme (that need not be compact). Let i be a constant parameter of the system (that represents the number of hops that we are shooting for). We construct a function-private i-hop homomorphic encryption scheme $\mathcal{H}^{(i)} = (\mathsf{KeyGen}^{(i)}, \mathsf{Enc}^{(i)}, \mathsf{Eval}^{(i)}, \mathsf{Dec}^{(i)})$ as follows.

$\mathsf{KeyGen}^{(i)}$**:** Run KeyGen for $i+1$ times, to get for $j = 0, 1, \ldots, i$:

$$(\text{PK}_j, \text{SK}_j) \leftarrow \mathsf{KeyGen}, \qquad \text{and for } j < i \text{ also: } \alpha_j \leftarrow \mathsf{Enc}\big(\underbrace{\text{PK}_{j+1}}_{\text{key}};\ \underbrace{\text{SK}_j}_{\text{ptxt}}\big)$$

Defining $\alpha_i = \perp$, the public key is the set $\text{PK}^{(i)} = \{(\text{PK}_j, \alpha_j) : j = 0, 1, \ldots, i\}$, and the secret key is $\text{SK}^{(i)} = (\text{SK}_0, \text{SK}_1, \ldots, \text{SK}_i)$.

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

$\mathsf{Eval}^{(i)}(\text{PK}^{(i)}, \tilde{c}, f_{j+1})$**:** Parse the ciphertext as $\tilde{c} = \big[\mathsf{level}\text{-}j, c_j\big]$(with $j < i$ and c_j an $\mathcal{H}$-ciphertext) . Compute the description of the function $F_{f_{j+1},c_j}(s) \stackrel{\text{def}}{=} f_{j+1}(\mathsf{Dec}(s;\ c_j))$, and set ,

$$c_{j+1} \leftarrow \mathsf{Eval}(\text{PK}_{j+1};\ F_{f_{j+1},c_j},\ \alpha_j).$$

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

$\mathsf{Dec}^{(i)}(\text{SK}^{(i)};\ \tilde{c})$**:** Parse the ciphertext as $\tilde{c} = \big[\mathsf{level}\text{-}j, c_j\big]$(with $j \le i$ and c_j an $\mathcal{H}$-ciphertext) . Compute and output $y \leftarrow \mathsf{Dec}(\text{SK}_j;\ c_j)$.

Theorem 4 *The scheme $\mathcal{H}^{(i)}$ above is an i-hop function private homomorphic encryption scheme.*

Proof (sketch) Correctness is easy to establish by induction. The correctness of the underlying 1-hop homomorphic encryption scheme $\mathcal{H}$ implies that for all $j \le i$ we have

$$\begin{aligned}
\mathsf{Dec}(\text{SK}_j, c_j) &= \mathsf{Dec}(\text{SK}_j, \mathsf{Eval}(\text{PK}_j;\ F_{f_j,c_{j-1}},\ \alpha_{j-1})) \\
&\stackrel{(a)}{=} F_{f_j,c_{j-1}}(\text{SK}_{j-1}) \stackrel{(b)}{=} f_j(\mathsf{Dec}(\text{SK}_{j-1}, c_{j-1})) \stackrel{(c)}{=} (f_j \circ \ldots \circ f_1)(x),
\end{aligned}$$

where f_j is the function that was used in the j'th hop, Equality (a) holds by correctness of the underlying 1-hop scheme, Equality (b) holds by definition of $F_{f_j,c_{j-1}}$, and Equality (c) holds by the induction hypothesis.

