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

March 24, 2013

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.

Contents

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 [Pri12, Ver]. 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 [Dav12].

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) [Gen09, DGHV10, SS10b, BV11b, BV11a, Vai11, BGV12, GHS12a, GHS12b, LTV12, Bra12].

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* [SW05, GPSW06, KSW08, LOS$^+$10, OT10, O'N10, BSW]. 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 [AGVW12] 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 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

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

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 [BW07], Katz, Sahai and Waters [KSW08], Agrawal, Freeman and Vaikuntanathan [AFV11], and Shen, Shi and Waters [SSW09] 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 [SS10a] and Gorbunov, Vaikuntanathan and Wee [GVW12] 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]). [SS10a] was the first to introduce the idea of single-key functional encryption and [GVW12] 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 [SW05, GPSW06, BSW07, GJPS08, LOS$^+$10, Wat11, Wat12, LW12]), and more generally predicate encryption (including [BW07, KSW08, OT09]). 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 [BSW]. Boneh and Waters [BW07] 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 [GVW13], and Sahai and Waters [SW12], 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

[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.

"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 (see Sec. 2.6). 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 [GVW13] achieve ABE for general circuits of bounded depth based on the subexponential Learning With Errors (LWE) intractability assumption. Sahai and Waters [SW12] achieve ABE for general circuits under the less standard k-Multilinear Decisional Diffie-Hellman (see [SW12] for more details); however, when instantiated with the only construction of multilinear maps currently known [GGH12], 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 [GVW13] and the FHE scheme of [BV11a, BGV12], 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 [GVW13] constructs both selectively secure and fully secure ABE schemes. However, the parameters of the LWE assumption are different in the two cases (Sec. 2.3).

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 [AGVW12]. Stated in these terms, our scheme is also a q-collusion-resistant functional encryption scheme like [GVW12], but our scheme's ciphertexts are succinct, whereas [GVW12]'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 [Yao82]. 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 [Yao86], multi-party secure protocols [GMW87], one-time programs [GKR08], KDM-security [BHHI10], verifiable computation [GGP10], homomorphic computations [GHV10] 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*.*

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 [GVW13] 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 [Yao86], multi-party secure protocols [GMW87]) 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

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

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. [BGI+01], followed by Goldwasser and Kalai [GK05], 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 [BGI+01, GK05] 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 [GKR08, GIS+10, BCG+11], relaxing the definition of security [GR07], or considering only specific functions [Wee05, CKVW10].

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 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 [GIS+10, BCG+11]. 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* and define them in Sec. 6.

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. *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.*

1.1.4 Publicly Verifiable Delegation with Secrecy

Recently, Parno, Raykova and Vaikuntanathan [PRV12] 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 [GVW13]'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 [PRV12] 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 [PRV12] (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.

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).

[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.

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 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 (called ABE_2)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 $c \leftarrow \mathsf{ABE}_2.\mathsf{Enc}(y, m_0, m_1)$. Then, one generates a key for a predicate g: $\mathsf{sk}_g \leftarrow \mathsf{ABE}_2.\mathsf{KeyGen}(g)$. 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 ABE_2 multiple times, once for every $i \in \{1, \ldots, \text{size of } \widehat{f(x)}\}$. For the i-th invocation of $\mathsf{ABE}_2.\mathsf{Enc}$, let m_0, m_1 be the garbled labels L_0^i, L_1^i, and let y be $\hat{x}$: $\mathsf{ABE}_2.\mathsf{Enc}(\hat{x}, L_0^i, L_1^i)$. Next, for the i-th invocation of $\mathsf{ABE}_2.\mathsf{KeyGen}$, let g be $\mathsf{FHE.Eval}_f^i$ (the predicate returning the i-th bit of the evaluation of f on an input ciphertext): $\mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{FHE.Eval}_f^i)$. Then, the evaluator can use $\mathsf{ABE}_2.\mathsf{Dec}$ 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.

We now explain how to use this result to obtain the aforementioned applications.

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. Sec. 6 presents the scheme formally.

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 key 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.

