Reusable Garbled Circuits and Succinct Functional Encryption

Shafi Goldwasser* Yael Kalai† Raluca Ada Popa*
Vinod Vaikuntanathan⋈ Nickolai Zeldovich*

* MIT CSAIL † Microsoft Research ⋈ University of Toronto

ABSTRACT

Garbled circuits, introduced by Yao in the mid 80s, allow computing a function f on an input x without leaking anything about f or x besides $f(x)$. Garbled circuits found numerous applications, but every known construction suffers from one limitation: it offers no security if used on multiple inputs x. In this paper, we construct for the first time *reusable* garbled circuits. The key building block is a new *succinct single-key functional encryption* scheme.

Functional encryption is an ambitious primitive: given an encryption $\mathsf{Enc}(x)$ of a value x, and a secret key sk_f for a function f, anyone can compute $f(x)$ without learning any other information about x. We construct, for the first time, a *succinct* functional encryption scheme for *any* polynomial-time function f where succinctness means that the ciphertext size does not grow with the size of the circuit for f, but only with its depth. The security of our construction is based on the intractability of the Learning with Errors (LWE) problem and holds as long as an adversary has access to a *single key* sk_f (or even an a priori bounded number of keys for different functions).

Building on our succinct single-key functional encryption scheme, we show several new applications in addition to reusable garbled circuits, such as a paradigm for general function obfuscation which we call token-based obfuscation, homomorphic encryption for a class of Turing machines where the evaluation runs in input-specific time rather than worst-case time, and a scheme for delegating computation which is publicly verifiable and maintains the privacy of the computation.

Categories and Subject Descriptors: E.3 [**Data Encryption**]
Keywords: Functional encryption; reusable garbled circuits; obfuscation.

1. INTRODUCTION

Breaches of confidential data are commonplace: personal information of millions of people, such as financial, medical, customer, and employee data, is disclosed every year [45, 53]. These disclosures often happen because untrustworthy systems handle confidential data. As applications move to cloud computing platforms, ensuring data confidentiality on third-party servers that may be untrustworthy becomes a top concern [16].

A powerful technique for preventing data disclosures without having to ensure the server is trustworthy is to encrypt the data provided to the server and then compute on the encrypted data. Thus, if the server does not have access to the plaintext or to the decryption key, it will be unable to disclose confidential data. The big leap of the last decade towards computing over encrypted data has been fully homomorphic encryption (FHE) [11–14, 17, 20–22, 38, 51, 52].

A fundamental question with this approach is: *who can decrypt the results of computations on encrypted data?* If data is encrypted using FHE, anyone can perform a computation on it (with knowledge of the public key), while the result of the computation can be decrypted only using the secret key. However, the secret key allows decrypting *all* data encrypted under the corresponding public key. This model suffices for certain applications, but it rules out a large class of applications in which the party computing on the encrypted data needs to determine the computation result on its own. For example, spam filters should be able to determine if an encrypted email is spam and discard it, without learning anything else about the email's content. With FHE, the spam filter can run the spam detection algorithm homomorphically on an encrypted email and obtain an encrypted result; however, it cannot tell if the algorithm deems the email spam or not. Having the data owner provide the decryption key to the spam filter is not a solution: the spam filter can now decrypt all the emails as well!

A promising approach to this problem is *functional encryption* [9, 33–35, 41, 42, 48]. In functional encryption, anyone can encrypt data with a master public key mpk and the holder of the master secret key can provide keys for functions, for example sk_f for function f. Anyone with access to a key sk_f and a ciphertext c for x can obtain the result of the computation in plaintext form: $f(x)$. The security of FE requires that the adversary does not learn anything about x, other than the computation result $f(x)$. It is easy to see, for example, how to solve the above spam filter problem with a functional encryption scheme. A user Alice publishes her public key online and gives the spam filter a key for the filtering function. Users sending email to Alice will encrypt the email with her public key. The spam filter can now determine by itself, for each email, whether to store it in Alice's mailbox or to discard it as spam, without learning anything about Alice's email (except for whether it was deemed spam or not).

The recent impossibility result of Agrawal, Gorbunov, Vaikuntanathan and Wee [2] says that functional encryption schemes where an adversary can receive an arbitrary number of keys for general functions are impossible for a natural simulation-based security definition;[1] stated differently, any functional encryption scheme

STOC'13, June 1–4, 2013, Palo Alto, California, USA.

[1] This impossibility result holds for non-adaptive simulation-based security, which is weaker than some existing simulation-based

that can securely provide q keys for general functions must have ciphertexts growing linearly in q. Since any scheme that can securely provide a single key yields a scheme that can securely provide q keys by repetition, the question becomes if one can construct a functional encryption scheme that can securely provide a *single key for a general function* under this simulation-based security definition. Such a single-key functional encryption scheme is a powerful tool, enabling the applications we will discuss.

In this paper, we construct the first single-key functional encryption scheme for a *general* function that is *succinct*: the size of the ciphertext grows with the depth d of the circuit computing the function and is independent of the size of the circuit. Up until our work, the known constructions of functional encryption were quite limited. First, the works of Boneh and Waters [10], Katz, Sahai and Waters [34], Agrawal, Freeman and Vaikuntanathan [1], and Shen, Shi and Waters [50] show functional encryption schemes (based on different assumptions) for a very simple function: the inner product function f_y (or a variant of it), that on input x outputs 1 if and only if $\langle x, y\rangle = 0$.[2] These works do not shed light on how to extend beyond inner products. Second, Sahai and Seyalioglu [47] and Gorbunov, Vaikuntanathan and Wee [29] provide a construction for single-key functional encryption for one general function with a *non-succinct* ciphertext size (at least the size of a universal circuit computing the functions allowed by the scheme[3]). [47] was the first to introduce the idea of single-key functional encryption and [29] also extends it to allow the adversary to see secret keys for q functions of his choice, by increasing the size of the ciphertexts linearly with q where q is known in advance.[4] We emphasize that the non-succinctness of these schemes is particularly undesirable and it precludes many useful applications of functional encryption (e.g., delegation, reusable garbled circuits, FHE for Turing machines), which we achieve. For example, in the setting of delegation, a data owner wants to delegate her computation to a cloud, but the mere effort of encrypting the data is greater than computing the circuit directly, so the owner is better off doing the computation herself.

We remark that functional encryption (FE) arises from, and generalizes, a beautiful sequence of papers on attribute-based encryption (including [7, 32, 33, 35, 36, 48, 54, 55]), and more generally predicate encryption (including [10, 34, 40]). We denote by attribute-based encryption (ABE) an encryption scheme where each ciphertext c of an underlying plaintext message m is tagged with a public attribute x. Each secret key sk_f is associated with a predicate f. Given a key sk_f and a ciphertext $c = \mathsf{Enc}(x, m)$, the message m can be recovered if and only if $f(x)$ is true. Whether the message gets recovered or not, the attribute x is always public; in other words, the input to the computation of f, x, leaks with attribute-based encryption, whereas with functional encryption, nothing leaks about x other than $f(x)$. Therefore, attribute-based encryption offers qualitatively weaker security than functional encryption. Attribute-based encryption schemes were also called public-index predicate encryption schemes in the literature [9]. Boneh and Waters [10] introduced the idea of not leaking the attribute as in functional encryption (also called private-index functional encryption).

Very recently, the landscape of attribute-based encryption has significantly improved with the works of Gorbunov, Vaikuntanathan and Wee [30], and Sahai and Waters [49], who construct attribute-based encryption schemes for general functions, and are a building block for our results.

1.1 Our Results

Our main result is the construction of a *succinct* single-key functional encryption scheme for *general functions*. We demonstrate the power of this result by showing that it can be used to address the long-standing open problem in cryptography of reusing garbled circuits, as well as making progress on other open problems.

We can state our main result as a reduction from *any* attribute-based encryption and *any* fully homomorphic encryption scheme. In particular, we show how to construct a (single-key and succinct) functional encryption scheme for any class of functions $\mathcal{F}$ by using a homomorphic encryption scheme which can do homomorphic evaluations for any function in $\mathcal{F}$ and an attribute-based encryption scheme for a "slightly larger" class of functions $\mathcal{F}'$; $\mathcal{F}'$ is the class of functions such that for any function $f \in \mathcal{F}$, the class $\mathcal{F}'$ contains the function computing the i-th bit of the FHE evaluation of f.