Semantic security of $\mathcal{H}^{(i)}$ follows trivially from that of the underlying (1-hop) encryption scheme. Similarly, i-hop function privacy follows easily from the 1-hop privacy of the underlying scheme (and the fact that the size of $F_{f_j,c_{j-1}}$ that the $\mathcal{H}$ simulator needs can be computed easily from the size of f_j and the size of c_{j-1} both of which the simulator for $\mathcal{H}^{(i)}$ knows). ■

Complexity. For "generic" 1-hop encryption schemes (such as the one that we can obtain from two-message SFE using the folklore construction described in Section 1.3), the size of the ciphertext resulting from $\mathsf{Eval}(f, c)$ is larger than the input length $|c|+|f|$ by some factor K which is polynomial in the security parameter k. Hence the size of the circuit for $F_{f_j,c_{j-1}}$ in our construction is at least

$$K(\cdots K(K(|c_0| + |f_1|) + |f_2|) \cdots) + |f_j| = |c_0|K^{j-1} + \sum_{t=1}^{j} |f_t|K^{j-t} \ = \ \Big(\sum_{t=1}^{j} |f_j|\Big) \cdot k^{O(j)}$$

which means that after i hops the ciphertext grows as $k^{O(i)}$. We comment also that the original encryptor can put non-constant many pairs (PK_j, α_j) in the public key, thus permitting the evaluation of any constant number of hops (as opposed to having a particular constant parameter i).

3.1 Compact and Function-Private Homomorphic Encryption

Recall that the exponential blowup in the construction above is due to the fact that the ciphertext that results from Eval is larger than the function size (by a multiplicative factor). On the other hand, if the underlying 1-hop scheme is compact (and function-private), then the construction above would yield a compact (and function-private) i-hop scheme.

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 scheme for all i, by the observation above).

The idea is 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 function-private since the only dependence of the compact ciphertext 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 function-privacy of f.[1]

We note that when using this technique, we again get a "hard-wired" parameter i that limits the number of hops that we can handle: to get an i-hop scheme, the public key must have size linear in i. Thus, the resulting scheme is not multi-hop, according to Definition 1. This limitation can be circumvented by relying on the circular security of the resulting 1-hop schemes; the details are deferred to the full version. This limitation can be circumvented by relying on circular security of the resulting 1-hop scheme, see Remark 1.

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

$\mathsf{KeyGen}^{(i)}$**:** Run each of $p\mathsf{KeyGen}, c\mathsf{KeyGen}$ for $i+1$ times, to get for $j = 0, 1, \dots, i$:

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

Defining $\alpha_i = \beta_i = \perp$, the public key is the set $\mathrm{PK}^{(i)} = \{(p\mathrm{PK}_j c\mathrm{PK}_j, \alpha_j, \beta_j) : j = 0, 1, \dots, i\}$, and the secret key is $\mathrm{SK}^{(i)} = (c\mathrm{SK}_0, c\mathrm{SK}_1, \dots, c\mathrm{SK}_i)$.

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

$\mathsf{Eval}^{(i)}(\mathrm{PK}^{(i)}; \tilde{c}, f_{j+1})$**:** Parse the ciphertext as $\tilde{c} = \big[\mathsf{level}\text{-}j, c_j\big]$ (with $j < i$ and c_j a compact ciphertext).

[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.

Compute the description of the function $F_{f_{j+1},c_j}(s) \stackrel{\text{def}}{=} f(c\mathsf{Dec}(s;\ c_j))$, 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 $G_{c'_j}(s) \stackrel{\text{def}}{=} p\mathsf{Dec}(s;\ c'_j))$, and set

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

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

$\mathsf{Dec}^{(i)}(\text{SK}^{(i)};\ \tilde{c})$**:** Parse the ciphertext as $\tilde{c} = \big[\mathsf{level}\text{-}j, c_j\big]$ (with $j \le i$ and c_j a compact ciphertext). Compute and output $y \leftarrow c\mathsf{Dec}(c\text{SK}_j;\ c_j)$.

Theorem 5 *For any $i = \mathsf{poly}(n)$, the scheme $\mathcal{H}^{(i)}$ above is a compact function 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 $t \le i$ 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

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

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 $\mathcal{H}^{(i)}$ 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 $\mathcal{H}^{(i)}$). Also, compactness follows trivially since the decryption algorithm is the same as that of the underlying compact scheme.