Publicly Verifiable Delegation with Secrecy. Delegation schemes aim to enable a weak verifier to delegate computation of a function f on an input x to a prover who can then prove to the verifier that he computed the function correctly. We now show that our single-key functional encryption scheme provides an improvement to publicly verifiable delegation by adding secrecy. We present this improvement only informally, because we prefer to focus on the other applications.

We now briefly recall the scheme of [PRV12] and then discuss how to modify it; we refer the reader to Section 2.6 for formal definitions of ABE and FE. There are two phases in the delegation scheme: the preprocessing phase when the verifier prepares the computation f, and an online phase repeating many times, in which the verifier gives x to the prover who computes $f(x)$ and proves the computation was correct.

In the preprocessing phase, the verifier generates two pairs of master secret and public keys $(\mathsf{msk}_1, \mathsf{mpk}_1)$ and $(\mathsf{msk}_2, \mathsf{mpk}_2)$ for the underlying attribute-based encryption scheme. If f is the function to delegate, the verifier uses msk_1 to generate a key for f denoted sk_f, and msk_2 to generate a key for the negation of f, $\bar{f}(x) := 1 - f(x)$, denoted $\mathsf{sk}_{\bar{f}}$. The verifier then sends both $(\mathsf{mpk}_1, \mathsf{mpk}_2)$ and $(\mathsf{sk}_f, \mathsf{sk}_{\bar{f}})$ to the prover. Generating sk_f and $\mathsf{sk}_{\bar{f}}$ takes time that is proportional to the size of the circuit computing f, and thus is a costly operation. However, this is done only once in the preprocessing phase.

Whenever the verifier wants the prover to compute f on an input x, he chooses two random messages

m_1, m_2 and sends the prover the encryptions of (x, m_*) under the two keys: $(\mathsf{Enc}(\mathsf{mpk}_1, x, m_1)$ and $\mathsf{Enc}(\mathsf{mpk}_2, x, m_2))$. The properties of the attribute-based encryption scheme guarantees that, if $f(x) = 1$, the prover obtains m_1 using sk_f and $\perp$ using $\mathsf{sk}_{\bar{f}}$ so no information about m_0, and vice versa if $f(x) = 0$. Therefore, the fact that the prover provides m_1 to the verifier is a proof that $f(x)$ was 1.

Importantly, this delegation scheme can be made to have the desired property of being *publicly verifiable*, meaning that the verifier can produce a "verification key" with which anyone can check the prover's work. This is done by having the verifier also send two point function obfuscations, one of the point m_1 and the other of the point m_2.

This reduction from ABE to publicly verifiable delegation can be combined with the recent result of [GVW13] providing ABE schemes for any depth circuit: the result is a publicly verifiable 2-message delegation scheme in the preprocessing model for any depth d circuit with verifier's work being proportional to the depth d and the prover's work proportional to the circuit size.

Note however, that this scheme is not secret because ABE does not hide the input x from the prover. It is well known that x can be made secret by encrypting everything using a fully homomorphic encryption scheme. However, this comes at the cost of losing the public verifiability property. Our idea is to replace the ABE scheme with our functional encryption scheme in the protocol above; now the ciphertexts $\mathsf{Enc}(\mathsf{mpk}_1, x, m_1)$ and $\mathsf{Enc}(\mathsf{mpk}_2, x, m_2)$ hide x and the scheme provides secrecy because the prover learns nothing about x other than $f(x)$. The public verifiability of the scheme remains the same.

We remark that we could provide a stronger version of secrecy by also hiding the result $f(x)$ from the prover; such stronger secrecy is non-standard for delegation, so we do not delve on it. (The idea is for the client to concatenate a random bit to each input x and have the function f output the opposite result when the bit is set. In this way, the prover does not learn anything from seeing which ciphertext decrypts to non-$\perp$.)

2 Preliminaries

2.1 Notation

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. We use $p(\cdot)$ to denote that p is a function that takes one input. Similarly, $p(\cdot, \cdot)$ denotes a function p that takes two inputs.