THEOREM 1.1 (INFORMAL). *There is a single-key functional encryption scheme with succinct ciphertexts (independent of circuit size) for the class of functions $\mathcal{F}$ assuming the existence of*

- *a fully homomorphic encryption scheme for the class of functions $\mathcal{F}$, and*
- *a (single-key) attribute-based encryption scheme for a class of predicates $\mathcal{F}'$ (as above).*

The literature has considered two types of security for ABE and FE: selective and full security. We show that if the underlying ABE scheme is selectively or fully secure, our resulting FE scheme is selectively or fully secure, respectively.

Two very recent results achieve attribute-based encryption for general functions. Gorbunov, Vaikuntanathan and Wee [30] achieve ABE for general circuits of bounded depth based on the subexponential Learning With Errors (LWE) intractability assumption. Sahai and Waters [49] achieve ABE for general circuits under the less standard k-Multilinear Decisional Diffie-Hellman (see [49] for more details); however, when instantiated with the only construction of multilinear maps currently known [18], they also achieve ABE for general circuits of bounded depth. Our scheme can be instantiated with any of these schemes because our result is a reduction.

When coupling our theorem with the ABE result of [30] and the FHE scheme of [12, 13], we obtain:

COROLLARY 1.2 (INFORMAL). *Under the subexponential LWE assumption, for any depth d, there is a single-key functional encryption scheme for general functions computable by circuits of depth d. The scheme has succinct ciphertexts: their size is polynomial in the depth d (and does not depend on the circuit size).*

This corollary holds for both selective and full security definitions, since [30] constructs both selectively secure and fully secure ABE schemes. However, the parameters of the LWE assumption are different in the two cases. For selective security, the LWE assumption reduces to the (polynomial) hardness of approximating shortest vectors in a lattice up to sub-exponential approximation factors. This assumption is known as the gapSVP assumption with sub-exponential approximation factors. For full security, the LWE assumption reduces to the same assumption as above, but where the hardness is assumed to hold even against sub-exponential time adversaries. Namely, the assumption is that it is hard to approximate shortest vectors in a lattice up to sub-exponential approximation

definitions such as adaptive security. Nevertheless, this result does not carry over to indistinguishability-based definitions, for which possibility or impossibility is currently an open question. In this paper, we are interested in achieving the simulation-based definition.

[2]These inner-product schemes allow an arbitrary number of keys.

[3]A universal circuit $\mathcal{F}$ is a circuit that takes as input a description of a circuit f and an input string x, runs f on x and outputs $f(x)$.

[4]Namely, parameter q (the maximum number of keys allowed) is fixed during setup, and the ciphertexts size grows linearly with q.

factors in sub-exponential time. Both of these assumptions are quite standard and well-studied assumptions that are believed to be true. (We refer the reader to our full paper [25] for details.)

Another corollary of our theorem is that, given a *universal* ABE scheme (the scheme is for all classes of circuits, independent of depth) and any fully homomorphic encryption scheme, there is a universal functional encryption scheme whose ciphertext size does not depend on the circuit's size or even the circuit's depth.

As mentioned, extending our scheme to be secure against an adversary who receives q keys is straightforward. The basic idea is simply to repeat the scheme q times in parallel. This strategy results in the ciphertext size growing linearly with q, which is unavoidable for the simulation-based security definition we consider, because of the discussed impossibility result [2]. Stated in these terms, our scheme is also a q-collusion-resistant functional encryption scheme like [29], but our scheme's ciphertexts are succinct, whereas [29]'s are proportional to the circuit size.

From now on, we restrict our attention to the single-key case, which is the essence of the new scheme. In the body of the paper we often omit the single-key or succinct adjectives and whenever we refer to a functional encryption scheme, we mean a succinct single-key functional encryption scheme.

We next show how to use our main theorem to make significant progress on some of the most intriguing open questions in cryptography today: the *reusability* of garbled circuits, a new paradigm for *general function obfuscation*, as well as applications to fully homomorphic encryption with evaluation running in *input-specific time* rather than in worst-case time, and to publicly verifiable delegation. Succinctness plays a central role in these applications and they would not be possible without it.

1.1.1 Main Application: Reusable Garbled Circuits

A circuit garbling scheme, which has been one of the most useful primitives in modern cryptography, is a construction originally suggested by Yao in the 80s in the context of secure two-party computation [57]. This construction relies on the existence of a one-way function to encode an arbitrary circuit C ("garbling" the circuit) and then encode any input x to the circuit (where the size of the encoding is short, namely, it does not grow with the size of the circuit C); a party given the garbling of C and the encoding of x can run the garbled circuit on the encoded x and obtain $C(x)$. The most basic properties of garbled circuits are circuit and input privacy: an adversary learns nothing about the circuit C or the input x other than the result $C(x)$.

Over the years, garbled circuits and variants thereof have found many applications: two party secure protocols [58], multi-party secure protocols [24], one-time programs [27], KDM-security [5], verifiable computation [19], homomorphic computations [23] and others. However, a basic limitation of the original construction remains: it offers only *one-time* usage. Specifically, providing an encoding of more than one input compromises the secrecy of the circuit. Thus, evaluating the circuit C on any new input requires an entirely new garbling of the circuit.

The problem of reusing garbled circuits has been open for 30 years. Using our newly constructed succinct functional encryption scheme we are now able to build *reusable garbled circuits* that achieve *circuit and input privacy*: a garbled circuit for any computation of depth d (where the parameters of the scheme depend on d), which can be run on *any polynomial number* of inputs without compromising the privacy of the circuit or the input. More generally, we prove the following:

THEOREM 1.3 (INFORMAL). *There exists a polynomial p, such that for any depth function d, there is a reusable circuit garbling scheme for the class of all arithmetic circuits of depth d, assuming there is a single-key functional encryption scheme for all arithmetic circuits of depth $p(d)$.*[5]

COROLLARY 1.4 (INFORMAL). *Under the subexponential LWE assumption, for any depth function d, there exists a reusable circuit garbling scheme with circuit and input privacy for all arithmetic circuits of depth d.*

We note that the parameters of this LWE assumption imply its reducibility to the assumption that gapSVP is hard to break in sub-exponential time with sub-exponential approximation factors. (We refer the reader to our full paper [25] for details.)

Reusability of garbled circuits (for depth-bounded computations) implies a multitude of applications as evidenced by the research on garbled circuits over the last 30 years. We note that for many of these applications, depth-bounded computation suffices. We also note that some applications do not require circuit privacy. In that situation, our succinct single-key functional encryption scheme already provides reusable garbled circuits with input-privacy and, moreover, the encoding of the input is a public-key algorithm.

We remark that [30] gives a restricted form of reusable circuit garbling: it provides authenticity of the circuit output, but does not provide input privacy or circuit privacy, as we do here. Informally, authenticity means that an adversary cannot obtain a different yet legitimate result from a garbled circuit. We note that most of the original garbling circuit applications (e.g., two party secure protocols [58], multi-party secure protocols [24]) rely on the privacy of the input or of the circuit.

One of the more intriguing applications of reusable garbled circuits pertains to a new model for program obfuscation, token-based obfuscation, which we discuss next.

1.1.2 Token-Based Obfuscation: a New Way to Circumvent Obfuscation Impossibility Results

Program obfuscation is the process of taking a program as input, and producing a functionally equivalent but different program, so that the new program reveals no information to a computationally bounded adversary about the original program, beyond what "black box access" to the program reveals. Whereas ad-hoc program obfuscators are built routinely, and are used in practice as the main software-based technique to fight reverse engineering of programs, in 2000 Barak et al. [4], followed by Goldwasser and Kalai [26], proved that program obfuscation for general functions is impossible using software alone, with respect to several strong but natural definitions of obfuscation.