Similarly, i-hop function 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 1 *To get a multi-hop scheme (without the parameter i), 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 function private, then we get a multi-hop compact and private scheme.*

4 Extendable and Re-randomizable Secure Computation

Below we define the tool of "extendable and re-randomizable SFE", and show how it is used for multi-hop homomorphic encryption. In the next section we show that this tool can be implemented under the decisional Diffie-Hellman assumption. We begin with definitions (which are similar to Ishai et al. [10]).

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 $U(f,x)$).

A two-message 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-message 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-message message m_1, and a circuit f. The output is m_2, the second-message message of the SFE protocol, $m_2 \leftarrow \mathsf{SFE2}(f, m_1)$.

- $\mathsf{SFE\text{-}Out}(r_1, m_2)$ is a procedure that takes Alice's state r_1 and Bob's second-message message m_2, and outputs some y.

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.

By $\mathsf{SFE1}(x)$ (resp. $\mathsf{SFE2}(m_1, f)$), we mean the distribution generated by the respective algorithms (over the choice of their randomness). We also say that $(m_1, r_1) \in \mathsf{SFE1}(x)$ (resp. $(m_2, r_2) \in \mathsf{SFE2}(m_1, f)$) to denote a particular element in the support of the distribution (together with the randomness involved).

Definition 5 (Client and (honest-but-curious) Server privacy) *A protocol* $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ *is said to be:*

- Client-private*, 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').*

- Server-private *in the honest-but-curious model, if there exists a polynomial time simulator* Sim *such that for every input x and function f, and every* $(m_1, r_1) \in \mathsf{SFE1}(x)$*, the distributions* $\mathsf{SFE2}(f, m_1)$ *and* $\mathsf{Sim}(m_1, 1^{|f|}, f(x))$ *are indistinguishable (even given f, x, m_1 and r_1).*

We now define the notion of an extendable SFE protocol.

Definition 6 (Extendable SFE, honest-but-curious) *A two-message SFE protocol* $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ *is extendable, if there exists an efficient procedure* Extend *such that for any two compatible functions f and f', any input x to f, and for every* $(m_1, r_1) \in \mathsf{SFE1}(x)$*, the distributions* $\mathsf{Extend}(\mathsf{SFE2}(m_1, f), f')$ *and* $\mathsf{SFE2}(m_1, f' \circ f)$ *are indistinguishable (even given x, f, f', m_1 and r_1).*

Extendable SFE from Yao Circuits. The construction of Cachin et al. [5, Sec. 5] can be cast in our language as describing an extendable SFE protocol based on Yao's garbled circuit construction [19]. As described in the introduction, the idea is that since the garbled circuit for f includes both the 0-label and the 1-label on any *output wire*, it can be extended by treating these labels as the input labels for f'.

We comment that garbling the gates hides only the type of these gates and not the topology of a circuit. To hide the function we must also use some form of canonicalization of circuits, so that all circuits of a given size will have the same topology. Moreover, to meet our definition of extendibility, it must be the case that canonicalizing f, then extending it with f' and canonicalizing the whole thing yields the same topology as canonicalizing the composed function $f' \circ f$.

We note that such canonicalization is possible, and the size of the canonicalized circuits does not grow much. For example, a circuit of maximum width w can be canonicalized to a leveled circuit with width w at every level, and a big "multiplexer gate" between every two successive levels that determines what output from the lower level goes to what input in the upper one. To get the additional property that we need (where the order of canonicalization does not matter) we would also have w output wires in the circuit, where the redundant output wires have both labels set to 0. (We may also need to supply some dummy gates that take as input the input wires and have both output labels set to 0, to be able to pad the circuit if the maximum width of f' is larger than that of f.)

From Extendable to Re-randomizable. Note that extendable SFE by itself already yields multi-hop homomorphic encryption with a weak form of function-privacy: to a recipient that does not know the intermediate values (namely, the output of $\mathsf{SFE2}(m_1, f)$), the output of Extend looks just as if it was generated "from scratch" by running $\mathsf{SFE2}$ with input $f' \circ f$, so Extend hides the function if $\mathsf{SFE2}$ does. This means that when the protocol Π is used for many hops, then as long as all the intermediate hops are "trusted" not to reveal their intermediate results (and only the sender and the recipient are honest-but-curious), using Extend would maintain the privacy of everyone's functions.