We say that a function f is negligible in an input parameter κ, if for all $d > 0$, there exists K such that for all $\kappa > K$, $f(\kappa) < k^{-d}$. For brevity, we write: for all sufficiently large κ, $f(\kappa) = \mathrm{negl}(\kappa)$. We say that a function f is polynomial in an input parameter κ, if there exists a polynomial p such that for all κ, $f(\kappa) \leq p(\kappa)$. We write $f(\kappa) = \mathrm{poly}(\kappa)$. A similar definition holds for $\mathrm{polylog}(\kappa)$.

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 work with arithmetic circuits over GF(2). These circuits have two types of gates: $+$ mod 2 and $\times$ mod 2. Unless the context specifies otherwise, we consider circuits with one bit of output (also called boolean).

Two ensembles, $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$, are said to be *computationally indistinguishable* (and denoted $\{X_\kappa\}_{\kappa \in \mathbb{N}} \stackrel{c}{\approx} \{Y_\kappa\}_{\kappa \in \mathbb{N}}$) if for every probabilistic polynomial-time algorithm D,

$$|\Pr[D(X_\kappa, 1^\kappa) = 1] - \Pr[D(Y_\kappa, 1^\kappa) = 1]| = \mathrm{negl}(\kappa).$$

In our security definitions, we will define probabilistic experiments and denote by random variables their outputs. For example, $\mathsf{Exp}^{\mathsf{real}}_{E,A}(1^\kappa)$ denotes the random variable representing the output of the real experiment for scheme E with adversary A on security parameter κ. Moreover, $\{\mathsf{Exp}^{\mathsf{real}}_{E,A}(1^\kappa)\}_{\kappa\in\mathbb{N}}$ denotes the ensemble of such random variables indexed by $\kappa \in \mathbb{N}$.

2.2 Background on Learning With Errors (LWE)

The security of our results will be based on the Learning with Errors (LWE) assumption, first introduced by Regev [Reg05]. 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 [Pei09] 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 [Reg05, Pei09] 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 $2^{\tilde{O}(\ell^{1-\epsilon})}$ [AKS01, MV10] 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.

Appendix A provides more detailed background information on LWE.

2.3 Fully Homomorphic Encryption (FHE)

The notion of fully homomorphic encryption was first proposed by Rivest, Adleman and Dertouzos [RAD78] in 1978. The first fully homomorphic encryption scheme was proposed in a breakthrough work by Gentry in 2009 [Gen09]. A history and recent developments on fully homomorphic encryption is surveyed in [Vai11]. We recall the definitions and semantic security of fully homomorphic encryption; the definitions below are based on [Vai11] with some adaptations.

Definition 2.1. *A homomorphic (public-key) encryption scheme* FHE *is a quadruple of polynomial time algorithms* (FHE.KeyGen, FHE.Enc, FHE.Dec, FHE.Eval) *as follows:*

- FHE.KeyGen(1^κ) *is a probabilistic algorithm that takes as input the security parameter* 1^κ *and outputs a public key* pk *and a secret key* sk.
- FHE.Enc(pk, $x \in \{0,1\}$) *is a probabilistic algorithm that takes as input the public key* pk *and an input bit* x *and outputs a ciphertext* ψ.
- FHE.Dec(sk, ψ) *is a deterministic algorithm that takes as input the secret key* sk *and a ciphertext* ψ *and outputs a message* $x^* \in \{0,1\}$.
- FHE.Eval(pk, $C, \psi_1, \psi_2, \ldots, \psi_n$) *is a deterministic algorithm that takes as input the public key* pk, *some circuit* C *that takes* n *bits as input and outputs one bit, as well as* n *ciphertexts* $\psi_1, \ldots, \psi_n$. *It outputs a ciphertext* ψ_C.