The results of [4, 26] mean that there exist functions which cannot be obfuscated. Still, the need to obfuscate or "garble" programs remains. A long array of works attempts to circumvent the impossibility results in various ways, including adding secure hardware components [8, 27, 31], relaxing the definition of security [28], or considering only specific functions [15, 56].

The problem of obfuscation seems intimately related to the "garbled circuit" problem where given a garbling of a circuit C and an encoding for an input x, one can learn the result of $C(x)$ but nothing else. One cannot help but wonder whether the new reusable garbling scheme would *immediately* imply a solution for the obfuscation problem (which we know is impossible). Consider an example illustrating this intuition: a vendor obfuscates her program (circuit) by garbling it and then gives the garbled circuit to a

[5] For this application we need to assume that the underlying functional encryption scheme is fully secure (as opposed to only selectively secure).

customer. In order to run the program on (multiple) inputs x_i, the customer simply encodes the inputs according to the garbling scheme and thus is able to compute $C(x_i)$. Unfortunately, although close, this scenario does not work with reusable garbled circuits. The key observation is that encoding x requires knowledge of a secret key! Thus, an adversary cannot produce encoded inputs on its own, and needs to obtain "tokens" in the form of encrypted inputs from the data owner.

Instead, we propose a new *token-based* model for obfuscation. The idea is for a vendor to obfuscate an arbitrary program as well as provide tokens representing rights to run this program on specific inputs. For example, consider that some researchers want to obtain statistics out of an obfuscated database containing sensitive information (the obfuscated program is the program running queries with the secret database hardcoded in it). Whenever the researchers want to input a query x to this program, they need to obtain a token for x from the program owner. To produce each token, the program owner does little work. The researchers perform the bulk of the computation by themselves using the token and obtain the computation result without further interaction with the owner.

CLAIM 1.5. *Assuming a reusable garbling scheme for a class of circuits, there is a token-based obfuscation scheme for the same class of circuits.*

COROLLARY 1.6 (INFORMAL). *Under the subexponential LWE assumption, for any depth function d, there exists a token-based obfuscation scheme for all arithmetic circuits of depth d.*

It is worthwhile to compare the token-based obfuscation model with previous work addressing obfuscation using trusted-hardware components such as [8, 31]. In these schemes, after a user finishes executing the obfuscated program on an input, the user needs to interact with the trusted hardware to obtain the decryption of the result; in comparison, in our scheme, the user needs to obtain only a token before the computation begins, and can then run the computation and obtain the decrypted result by herself.

1.1.3 Computing on Encrypted Data in Input-Specific Time

All current FHE constructions work according to the following template. For a fixed input size, a program is transformed into an arithmetic circuit; homomorphic evaluation happens gate by gate on this circuit. The size of the circuit reflects the *worst-case* running time of the program: for example, every loop is unfolded into the maximum number of steps corresponding to the worst-case input, and each function is called the maximum number of times possible. Such a circuit can be potentially very large, despite the fact that there could be many inputs on which the execution is short.

A fascinating open question has been whether it is possible to perform FHE following a Turing-machine-like template: *the computation time is input-specific* and can terminate earlier depending on the input at hand. Of course, to compute in input-specific time, the running time must unavoidably leak to the evaluator, but such leakage is acceptable in certain applications and the efficiency gains can be significant; therefore, such a scheme provides weaker security than fully homomorphic encryption (namely, nothing other than the running time leaks about the input), at the increase of efficiency.

Using our functional encryption scheme, we show how to achieve this goal. The idea is to use the scheme to test when an encrypted circuit computation has terminated, so the computation can stop earlier on certain inputs. We overview our technique in Sec. 1.2.

Because the ciphertexts in our functional encryption scheme grow with the depth of the circuits, such a scheme is useful only for Turing machines that can be expressed as circuits of depth at most $d(n)$ for inputs of size n. We refer to such Turing machines as *d-depth-bounded* (see our full paper [25] for details).

THEOREM 1.7. *There is a scheme for evaluating Turing machines on encrypted inputs in input-specific time for any class of d-depth-bounded Turing machines, assuming the existence of a succinct single-key functional encryption scheme for circuits of depth d,[6] and a fully homomorphic encryption scheme for circuits of depth d.*

COROLLARY 1.8 (INFORMAL). *Under the subexponential LWE assumption, for any depth d, there is a scheme for evaluating Turing machines on encrypted data in input-specific time for any class of d-depth-bounded Turing machines.*

The parameters of this LWE assumption are the same as discussed in Corollary 1.2.

1.1.4 Publicly-Verifiable Delegation with Secrecy

Recently, Parno, Raykova and Vaikuntanathan [43] showed how to construct a 2-message delegation scheme that is *publicly verifiable*, in the preprocessing model, from any attribute-based encryption scheme. This reduction can be combined with [30]'s ABE scheme to achieve such a delegation scheme.

However, this scheme does not provide *secrecy* of the inputs: the prover can learn the inputs. By replacing the ABE scheme in the construction of [43] with our new functional encryption scheme, we add secrecy to the scheme; namely, we obtain a delegation scheme which is both *publicly verifiable* as in [43] (anyone can verify that a transcript is accepting using only public information) and *secret* (the prover does not learn anything about the input of the function being delegated).[7] More specifically, we construct a 2-message delegation scheme in the preprocessing model that is based on the subexponential LWE assumption, and is for general depth-bounded circuits, where the verifier works in time that depends on the *depth* of the circuit being delegated, but is independent of the size of the circuit, and the prover works in time dependent on the *size* of the circuit. For more details, see our full paper [25].

1.2 Technique Outline

Our functional encryption scheme. We first describe the ideas behind our main technical result: a reduction from attribute-based encryption (ABE) and fully homomorphic encryption (FHE) to functional encryption (FE).

Compute on encrypted data with FHE. A natural starting point is FHE because it enables computation on encrypted data, which is needed with functional encryption. Using FHE, the FE encryption of an input x consists of an FHE encryption of x, denoted $\hat{x}$, while the secret key for a function f is simply f itself. The semantic security of FHE provides the desired security (and more) because nothing leaks about x; however, using FHE evaluation, the evaluator obtains an encrypted computation result, $\widehat{f(x)}$, instead of the decrypted value $f(x)$. Giving the evaluator the FHE decryption key is not an option because the evaluator can use it to decrypt x as well.

Attempt to decrypt using a Yao garbled circuit. We would like the evaluator to decrypt the FHE ciphertext $\widehat{f(x)}$, but not be able to decrypt anything else. An idea is for the owner to give the evaluator a

[6] As in previous applications, we need to assume that the underlying functional encryption scheme is fully secure (as opposed to only selectively secure).

[7] We note that secrecy can be easily obtained by using an FHE scheme, however, this destroys public-verifiability.

Yao garbled circuit for the FHE decryption function $\mathsf{FHE.Dec}$ with the FHE secret key hsk hardcoded in it, namely a garbled circuit for $\mathsf{FHE.Dec}_{\mathsf{hsk}}$. When the owner garbles $\mathsf{FHE.Dec}_{\mathsf{hsk}}$, the owner also obtains a set of garbled circuit labels $\{L_0^i, L_1^i\}_i$. The evaluator must only receive the input labels corresponding to $\widehat{f(x)}$: namely, the labels $\{L_{b_i}^i\}_i$ where b_i is the i-th bit of $\widehat{f(x)}$. But this is not possible because the owner does not know a priori $\widehat{f(x)}$ which is determined only after the FHE evaluation; furthermore, after providing more than one set of labels (which happens when encrypting another input x'), the security of the garbled circuit (and hence of the FHE secret key) is compromised. One idea is to have the owner and the evaluator interact, but the syntax of functional encryption does not allow interaction. Therefore, the evaluator needs to determine the set of labels corresponding to $\widehat{f(x)}$ *by herself*, and *should not obtain any other labels*.

Constraining decryption using ABE. It turns out that what we need here is very close to what ABE provides. Consider the following variant of ABE that can be constructed easily from a standard ABE scheme. One encrypts a value y together with two messages m_0, m_1 and obtains a ciphertext $c \leftarrow \mathsf{ABE.Enc}(y, m_0, m_1)$. Then, one generates a key for a predicate g: $\mathsf{sk}_g \leftarrow \mathsf{ABE.KeyGen}(g)$. The decryption algorithm on input c and sk_g outputs m_0 if $g(y) = 0$ or outputs m_1 if $g(y) = 1$.