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

Definition 7 (Re-randomizable SFE, honest-but-curious) *A two-message SFE protocol* Π *is re-randomizable if there exists an efficient procedure* reRand *such that for every input* x *and function* f *and every* $(m_1, r_1) \in \mathsf{SFE1}(x)$ *and* $(m_2, r_2) \in \mathsf{SFE2}(m_1, f)$*, the distributions* $\mathsf{reRand}(m_1, m_2)$ *and* $\mathsf{SFE2}(m_1, f)$ *are indistinguishable, even given* x, f, m_1, r_1, m_2, r_2*.*

From Extendable and Re-randomizable SFE to Multi-hop HE. Let $\Pi = (\mathsf{SFE1}, \mathsf{SFE2}, \mathsf{SFE\text{-}Out})$ be an extendable and re-randomizable two message SFE protocol with client and server privacy, and let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically secure public-key encryption scheme. We now describe the construction of the multi-hop homomorphic scheme $\mathcal{H}^* = (\mathsf{KeyGen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*, \mathsf{Eval}^*)$.

The key generation KeyGen^* is the same as KeyGen for the underlying encryption. The encryption procedure $\mathsf{Enc}^*(\text{PK}; x)$ first runs $(m_1, r_1) \leftarrow \mathsf{SFE1}(x)$, then encrypts r_1 using $\mathcal{E}$ to get $c \leftarrow \mathsf{Enc}(\text{PK}; r_1)$, and finally, computes $m_2 \leftarrow \mathsf{SFE2}(m_1, f_{ID})$ (where f_{ID} is the identity function). The ciphertext is (c, m_1, m_2).

To evaluate a function f_j on an $\mathcal{H}^*$-ciphertext c_{j-1}, first parse c_{j-1} as a tuple $(c, m_1, m_2^{(j-1)})$, then set $m_2' \leftarrow \mathsf{Extend}(m_2^{(j-1)}, f_j)$ and $m_2^{(j)} \leftarrow \mathsf{reRand}(m_1, m_2')$. The evaluated ciphertext is $(c, m_1, m_2^{(j)})$. Decrypting $c_j = (c, m_1, m_2^{(j)})$ consists of using the decryption of $\mathcal{E}$ to get $r_1 \leftarrow \mathsf{Dec}(\text{SK}, c)$, then outputting $y \leftarrow \mathsf{SFE\text{-}Out}(r_1, m_2^{(j)})$.

Theorem 6 (Extendable+Re-randomizable $\Rightarrow$ Multi-hop) *Assume that the encryption scheme $\mathcal{E}$ is semantically secure, the SFE protocol Π is extendable and re-randomizable with client and server privacy, and in addition that the size of any function f can be efficiently determined from the output of $\mathsf{SFE2}(m_1, f)$.*

Then the scheme $\mathcal{H}^*$ above is a multi-hop function-private homomorphic encryption scheme. Moreover, the size of an evaluated ciphertext in $\mathcal{H}^*$ does not depend on the number of hops, but only on the size of the composed function.

Proof (sketch) Correctness of $\mathcal{H}^*$ 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 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. Semantic security follows from semantic security of the underlying encryption and from the client-privacy of Π.

To show function privacy, we need to describe a simulator $\mathsf{Sim}_{\mathcal{H}^*}$ that on input $c_{j-1} = (c, m_1, m_2^{(j-1)})$, $|f_j|$, and $y_j = (f_1 \circ \cdots \circ f_j)(x)$, generates a distribution indistinguishable from $c_j = (c, m_1, m_2^{(j)})$. The simulator recovers from $m_2^{(j-1)}$ the size $|f_1 \circ \cdots \circ f_{j-1}|$ and adds it to $|f_j|$ to get $\gamma = |f_1 \circ \cdots \circ f_j|$. Then $\mathsf{Sim}_{\mathcal{H}^*}$ uses the simulator for Π to get $m_2^{(j)} \leftarrow \mathsf{Sim}_\Pi(m_1, \gamma, y_j)$ and outputs $c_j = (c, m_1, m_2^{(j)})$.