Compactness: *For all security parameters* κ, *there exists a polynomial* $p(\cdot)$ *such that for all input sizes* n, *for all* $x_1 \ldots x_n$, *for all* C, *the output length of* FHE.Eval *is at most* $p(n)$ *bits long.*

Definition 2.2 ($\mathcal{C}$-homomorphism)**.** *Let* $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *be a class of boolean circuits, where* $\mathcal{C}_n$ *is a set of boolean circuits taking* n *bits as input. A scheme* FHE *is* $\mathcal{C}$*-homomorphic if for every polynomial* $n(\cdot)$, *for* *every sufficiently large security parameter κ, for every circuit $C \in \mathcal{C}_n$, and for every input bit sequence $x_1, \ldots, x_n$, where $n = n(\kappa)$,*

$$\begin{aligned}\Pr[&(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa);\\ &\psi_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x_i) \text{ for } i = 1 \ldots n;\\ &\psi \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, C, \psi_1, \ldots, \psi_n) :\\ &\mathsf{FHE.Dec}(\mathsf{sk}, \psi) \neq C(x_1, \ldots x_n)] = \mathrm{negl}(\kappa).\end{aligned}$$

where the probability is over the coin tosses of FHE.KeyGen *and* FHE.Enc.

Definition 2.3 (Fully homomorphic encryption)**.** *A scheme* FHE *is fully homomorphic if it is homomorphic for the class of all arithmetic circuits over* $\mathsf{GF}(2)$.

Definition 2.4 (Leveled fully homomorphic encryption)**.** *A leveled fully homomorphic encryption scheme is a homomorphic scheme where* FHE.KeyGen *receives an additional input 1^d and the resulting scheme is homomorphic for all depth-d arithmetic circuits over* $\mathsf{GF}(2)$.

Definition 2.5 (IND-CPA security)**.** *A scheme* FHE *is IND-CPA secure if for any p.p.t. adversary $\mathcal{A}$,*

$$\begin{aligned}\big|\Pr[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa) : \mathcal{A}(\mathsf{pk}, \mathsf{FHE.Enc}(\mathsf{pk}, 0)) = 1] -&\\ \Pr[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa) : \mathcal{A}(\mathsf{pk}, \mathsf{FHE.Enc}(\mathsf{pk}, 1)) = 1]\big| &= \mathrm{negl}(\kappa).\end{aligned}$$

We now state the result of Brakerski, Gentry and Vaikuntanathan [BGV12] that shows a leveled fully homomorphic encryption scheme based on the LWE assumption:

Theorem 2.1 ([BV11a, BGV12])**.** *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* $\mathsf{depth}(C_f) \cdot \mathrm{poly}(\log n, \log d)$.

All known fully homomorphic encryption schemes (as opposed to merely leveled schemes) require an additional assumption related to circular security of the associated encryption schemes. However, we do not need to make such an assumption in this work because we only use a leveled homomorphic encryption scheme in our constructions.

2.4 Background on Garbled Circuits

We will now define garbled circuits. Initially, garbled circuits were presented by Yao [Yao82] in the context of secure two-party computation and later, they were then proven secure by Lindell and Pinkas [LP09]. Very recently, the notion has been formalized by Bellare et al. [BHR12]. For simplicity, we present more concise definitions of garbled circuits than in [BHR12].

Definition 2.6 (Garbling scheme)**.** *A garbling scheme for a family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n \in \mathbb{N}}$ with $\mathcal{C}_n$ a set of boolean circuits taking as input n bits, is a tuple of p.p.t. algorithms* Gb = (Gb.Garble, Gb.Enc, 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*.*

- $\mathsf{Gb.Eval}(\Gamma, c)$ *takes as input a garbled circuit* Γ*, an encoding* c *and outputs a value* y *which should be* $C(x)$*.*

Correctness. *For any polynomial* $n(\cdot)$*, for all sufficiently large security parameters* κ*, for* $n = n(\kappa)$*, for all circuits* $C \in \mathcal{C}_n$ *and all* $x \in \{0,1\}^n$*,*

$$\Pr[(\Gamma, \mathsf{sk}) \leftarrow \mathsf{Gb.Garble}(1^\kappa, C); c \leftarrow \mathsf{Gb.Enc}(\mathsf{sk}, x); y \leftarrow \mathsf{Gb.Eval}(\Gamma, c) : C(x) = y] = 1 - \mathrm{negl}(\kappa).$$