Now consider using this ABE variant multiple times, once for every $i \in \{1, \ldots, \text{size of } \widehat{f(x)}\}$. For the i-th invocation of $\mathsf{ABE.Enc}$, let m_0, m_1 be the garbled labels L_0^i, L_1^i, and let y be $\hat{x}$: $\mathsf{ABE.Enc}(\hat{x}, L_0^i, L_1^i)$. Next, for the i-th invocation of $\mathsf{ABE.KeyGen}$, let g be $\mathsf{FHE.Eval}_f^i$ (the predicate returning the i-th bit of the evaluation of f on an input ciphertext): $\mathsf{ABE.KeyGen}(\mathsf{FHE.Eval}_f^i)$. Then, the evaluator can use $\mathsf{ABE.Dec}$ to obtain the needed label: $L_{b_i}^i$ where b_i is the i-th bit of $\widehat{f(x)}$. Armed with these labels and the garbled circuit, the evaluator decrypts $f(x)$.

The security of the ABE scheme ensures the evaluator cannot decrypt any other labels, so the evaluator cannot learn more than $f(x)$. Finally, note that the one-time aspect of garbled circuits does not restrict the number of encryptions with our FE scheme because the encryption algorithm generates a new garbled circuit every time; since the garbled circuit is for the FHE decryption algorithm (which is a fixed algorithm), the size of the ciphertexts remains independent of the size of f.

From FE to reusable garbled circuits. The goal of garbled circuits is to hide the input and the circuit C. Our succinct single-key FE already provides a reusable garbling scheme with input privacy (the single key corresponds to the circuit to garble). To obtain circuit privacy, the insight is to leverage the secrecy of the inputs to hide the circuit. The first idea that comes to mind is to generate a key for the universal circuit instead of C, and include C in the ciphertext when encrypting an input. However, this approach will yield large ciphertexts, as large as the circuit size.

Instead, the insight is to garble C by using a semantically secure encryption scheme $\mathsf{E.Enc}$ together with our FE scheme: the garbling of C will be an FE secret key for a circuit U that contains $\mathsf{E.Enc}_{\mathsf{sk}}(C)$; on input (sk, x), U uses sk to decrypt C and then runs C on the input x. The token for an input x will be an FE encryption of (sk, x). Now, even if the FE scheme does not hide $\mathsf{E.Enc}_{\mathsf{sk}}(C)$, the security of the encryption scheme E hides C.

Computing on encrypted data in input-specific time. We now summarize our approach to evaluating a Turing machine (TM) M homomorphically over encrypted data without running in worst-case time on all inputs. We refer the reader to our full paper [25] for a formal presentation.

Our idea is to use our functional encryption scheme to enable the evaluator to determine at various intermediary steps in the evaluation whether the computation finished or not. For each intermediary step, the client provides a secret key for a function that returns a bit indicating whether the computation finished or not. However, if the client provides a key for every computation step, then the amount of keys corresponds to the worst-case running time. Thus, instead, we choose intermediary points spaced at exponentially increasing intervals. In this way, the client generates only a logarithmic number of keys, namely for functions indicating if the computation finishes in $1, 2, 4, \ldots, 2^i, \ldots, 2^{\lceil \log t_{\mathsf{max}} \rceil}$ steps, where t_{max} is the worst-case running time of M on all inputs of a certain size.

Because of the single-key aspect of our FE scheme, the client cannot provide keys for an arbitrary number of TMs to the evaluator. However, this does not mean that the evaluator can run only an a priori fixed number of TMs on the encrypted data. The reason is that the client can provide keys for the universal TMs $U_0, \ldots, U_{\lceil \log t_{\mathsf{max}} \rceil}$, where TM U_i is the TM that on input a TM M and a value x, runs M on x for 2^i steps and outputs whether M finished.

Therefore, in an offline preprocessing phase, the client provides $1 + \lceil \log t_{\mathsf{max}} \rceil$ keys where the i-th key is for a circuit corresponding to U_i, each being generated with a different master secret key. The work of the client in this phase is at least t_{max} which is costly, but this work happens only once and is amortized over all subsequent inputs in the online phase.

In an online phase, the client receives an input x and wants the evaluator to compute $M(x)$ for her. The client provides FE encryptions of (M, x) to the evaluator together with an FHE ciphertext $(\hat{M}, \hat{x})$ for (M, x) to be used for a separate FHE evaluation. The evaluator tries each key sk_{U_i} from the preprocessing phase and learns the smallest i for which the computation of M on x stops in 2^i steps. The evaluator then computes a universal circuit of size $\tilde{O}(2^i)$ and evaluates it homomorphically over $(\hat{M}, \hat{x})$, obtaining the FHE encryption of $M(x)$. Thus, we can see that the evaluator runs in time polynomial in the runtime of M on x.

2. PRELIMINARIES

Let κ denote the security parameter throughout this paper. For a distribution $\mathcal{D}$, we say $x \leftarrow \mathcal{D}$ when x is sampled from the distribution $\mathcal{D}$. If S is a finite set, by $x \leftarrow S$ we mean x is sampled from the uniform distribution over the set S. Let $[n]$ denote the set $\{1, \ldots, n\}$ for $n \in \mathbb{N}^*$. When saying that a Turing machine A is p.p.t. we mean that A is a non-uniform probabilistic polynomial-time machine. In this paper, we only consider boolean arithmetic circuits, that is, circuits formed of $+$ and $\times$ gates mod 2 with one bit of output.

2.1 Building Blocks

We present the building blocks that our construction relies on. We provide only informal definitions and theorems here, and refer the reader to our full paper [25] for their formal counterparts.

The LWE assumption. The security of our results will be based on the Learning with Errors (LWE) assumption, first introduced by Regev [46]. Regev showed that solving the LWE problem *on average* is (quantumly) as hard as solving the approximate version of several standard lattice problems, such as gapSVP *in the worst case*. Peikert [44] later removed the quantum assumption from a variant of this reduction. Given this connection, we state all our results under worst-case lattice assumptions, and in particular, under (a variant of) the gapSVP assumption. We refer the reader to [44, 46] for details about the worst-case/average-case connection.

The best known algorithms to solve these lattice problems with

an approximation factor 2^{ℓ^ϵ} in ℓ-dimensional lattices run in time $2^{\tilde{O}(\ell^{1-\epsilon})}$ [3, 39] for any constant $0 < \epsilon < 1$. Specifically, given the current state-of-the-art on lattice algorithms, it is quite plausible that achieving approximation factors 2^{ℓ^ϵ} for these lattice problems is hard for polynomial time algorithms.

FHE. Fully homomorphic encryption enables an evaluator to compute on encrypted data without learning anything about the underlying data. Formally, a $\mathcal{C}$-homomorphic encryption scheme FHE for a class of circuits $\mathcal{C}$ is a tuple of polynomial-time algorithms $(\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$. The key generation algorithm $\mathsf{FHE.KeyGen}(1^\kappa)$ takes as input the security parameter 1^κ and outputs a public/secret key pair $(\mathsf{hpk}, \mathsf{hsk})$. The encryption algorithm $\mathsf{FHE.Enc}$ $(\mathsf{hpk}, x \in \{0,1\})$ takes as input the public key hpk and a bit x and outputs a ciphertext ψ, whereas the decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \psi)$ takes as input the secret key hsk and a ciphertext ψ and outputs a decrypted bit. The homomorphic evaluation algorithm $\mathsf{FHE.Eval}(\mathsf{hpk}, C, \psi_1, \psi_2, \ldots, \psi_n)$ takes as input the public key hpk, n ciphertexts $\psi_1, \ldots, \psi_n$ (which are encryptions of bits $x_1, \ldots, x_n$) and a circuit $C \in \mathcal{C}$ that takes n bits as input. It outputs a ciphertext ψ_C which decrypts to $C(x_1, \ldots, x_n)$. The security definition is semantic security (or IND-CPA).