By the server-privacy of Π, the distribution of $m_2^{(j)}$ so generated is indistinguishable from $\mathsf{SFE2}(m_1, f_1 \circ \cdots \circ f_j)$. On the other hand, by the extendability and re-randomizability properties of Π, the distribution of $m_2^{(j)}$ in $\mathcal{H}^*$ is also indistinguishable from the same $\mathsf{SFE2}(m_1, f_1 \circ \cdots \circ f_j)$. Hence these two distributions are indistinguishable from each other. ■

5 Extendable and Re-randomizable SFE from DDH

Given Theorem 6, we now focus on building an extendable and re-randomizable SFE protocol. Our starting point is Yao's garbled circuit construction [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/Aiello-Ishai-Reingold [13, 2], and on the encryption scheme of Boneh-Halevi-Hamburg-Ostrovsky [4], the security of both of which is equivalent to the decisional Diffie-Hellman assumption. Below we briefly summarize some properties of these building blocks; a slightly longer description (and the definitions of OT) can be found in the appendix.

Re-randomizable oblivious transfer. The protocol in [13, 2] is a two-message protocol. The receiver that has a choice bit $\sigma \in \{0,1\}$ sends the first message $m_1 \leftarrow OT1(\sigma)$, the sender that has two bits $\gamma_0, \gamma_1 \in \{0,1\}$ replies with $m_2 \leftarrow OT2(m_1, \gamma_0, \gamma_1)$, and the receiver can recover the bit γ_σ from m_2 and the state that it keeps. Receiver security means that $OT1(0)$, $OT1(1)$ are indistinguishable, and sender security means that $OT2(m_1, \gamma_0, \gamma_1)$ can be simulated knowing only m_1 and γ_σ. We note that if the sender has two strings $\vec{\gamma}_0, \vec{\gamma}_1$, (rather than just two bits) then it can use the same m_1 from the receiver and send many m_2's in reply, one for every bit position in the input vectors.

Another property we use is that the protocol from [13, 2] is *re-randomizable*: given m_1, m_2, anyone can re-randomize the reply, computing another random m_2' from the distribution $OT2(m_1, \gamma_0, \gamma_1)$ (even without knowing γ_0, γ_1).

Key and plaintext additively homomorphic encryption. The BHHO scheme [4] is a semantically secure public key encryption scheme where the secret key is a string $\vec{s} \in \{0,1\}^\ell$ and the plaintext is also a string $\vec{x} \in \{0,1\}^n$. (In our application we use $n = 2\ell$.) The public key and ciphertexts are vectors of elements over a group of some prime order q.

The BHHO scheme has the following "additively homomorphic" property: Let T, T' be two known affine transformations on vectors over Z_q *that map 0-1 vectors to 0-1 vectors of the same length*. Then, given a public key PK corresponding to some secret key $\vec{s}$ and a ciphertext $c \in \mathsf{Enc}(\text{PK}; \vec{x})$, anyone can generate a random public key PK$'$ corresponding to $T(\vec{s})$ and a random ciphertext $c' \in \mathsf{Enc}(\text{PK}'; T'(\vec{x}))$. In particular, this means that anyone can XOR known strings Δ, Δ' into $\vec{s}$ and $\vec{x}$, and also anyone can permute the bits in either $\vec{s}$ or $\vec{x}$ (or both) according to known permutations.

5.1 Our Construction

Our construction closely follows Yao's original garbled circuit construction [19]. The client (Alice) on input $\vec{x} = \langle x_1, \ldots, x_n \rangle$, sends n first 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.[2]

Bob receives from Alice the n OT first 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 message $m_1[i]$, Bob computes the OT second 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.

[2] 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.

Bob chooses four fresh random 2ℓ-bit masks $\delta_{i,j}$ for $i,j \in \{0,1\}$ and computes the four pairs:

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

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). The "gadget" for this gate consists of the four pairs of ciphertexts from Eq. (1) in random order. The garbled circuit that Bob sends back to Alice consists of the n OT second 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.)