A fully homomorphic encryption scheme is homomorphic for the class of all polynomial-sized circuits. A special type of homomorphic encryption, called leveled fully homomorphic encryption, suffices for our purposes: in a d-leveled FHE scheme, $\mathsf{FHE.KeyGen}$ takes an additional input 1^d and the resulting scheme is homomorphic for all depth-d arithmetic circuits over $\mathsf{GF}(2)$.

THEOREM 2.1 ([12, 13]). *Assume that there is a constant $0 < \epsilon < 1$ such that for every sufficiently large ℓ, the approximate shortest vector problem gapSVP in ℓ dimensions is hard to approximate to within a $2^{O(\ell^\epsilon)}$ factor in the worst case. Then, for every n and every polynomial $d = d(n)$, there is an IND-CPA secure d-leveled fully homomorphic encryption scheme where encrypting n bits produces ciphertexts of length $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$, the size of the circuit for homomorphic evaluation of a function f is $\mathsf{size}(C_f) \cdot \mathrm{poly}(n, \kappa, d^{1/\epsilon})$ and its depth is $\mathsf{depth}(C_f) \cdot \mathrm{poly}(\log n, \log d)$.*

Garbled circuits. Garbled circuits were initially presented by Yao [57], then proven secure by Lindell and Pinkas [37], and recently formalized by Bellare et al. [6].

A garbling scheme for a family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where $\mathcal{C}_n$ is a set of boolean circuits taking n-bit inputs, is a tuple of p.p.t. algorithms $\mathsf{Gb} = (\mathsf{Gb.Garble}, \mathsf{Gb.Enc}, \mathsf{Gb.Eval})$ such that $\mathsf{Gb.Garble}(1^\kappa, C)$ takes as input the security parameter κ and a circuit $C \in \mathcal{C}_n$ for some n, and outputs the garbled circuit Γ and a secret key sk; $\mathsf{Gb.Enc}(\mathsf{sk}, x)$ takes as input $x \in \{0,1\}^*$ and outputs an encoding c whose size must not depend on the size of the circuit C; and $\mathsf{Gb.Eval}(\Gamma, c)$ takes as input a garbled circuit Γ and an encoding c, and outputs a value y that must be equal to $C(x)$.

The garbling scheme presented by Yao has a specific property that is useful in various secure function evaluation (SFE) protocols and in our construction as well. The secret key is of the form $\mathsf{sk} = \{L_i^0, L_i^1\}_{i=1}^n$ and the encoding of an n-bit input x is of the form $c = (L_1^{x_1}, \ldots, L_n^{x_n})$ where x_i is the i-th bit of x.

Two security guarantees are of interest: input privacy (the input to the garbled circuit does not leak to the adversary) and circuit privacy (the circuit does not leak to the adversary). In all known garbling schemes, these properties hold *only for one-time* evaluation of the circuit: the adversary can receive at most one encoding of an input to use with a garbled circuit; obtaining more than one encoding breaks these security guarantees. More formally, the security definition states that there exists a p.p.t. simulator $\mathsf{Sim}_{\mathsf{Garble}}$ that given the result $C(x)$ of a (secret) circuit C on a *single* (secret) input x, and given the sizes of C and x (but not the actual values of C and x), outputs a simulated garbled circuit $\tilde{\Gamma}$ and an encoding $\tilde{c}$, $(\tilde{\Gamma}, \tilde{c}) \leftarrow \mathsf{Sim}_{\mathsf{Garble}}(1^\kappa, C(x), 1^{|C|}, 1^{|x|})$, that are computationally indistinguishable from the real garbled circuit Γ and encoding c.

THEOREM 2.2 ([37, 57]). *Assuming one-way functions exist, there exists a Yao (one-time) garbling scheme that is input- and circuit-private for all circuits over GF(2).*

2.2 Attribute-Based Encryption (ABE)

Attribute-based encryption is an important component of our construction. We present a slight (but equivalent) variant of ABE that better serves our goal.

Definition 1. An attribute-based encryption scheme (ABE) for a class of predicates $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ represented as circuits with n input bits, and a message space $\mathcal{M}$ is a tuple of algorithms $(\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ as follows:

- $\mathsf{ABE.Setup}(1^\kappa)$: Takes as input a security parameter 1^κ and outputs a public master key fmpk and a master secret key fmsk.
- $\mathsf{ABE.KeyGen}(\mathsf{fmsk}, P)$: Given a master secret key fmsk and a predicate $P \in \mathcal{P}$, outputs a key fsk_P corresponding to P.
- $\mathsf{ABE.Enc}(\mathsf{fmpk}, x, M_0, M_1)$: Takes as input the public key fmpk, an attribute $x \in \{0,1\}^n$, for some n, and two messages $M_0, M_1 \in \mathcal{M}$ and outputs a ciphertext c.
- $\mathsf{ABE.Dec}(\mathsf{fsk}_P, c)$: Takes as input a secret key for a predicate and a ciphertext and outputs $M^* \in \mathcal{M}$.

Correctness. For any polynomial $n(\cdot)$, for every sufficiently large security parameter κ, if $n = n(\kappa)$, for all predicates $P \in \mathcal{P}_n$, attributes $x \in \{0,1\}^n$, messages $M_0, M_1 \in \mathcal{M}$:

$$\Pr\left[\begin{array}{l}(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{ABE.Setup}(1^\kappa);\\ \mathsf{fsk}_P \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}, P);\\ c \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}, x, M_0, M_1);\\ M^* \leftarrow \mathsf{ABE.Dec}(\mathsf{fsk}_P, c) : M^* = M_{P(x)}\end{array}\right] = 1 - \mathrm{negl}(\kappa).$$

Informally, the security of ABE guarantees that nothing leaks about M_0 if $P(x) = 1$ and nothing leaks about M_1 if $P(x) = 0$. However, the scheme does not hide the attribute x, and x may leak no matter what $P(x)$ is. The security of ABE is thus conceptually weaker than the security for FE: the input that the computation happens on leaks with ABE, while this input does not leak with FE.

We call an ABE or FE scheme for circuits of depth d a *d-leveled* ABE or *d-leveled* FE scheme, respectively.

THEOREM 2.3 ([30]). *Assume there is a constant $0 < \epsilon < 1$ such that for every sufficiently large ℓ, the approximate shortest vector problem gapSVP in ℓ dimensions is hard to approximate to within a $2^{O(\ell^\epsilon)}$ factor in $\mathrm{poly}(\ell)$ (resp. 2^{ℓ^ϵ}) time. Then, for every n and every polynomial $d = d(n)$, there is a selectively (resp. fully) secure d-leveled attribute-based encryption scheme where encrypting n bits produces ciphertexts of length $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$ (resp. $\mathsf{poly}(n, \kappa, d^{1/\epsilon^2})$).*

2.3 Functional Encryption (FE)

We recall the functional encryption definition from the literature [9, 29, 34] with some notational changes.

Definition 2. A functional encryption scheme FE for a class of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ represented as boolean circuits with an n-bit input, is a tuple of four p.p.t. algorithms $(\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ such that:

- $\mathsf{FE.Setup}(1^\kappa)$ takes as input the security parameter 1^κ and outputs a master public key fmpk and a master secret key fmsk.
- $\mathsf{FE.KeyGen}(\mathsf{fmsk}, f)$ takes as input the master secret key fmsk and a function $f \in \mathcal{F}$ and outputs a key fsk_f.
- $\mathsf{FE.Enc}(\mathsf{fmpk}, x)$ takes as input the master public key fmpk and an input $x \in \{0,1\}^*$ and outputs a ciphertext c.
- $\mathsf{FE.Dec}(\mathsf{fsk}_f, c)$ takes as input a key fsk_f and a ciphertext c and outputs a value y.

Correctness. For any polynomial $n(\cdot)$, for every sufficiently large security parameter κ, for $n = n(\kappa)$, for all $f \in \mathcal{F}_n$, and all $x \in \{0,1\}^n$,

$$\Pr[(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa); \mathsf{fsk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, f); c \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, x) : \mathsf{FE.Dec}(\mathsf{fsk}_f, c) = f(x)] = 1 - \mathrm{negl}(\kappa).$$

Intuitively, the security of functional encryption requires that an adversary should not learn anything about the input x other than the computation result $C(x)$, for some circuit C for which a key was issued. In this paper, we present only the definition of full security and defer the definition of selective security to our full paper [25]. The security definition states that whatever information an adversary is able to learn from the ciphertext and the function keys can be simulated given only the function keys and the output of the function on the inputs.

Definition 3. (FE Security) Let FE be a functional encryption scheme for the family of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$. For every p.p.t. adversary $A = (A_1, A_2)$ and p.p.t. simulator S, consider the following two experiments:

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},A,S}(1^\kappa)$:
$(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$	
$(f, \mathsf{state}_A) \leftarrow A_1(\mathsf{fmpk})$	
$\mathsf{fsk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, f)$	
$(x, \mathsf{state}'_A) \leftarrow A_2(\mathsf{state}_A, \mathsf{fsk}_f)$	
$c \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, x)$	$\tilde{c} \leftarrow S(\mathsf{fmpk}, \mathsf{fsk}_f, f, f(x), 1^{\lvert x\rvert})$
Output (state'_A, c)	Output $(\mathsf{state}'_A, \tilde{c})$

The scheme is said to be (single-key) $\mathsf{FULL\text{-}SIM}$−secure if there exists a p.p.t. simulator S such that for all pairs of p.p.t. adversaries (A_1, A_2), the outcomes of the two experiments are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},A,S}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}.$$

3. OUR FUNCTIONAL ENCRYPTION

In this section, we present our main result: the construction of a functional encryption scheme FE. We refer the reader to the introduction (Sec. 1.2) for an overview of our approach, and we proceed directly with the construction here.

We use three building blocks in our construction: a (leveled) fully homomorphic encryption scheme FHE, a (leveled) attribute-based encryption scheme ABE, and a Yao garbling scheme Gb.

For simplicity, we construct FE for functions outputting one bit; functions with larger outputs can be handled by repeating our scheme below for every output bit. Let $\lambda = \lambda(\kappa)$ be the length of the ciphertexts in the FHE scheme (both from encryption and evaluation). The construction of $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ proceeds as follows.

Setup $\mathsf{FE.Setup}(1^\kappa)$**:** Run the setup algorithm for the ABE scheme λ times:

$$(\mathsf{fmpk}_i, \mathsf{fmsk}_i) \leftarrow \mathsf{ABE.Setup}(1^\kappa) \text{ for } i \in [\lambda].$$

Output as master public key and secret key:

$$\mathsf{MPK} = (\mathsf{fmpk}_1, \ldots, \mathsf{fmpk}_\lambda) \text{ and } \mathsf{MSK} = (\mathsf{fmsk}_1, \ldots, \mathsf{fmsk}_\lambda).$$

Key Generation $\mathsf{FE.KeyGen}(\mathsf{MSK}, f)$**:** Let n be the number of bits in the input to the circuit f. If hpk is an FHE public key and $\psi_1, \ldots, \psi_n$ are FHE ciphertexts, let $\mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi_1, \ldots, \psi_n)$ be the i-th bit of the homomorphic evaluation of f on $\psi_1, \ldots, \psi_n$ ($\mathsf{FHE.Eval}(\mathsf{hpk}, f, \psi_1, \ldots, \psi_n)$), where $i \in [\lambda]$. Thus, $\mathsf{FHE.Eval}^i_f : \{0,1\}^{|\mathsf{hpk}|} \times \{0,1\}^{n\lambda} \to \{0,1\}$.

1. Run the key generation algorithm of ABE for the functions $\mathsf{FHE.Eval}^i_f$ (under the different master secret keys) to construct secret keys:
$$\mathsf{fsk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}_i, \mathsf{FHE.Eval}^i_f) \text{ for } i \in [\lambda].$$
2. Output the tuple $\mathsf{fsk}_f := (\mathsf{fsk}_1, \ldots, \mathsf{fsk}_\lambda)$ as the secret key for the function f.

Encryption $\mathsf{FE.Enc}(\mathsf{MPK}, x)$**:** Let n be the number of bits of x, namely $x = x_1 \ldots x_n$. Encryption proceeds in three steps.

1. Generate a fresh key pair $(\mathsf{hpk}, \mathsf{hsk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$ for the (leveled) fully homomorphic encryption scheme. Encrypt each bit of x homomorphically: $\psi_i \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, x_i)$. Let $\psi := (\psi_1, \ldots, \psi_n)$ be the encryption of the input x.
2. Run the Yao garbled circuit generation algorithm to produce a garbled circuit for the FHE decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$ together with 2λ labels L^b_i for $i \in [\lambda]$ and $b \in \{0,1\}$. Namely,
$$\left(\Gamma, \{L^0_i, L^1_i\}^\lambda_{i=1}\right) \leftarrow \mathsf{Gb.Garble}(1^\kappa, \mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)),$$
where Γ is the garbled circuit and the L^b_i are the input labels.
3. Produce encryptions $c_1, \ldots, c_\lambda$ using the ABE scheme:
$$c_i \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), L^0_i, L^1_i\right) \text{ for } i \in [\lambda],$$
where (hpk, ψ) comes from the first step, and the labels (L^0_i, L^1_i) come from the second step.
4. Output the ciphertext $c = (c_1, \ldots, c_\lambda, \Gamma)$.

Decryption $\mathsf{FE.Dec}(\mathsf{fsk}_f, c)$**:**

1. Run the ABE decryption algorithm on the ciphertexts $c_1, \ldots, c_\lambda$ to recover the labels for the garbled circuit. In particular, let
$$L^{d_i}_i \leftarrow \mathsf{ABE.Dec}(\mathsf{fsk}_i, c_i) \text{ for } i \in [\lambda],$$
where d_i is equal to $\mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi)$.
2. Now, armed with the garbled circuit Γ and the labels $L^{d_i}_i$, run the garbled circuit evaluation algorithm to compute

$$\mathsf{Gb.Eval}(\Gamma, L^{d_1}_1, \ldots, L^{d_\lambda}_\lambda) = \mathsf{FHE.Dec}(\mathsf{hsk}, d_1 d_2 \ldots d_\lambda) = f(x).$$

We now provide a proof for our main Theorem 1.1, delegating certain details to the full paper [25].

PROOF OF THEOREM 1.1. Let us first argue that the scheme is correct. We examine the values we obtain in $\mathsf{FE.Dec}(\mathsf{fsk}_f, c)$. In Step (1), by the correctness of the ABE scheme used, d_i is $\mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi)$: $\mathsf{FHE.Eval}^i_f$ comes from fsk_f and (hpk, ψ) come from c_i. Therefore, the inputs to the garbled circuit Γ in Step (2) are the set of λ labels corresponding to the value of

$\mathsf{FHE.Eval}_f(\mathsf{hpk}, \psi)$. By the correctness of the FHE scheme, this value corresponds to an FHE encryption of $f(x)$. Finally, by the correctness of the garbled circuit scheme, and by how Γ was constructed in FE.Enc, the FHE ciphertext gets decrypted by Γ correctly, yielding $f(x)$ as the output from FE.Dec.

We now prove the succinctness property—namely, that the size of FE ciphertexts is independent of the size of the circuit. FE's ciphertext is the output of FE.Enc, which outputs λ ciphertexts from ABE.Enc and a garbled circuit from Gb.Garble. These add up as follows. First, $\lambda = \mathsf{ctsize}_{\mathsf{FHE}}$, the size of the ciphertext in FHE. Second, we denote the size of the ciphertext produced by ABE.Enc as $\mathsf{ctsize}_{\mathsf{ABE}}(\cdot)$, which is a function of ABE.Enc's input size. The input provided by FE.Enc to ABE.Enc consists of $\mathsf{pksize}_{\mathsf{FHE}}$ bits for hpk, $n \cdot \mathsf{ctsize}_{\mathsf{FHE}}$ bits for ψ, and $\mathrm{poly}(\kappa)$ bits for the labels. Finally, the garbled circuit size is polynomial in the size of the input circuit passed to Gb.Garble, which in turn is polynomial in $\mathsf{sksize}_{\mathsf{FHE}}$ and $\mathsf{ctsize}_{\mathsf{FHE}}$. Thus, we obtain the overall ciphertext size of FE: $\mathsf{ctsize}_{\mathsf{FE}} = \mathsf{ctsize}_{\mathsf{FHE}} \cdot \mathsf{ctsize}_{\mathsf{ABE}}(\mathsf{pksize}_{\mathsf{FHE}} + n \cdot \mathsf{ctsize}_{\mathsf{FHE}} + \mathrm{poly}(\kappa)) + \mathrm{poly}(\kappa, \mathsf{sksize}_{\mathsf{FHE}}, \mathsf{ctsize}_{\mathsf{FHE}})$. We can thus see that if FHE and ABE produce ciphertexts and public keys independent of the circuit size, then so will our functional encryption scheme.