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 [12, Thm 5]. The client-privacy part is completely identical to [12], and is omitted here. The high-level structure of the server-privacy proof is also similar to [12], 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-message 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-message 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:

$$\begin{array}{ccc} \left(\mathsf{Enc}_{L_{w_1}}(\delta_1),\ \mathsf{Enc}_{L_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_1) \right) & , & \left(\mathsf{Enc}_{L_{w_1}}(\delta_2),\ \mathsf{Enc}_{L'_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_2) \right) \\ \left(\mathsf{Enc}_{L'_{w_1}}(\delta_3),\ \mathsf{Enc}_{L_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_3) \right) & , & \left(\mathsf{Enc}_{L'_{w_1}}(\delta_4),\ \mathsf{Enc}_{L'_{w_2}}((L_{w_3}|0^\ell) \oplus \delta_4) \right) \end{array} \tag{2}$$

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-message 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 active label of the corresponding gate-output wire (with ℓ trailing 0's).[3]

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

[3] This may or may not be the same value that was encrypted in these ciphertexts in game $i-1$.

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

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

In Eq. (3), 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 $\delta'_{j,2} \stackrel{\text{def}}{=} \delta_{j,2} \oplus ((L_{u_j} \oplus X_j)|0^\ell)$ and $\delta'_{j,4} \stackrel{\text{def}}{=} \delta_{j,4} \oplus ((L_{u_j} \oplus Z_j)|0^\ell)$ (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 $i-1$. ■

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 5.2. 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. [14, Sec 5.2]). We use this stronger result in our proof of the re-randomization property in Section 5.2.

5.2 Re-randomizing garbled circuits

We 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: 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 9 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 (see Lemma 9 below). [4] The Naor-Segev result [14] 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 8 *Under the DDH assumption, the BHHO-based protocol from above is computationally re-randomizable.*

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 re-randomizes them to hide the permutation. Namely, the second-message 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 re-randomizes them all.

For the garbled circuit portion, consider one particular gate in the circuit, which is represented by four ciphertexts as in Eq. (1)

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

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

[4] One can view Lemma 9 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 when the Hamming distance between L, L' is large enough.

chose, it can transform these ciphertexts first into

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

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.

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 [14]. 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 9 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 permutations 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. [6]), that is

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

Lemma 9 *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 B. 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. ■

Acknowledgments. We thank Yuval Ishai for several inspiring discussions.

References

[1] C. Aguilar-Melchor, P. Gaborit, and J. Herranz. Additively Homomorphic Encryption with d-Operand Multiplications. In *Advances in Cryptology - CRYPTO'10*, Lecture Notes in Computer Science. Springer, 2010. `http://eprint.iacr.org/2008/378/`.

[2] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology - EUROCRYPT'01*, volume 2139 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2001.

[3] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai. Bounded key-dependent message security. In *Advances in Cryptology - EUROCRYPT'10*, Lecture Notes in Computer Science. Springer, 2010. `http://eprint.iacr.org/2009/511`.

[4] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *Advances in Cryptology - CRYPTO'08*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2008.

[5] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *ICALP'00*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523. Springer, 2000.

[6] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology - EUROCRYPT'04*, volume 3027, pages 523–540. Springer, 2004.

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

[8] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption schemes. Cryptology ePrint Archive, Report 2010/145, 2010. `http://eprint.iacr.org/`.

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

[10] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography - TCC'07*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[11] E. Kushilevitz and R. Ostrovsky. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *FOCS'97*, pages 364–373. IEEE, 1997.

[12] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptology*, 22(2):161–188, 2009.

[13] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM symposium on Discrete algorithms - SODA'01*, pages 448–457. ACM, 2001.

[14] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *Advances in Cryptology - CRYPTO'09*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2009.

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

[16] T. Sander, A. Young, and M. Yung. Non-interactive CryptoComputing for NC1. In *40th Annual Symposium on Foundations of Computer Science - FOCS'99*, pages 554–567. IEEE, 1999.