Finally, we prove security of our scheme based on Def. 3. We construct a p.p.t. simulator S that achieves Def. 3. S receives as input $(\mathsf{MPK}, \mathsf{fsk}_f, f, f(x), 1^n)$ and must output $\tilde{c}$ such that the real and ideal experiments in Def. 3 are computationally indistinguishable. Intuitively, S runs a modified version of FE.Enc to mask the fact that it does not know x.

Simulator S on input $(\mathsf{MPK}, \mathsf{fsk}_f, f, f(x), 1^n)$:

1. Choose a key pair $(\mathsf{hpk}, \mathsf{hsk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$ for the homomorphic encryption scheme (where S can derive the security parameter κ from the sizes of the inputs it gets). Encrypt 0^n (n zero bits) with FHE by encrypting each bit individually and denote the ciphertext $\hat{0} := (\hat{0}_1 \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, 0), \ldots, \hat{0}_n \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, 0))$.
2. Let $\mathsf{Sim}_{\mathsf{Garble}}$ be the simulator for the Yao garbling scheme (described in Sec. 2.1) for the class of circuits corresponding to $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$. Run $\mathsf{Sim}_{\mathsf{Garble}}$ to produce a simulated garbled circuit $\tilde{\Gamma}$ for the FHE decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\lambda \rightarrow \{0,1\}$ together with the simulated encoding consisting of one set of λ labels $\tilde{L}_i$ for $i = 1 \ldots \lambda$. Namely,
$$\left(\tilde{\Gamma}, \{\tilde{L}_i\}_{i=1}^{\lambda}\right) \leftarrow \mathsf{Sim}_{\mathsf{Garble}}(1^\kappa, f(x), 1^{|\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)|}, 1^\lambda).$$
The simulator S can invoke $\mathsf{Sim}_{\mathsf{Garble}}$ because it knows $f(x)$, and can compute the size of the $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$ circuit, and λ from the sizes of the input parameters.
3. Produce encryptions $\tilde{c}_1, \ldots, \tilde{c}_\lambda$ under the ABE scheme in the following way. Let
$$\tilde{c}_i \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \hat{0}), \tilde{L}_i, \tilde{L}_i\right),$$
where S uses each simulated label $\tilde{L}_i$ twice.
4. Output $\tilde{c} = (\tilde{c}_1, \ldots, \tilde{c}_\lambda, \tilde{\Gamma})$.

To prove indistinguishability of the real and ideal experiments (Def. 3), we define a sequence of hybrid experiments, and then invoke the security definitions of the underlying schemes (FHE, garbled circuit, and ABE respectively) to show that the outcome of the hybrid experiments are computationally indistinguishable.

Hybrid 0 is the output of the ideal experiment from Def. 3 for our FE construction with simulator S.

Hybrid 1 is the same as Hybrid 0, except that the simulated ciphertext for Hybrid 1 (which we denote $\tilde{c}^{(1)}$), changes. Let $\tilde{c}^{(1)}$ be the ciphertext obtained by running the algorithm of S, except that in Step (3), encrypt x instead of 0, namely:
$$\tilde{c}_i^{(1)} \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), \tilde{L}_i, \tilde{L}_i\right),$$
where $\psi \leftarrow (\mathsf{FHE.Enc}(\mathsf{hpk}, x_1), \ldots, \mathsf{FHE.Enc}(\mathsf{hpk}, x_n))$. Let
$$\tilde{c}^{(1)} = (\tilde{c}_1^{(1)}, \ldots, \tilde{c}_\lambda^{(1)}, \tilde{\Gamma}).$$

Hybrid 2 is the same as Hybrid 1, except that in Step (2), the ciphertext contains a real garbled circuit
$$\left(\Gamma, \{L_i^0, L_i^1\}_{i=1}^{\lambda}\right) \leftarrow \mathsf{Gb.Garble}(\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)).$$
Let $d_i = \mathsf{FHE.Eval}_f^i(\mathsf{hpk}, \psi)$. In Step (3), include L^{d_i} twice in the ABE encryption; namely:
$$\tilde{c}_i^{(2)} \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), L_i^{d_i}, L_i^{d_i}\right), \text{ and}$$
$$\tilde{c}^{(2)} = (\tilde{c}_1^{(2)}, \ldots, \tilde{c}_\lambda^{(2)}, \Gamma).$$

Hybrid 3 is the output of the real experiment from Def. 3 for our FE construction.

In our full paper [25], we prove that each pair of consecutive hybrids are computationally indistinguishable: Hybrid 0 and Hybrid 1 by the security of the homomorphic scheme FHE, Hybrid 1 and Hybrid 2 by the security of the garbled circuit scheme Gb, and Hybrid 2 and Hybrid 3 by the security of the ABE scheme ABE, thus completing our proof. □

Instantiating the components with the leveled fully homomorphic encryption scheme of [13] (see Theorem 2.1), the leveled attribute-based encryption scheme of [30] (see Theorem 2.3) and Yao garbled circuit from one-way functions (see Theorem 2.2), we get the following corollary of Theorem 1.1:

COROLLARY 3.1 (THE LWE INSTANTIATION). *Assume that there is a constant $0 < \epsilon < 1$ such that for every sufficiently large ℓ, the approximate shortest vector problem* gapSVP *in ℓ dimensions is hard to approximate to within a $2^{O(\ell^\epsilon)}$ factor in the worst case in time $\mathrm{poly}(\ell)$ (resp. 2^{ℓ^ϵ}) time. Then, for every n and every polynomial $d = d(n)$, there is a selectively secure (resp. fully secure) functional encryption scheme for depth d circuits, where encrypting n bits produces ciphertexts of length $\mathrm{poly}(n, \kappa, d^{1/\epsilon})$ (resp. $\mathrm{poly}(n^{1/\epsilon}, \kappa, d^{1/\epsilon^2})$).*

4. REUSABLE GARBLED CIRCUITS

In this section, we show how to use our functional encryption scheme to construct *reusable* garbled circuits. The syntax of a reusable garbling scheme RGb = (RGb.Garble, RGb.Enc, RGb.Eval) is the same as the syntax for a one-time garbling scheme (Sec. 2.1). The security of the scheme (defined in our full paper [25]), intuitively says that a garbled circuit can be used with many encodings while still hiding the circuit and the inputs. More formally, a garbling scheme is input- and circuit-private with reusability if there exists a stateful p.p.t. simulator $S = (S_1, S_2)$ such that S_1, when given as input a circuit size $|C|$, outputs a simulated garbled circuit, and S_2 when given as input $C(x)$ outputs a simulated encoding of x. The simulated garbled circuit and the encodings must be computationally indistinguishable from the real garbled circuit and encodings. Note that S never gets x or C and the adversary can invoke S_2 many times (reusability). Sec. 1.2 already gave an

overview of the idea behind our construction, so we proceed to the construction. Let $\mathsf{E} = (\mathsf{E.KeyGen}, \mathsf{E.Enc}, \mathsf{E.Dec})$ be a semantically secure symmetric-key encryption scheme.

Garbling $\mathsf{RGb.Garble}(1^\kappa, C)$**:**

1. Generate FE keys $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ and a secret key $\mathsf{sk} \leftarrow \mathsf{E.KeyGen}(1^\kappa)$.
2. Let $E := \mathsf{E.Enc}(\mathsf{sk}, C)$.
3. Define U_E to be the following universal circuit:

   > U_E takes as input a secret key sk and a value x:
   > (a) Compute $C := \mathsf{E.Dec}(\mathsf{sk}, E)$.
   > (b) Run C on x.

4. Let $\Gamma \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, U_E)$ be the reusable garbled circuit.
5. Output $\mathsf{gsk} := (\mathsf{fmpk}, \mathsf{sk})$ as the secret key and Γ as the garbling of C.

Encoding $\mathsf{RGb.Enc}(\mathsf{gsk}, x)$**:** Compute $c_x \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, (\mathsf{sk}, x))$ and output c_x.

Evaluation $\mathsf{RGb.Eval}(\Gamma, c_x)$**:** Compute and output $\mathsf{FE.Dec}(\Gamma, c_x)$.

The existence of a semantically secure encryption scheme does not introduce new assumptions because the FE scheme itself is a semantically secure encryption scheme if no key (computed by $\mathsf{FE.KeyGen}$) is ever provided to an adversary.

Tightness of the scheme. The astute reader may have observed that the resulting scheme requires that the encodings be generated in the secret key setting because the encoding of x includes sk. It turns out that generating encodings privately is in fact necessary; if the encodings were publicly generated, the power of the adversary would be the same as in traditional obfuscation, which was shown impossible [4, 26] (see discussion in Sec. 1.1.2).

One might wonder though, whether a reusable garbling scheme exists where the encoding generation is secret key, but $\mathsf{RGb.Garble}$ is public key. We prove in our full paper that this is also not possible based on the impossibility result of [2]; hence, with regard to public versus private key, our reusable garbling result is tight.

PROOF SKETCH OF THEOREM 1.3. Let us first argue $\mathsf{RGb.Eval}$ is correct. By the definition of $\mathsf{RGb.Eval}$, $\mathsf{RGb.Eval}(\Gamma, c_x)$ equals $\mathsf{FE.Dec}(\Gamma, c_x)$, which equals $U_E(\mathsf{sk}, x)$ by the correctness of FE. Finally, by the definition of U_E, $U_E(\mathsf{sk}, x) = C(x)$.

Notice that the encoding algorithm $\mathsf{RGb.Enc}$ produces ciphertexts that do not depend on the circuit size, because of the succinctness property of FE.

We can see that to obtain a RGb scheme for circuits of depth d, we need a FE scheme for polynomially deeper circuits: the overhead comes from the fact that U is universal and it also needs to perform decryption of E to obtain C.

Intuitively, the scheme is secure because E hides the circuit C. Now since FE hides the inputs to $\mathsf{FE.Enc}$, it hides x and sk, and reveals only the result of the computation which is $C(x)$. To prove security formally, we need to construct a simulator $S = (S_1, S_2)$ such that the simulated garbled circuit and encodings are computationally indistinguishable from the real ones. (Our full paper [25] precisely defines security for RGb, including the games for the simulator.) To produce a simulated garbled circuit $\tilde{\Gamma}$, S_1 on input $(1^\kappa, 1^{|C|})$ runs:

1. Generate fresh fmpk, fmsk, and sk as in $\mathsf{RGb.Garble}$.
2. Compute $\tilde{E} := \mathsf{E.Enc}(\mathsf{sk}, 0^{|C|})$. (The reason for encrypting $0^{|C|}$ is that S_1 does not know C).
3. Compute and output $\tilde{\Gamma} \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, U_{\tilde{E}})$.

S_2 receives queries for values $x_1, \ldots, x_t \in \{0,1\}^*$ for some t and needs to output a simulated encoding for each of these. To produce a simulated encoding for x_i, S_2 receives inputs $(C(x_i), 1^{|x_i|}$, and the latest simulator's state) and invokes the simulator $\mathsf{Sim_{FE}}$ of the FE scheme and outputs

$$\tilde{c}_x := \mathsf{Sim_{FE}}(\mathsf{fmpk}, \mathsf{fsk}_{U_{\tilde{E}}}, U_{\tilde{E}}, C(x), 1^{|\mathsf{sk}|+|x_i|}).$$

A potentially alarming aspect of this simulation is that S generates a key for the circuit $0^{|C|}$. Whatever circuit $0^{|C|}$ may represent, it may happen that there is no input x to $0^{|C|}$ that results in the value $C(x)$. The concern may then be that $\mathsf{Sim_{FE}}$ may not simulate correctly. However, this is not a problem because, by semantic security, E and $\tilde{E}$ are computationally indistinguishable so $\mathsf{Sim_{FE}}$ must work correctly, otherwise it breaks semantic security of the encryption scheme E.

To prove indistinguishability of the simulated/ideal and real experiment outputs, we introduce a hybrid experiment. This experiment is the same as the ideal experiment, but $\tilde{E}$ is replaced with $E = \mathsf{E.Enc}(\mathsf{sk}, C)$. This means that the adversary receives a real garbled circuit, but the encodings are still simulated. Thus, the views of the adversary in the ideal and the hybrid experiment differ only in $\tilde{E}$ and E. By the semantic security of the encryption scheme used, these views are computationally indistinguishable. Finally, the hybrid and the real experiment are computationally indistinguishable based on the properties of $\mathsf{Sim_{FE}}$ guaranteed by the security of FE. For the full proof, we refer the reader to our full paper [25]. □

Acknowledgments

This work was supported by an NSERC Discovery Grant, by DARPA awards FA8750-11-2-0225 and N66001-10-2-4089, by NSF awards CNS-1053143 and IIS-1065219, and by Google.

References

[1] S. Agrawal, D. M. Freeman, and V. Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.
[2] S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption: New perspectives and lower bounds. *Cryptology ePrint Archive, Report 2012/468*, 2012.
[3] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.
[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.
[5] B. Barak, I. Haitner, D. Hofheinz, and Y. Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.
[6] M. Bellare, V. T. Hoang, and P. Rogaway. Garbling schemes. *Cryptology ePrint Archive, Report 2012/265*, 2012.
[7] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 28th IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
[8] N. Bitansky, R. Canetti, S. Goldwasser, S. Halevi, Y. T. Kalai, and G. N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.
[9] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.
[10] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[11] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, pages 868–886, 2012.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[13] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[14] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[15] R. Canetti, Y. T. Kalai, M. Varia, and D. Wichs. On symmetric encryption and point obfuscation. In *TCC*, pages 52–71, 2010.

[16] M. A. Davis. Cloud security: Verify, don't trust. *Information Week*, August 2012. http://reports.informationweek.com/abstract/5/8978/.

[17] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[18] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices and applications. Cryptology ePrint Archive, Report 2012/610, 2012.

[19] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

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

[21] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, 2012.

[22] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099, 2012.

[23] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *EUROCRYPT*, pages 506–522, 2010.

[24] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.

[25] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. Cryptology ePrint Archive, Report 2012/733, 2012.

[26] S. Goldwasser and Y. T. Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[27] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[28] S. Goldwasser and G. N. Rothblum. On best-possible obfuscation. In *TCC*, pages 194–213, 2007.

[29] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, August 2012.

[30] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[31] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[32] V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP*, pages 579–591, 2008.

[33] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[34] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[35] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[36] A. B. Lewko and B. Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, 2012.

[37] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22:161–188, April 2009.

[38] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[39] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358, 2010.

[40] T. Okamoto and K. Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.

[41] T. Okamoto and K. Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[42] A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[43] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[44] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[45] Privacy Rights Clearinghouse. Chronology of data breaches, 2012. http://www.privacyrights.org/data-breach.

[46] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[47] A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.

[48] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[49] A. Sahai and B. Waters. Attribute-based encryption for circuits from multilinear maps. Cryptology ePrint Archive, Report 2012/592, 2012.

[50] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

[51] D. Stehlé and R. Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, pages 377–394, 2010.

[52] V. Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In *FOCS*, pages 5–16, 2011.

[53] Verizon RISK Team. 2012 data breach investigations report. http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf.

[54] B. Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

[55] B. Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

[56] H. Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.

[57] A. C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[58] A. C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.