[17] N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC'10*, Lecture Notes in Computer Science. Springer, 2010. `http://eprint.iacr.org/2009/571`.

[18] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT'10*, Lecture Notes in Computer Science. Springer, 2010. `http://eprint.iacr.org/2009/616`.

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

A Building Blocks for the Construction from Section 5

A.1 Re-randomizable oblivious transfer

Definition 8 (Oblivious Transfer – honest but curious) *A two-message 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 first-message 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 9 (Re-randomizable OT) *A two-message oblivious transfer protocol is re-randomizable if there exists an efficient algorithm* reRand *such that for every three bits $\sigma, \gamma_0, \gamma_1$, every $(m_1, r_1) \in$*

$OT1(\sigma)$ and every second-message message $(m_2, r_2) \in OT2(1^k, \gamma_0, \gamma_1, m_1)$, the distributions $\mathsf{reRand}(m_1, m_2)$ *and $OT2(\gamma_0, \gamma_1, m_1)$ are indistinguishable, even given $\sigma, \gamma_0, \gamma_1, m_1, r_1, m_2, r_2$.*

Naor-Pinkas and Aiello et al. [13, 2] proved that the following protocol meets Definition 8.[5] 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 is also re-randomizable: On input $m_1 = (g, h, x, y_0, y_1)$ and $m_2 = (a_0, b_0, a_1, b_1)$, the reRand 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 $b_i' := x^{s_i'} y_i^{t_i'} \cdot b_i$ $(= x^{s_i'+s_i} y_i^{t_i'+t_i} \cdot g^{\gamma_i})$.

A.2 The BHHO encryption scheme

Boneh et al. described in [4] 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 $\ell \stackrel{\text{def}}{=} \lceil 3 \log q \rceil$.

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 outputting zero if $u_{\ell+1} \cdot \prod_{i=1}^{\ell} u_i^{s_i} = 1$, one if $u_{\ell+1} \cdot \prod_{i=1}^{\ell} u_i^{s_i} = g$, 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 [4] 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 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 $\vec{s'}$. 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.

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

(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 [4]. 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.)

B Proof of Lemma 9

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

$$\widetilde{H}_\infty(\pi(y) \mid x, y, \pi(x)) = 2\log\binom{n/2}{d/2} \tag{4}$$

where $\pi \leftarrow S_\ell$ is uniformly random. This immediately implies Lemma 9, since

$$\begin{aligned}
\widetilde{H}_\infty(\pi(L_2) \mid L_1, L_2, \pi(L_1)) &= -\log \mathop{\mathbb{E}}_{x,y\leftarrow HW_{n,n/2}}\left(2^{-\widetilde{H}_\infty(\pi(y)|x,y,\pi(x))}\right) && \text{(by definition of } \widetilde{H}_\infty\text{)}\\
&= -\log\left(\sum_{\text{even } d}\Pr[HD(\ell_1,\ell_2)=d]\cdot\frac{1}{\binom{\ell/2}{d/2}^2}\right) && \text{(by Equation 4)}\\
&= -\log\left(\frac{1}{\binom{\ell}{\ell/2}}\cdot\sum_{\text{even } d}\binom{\ell/2}{d/2}^2\cdot\frac{1}{\binom{\ell/2}{d/2}^2}\right) && \text{(by prob. calculation)}\\
&= \log\left(\tbinom{\ell}{\ell/2}/(\tfrac{\ell}{2}+1)\right) \;\geq\; \log\left(2^{\ell-1}/(\tfrac{\ell}{2})^{3/2}\right) \;\geq\; \ell - \tfrac{3}{2}\log\ell
\end{aligned}$$

■

It remains to prove Equation 4. Fix $x, x' \in HW_{\ell,\ell/2}$, and define $S_{x,x'} \stackrel{\text{def}}{=} \{\pi : \pi(x) = x'\}$. 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 $T_{x,y,x',y'} \stackrel{\text{def}}{=} \{\pi : \pi(x) = x',\ \pi(y) = y'\}$. 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 $\binom{\ell/2}{d/2}^2$. ■