Efficiency. *There exists a universal polynomial* $p = p(\kappa, n)$ *(*p *is the same for all classes of circuits* $\mathcal{C}$*) such that for all input sizes* n*, security parameters* κ*, for all boolean circuits* C *of with* n *bits of input, for all* $x \in \{0,1\}^n$*,*

$$\Pr[(\Gamma, \mathsf{sk}) \leftarrow \mathsf{Gb.Garble}(1^\kappa, C) : |\mathsf{sk}| \leq p(\kappa, n) \text{ and } runtime(\mathsf{Gb.Enc}(\mathsf{sk}, x)) \leq p(\kappa, n)] = 1.$$

Note that since $\mathsf{Gb.Enc}$ is a p.p.t. algorithm, it suffices to ensure that $|\mathsf{sk}| \leq p(\kappa, n)$ and obtain that $\mathsf{Gb.Enc}$'s runtime is also at most a polynomial. We prefer to keep the runtime of $\mathsf{Gb.Enc}$ in the definition as well for clarity.

Remark 2.2 (Remark on the efficiency property)**.** *Intuitively, a garbling scheme is efficient if the time to encode is shorter than the time to run the circuit. This requirement can be formalized in a few ways. A first definition is as provided above in Def. 2.6. Another definition is to allow* $|\mathsf{sk}|$ *and the runtime of* $\mathsf{Gb.Enc}$ *to also depend on the depth of the circuits in* $\mathcal{C}$*, but require that it does not depend on their size.*

Yao garbled circuits. The garbled circuits presented by Yao have a specific property of the encoding scheme that is useful in various secure function evaluation 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 input x of n bits 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). All 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.

Bellare et al. [BHR12] also present a third property which they call authenticity; informally, this requires that an adversary should not be able to come up with a different result of the garbled circuit that could be "de-garbled" into a valid value. We do not present this property here because it is straightforward to show that a garbling scheme with input and circuit privacy as we define them below implies a different garbling scheme with the authenticity property and we would need to provide a slightly more complicated syntax for the definition of garbled circuits (with an additional "de-garbling" algorithm).

We now present the one-time security of garbling circuits. The security definition for reusable garbled will be presented later, in Sec. 4.

Definition 2.7 (Input and circuit privacy)**.** *A garbling scheme* Gb *for a family of circuits* $\{\mathcal{C}_n\}_{n \in \mathbb{N}}$ *is input and circuit private if there exists a p.p.t. simulator* $\mathsf{Sim}_{\mathsf{Garble}}$*, such that for every p.p.t. adversaries* A *and* D*, for all sufficiently large security parameters* κ*,*

$$
\begin{aligned}
&|\Pr[(x,C,\alpha)\leftarrow A(1^\kappa);(\Gamma,\mathsf{sk})\leftarrow \mathsf{Gb.Garble}(1^\kappa,C);c\leftarrow \mathsf{Gb.Enc}(\mathsf{sk},x): D(\alpha,x,C,\Gamma,c)=1]-\\
&\ \Pr[(x,C,\alpha)\leftarrow A(1^\kappa);(\tilde{\Gamma},\tilde{c})\leftarrow \mathsf{Sim}_{\mathsf{Garble}}(1^\kappa,C(x),1^{|C|},1^{|x|}): \qquad D(\alpha,x,C,\tilde{\Gamma},\tilde{c})=1]|=\mathrm{negl}(\kappa),
\end{aligned}
$$

where we consider only A such that for some n, $x\in\{0,1\}^n$ and $C\in\mathcal{C}_n$.

Intuitively, this definition says that, for any circuit or input chosen adversarially, one can simulate in polynomial time the garbled circuit and the encoding solely based on the computation result (and relevant sizes). The variable α represents any state that A may want to convey to D.

A few variants of Yao garbling schemes exist (for example, [BHR12]) that provide both input and circuit privacy under the basic one-way function assumption. Any such construction is suitable for our scheme.

Theorem 2.3 ([Yao82, LP09])**.** *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).*

2.5 Attribute-Based Encryption (ABE)

We now provide the definition of attribute-based encryption from the literature (e.g., [GPSW06, LOS+10, GVW13]).

Definition 2.8 (Attribute-Based Encryption)**.** *An attribute-based encryption scheme (*ABE*) for a class of predicates $\mathcal{P}=\{\mathcal{P}_n\}_{n\in\mathbb{N}}$ represented as boolean circuits with n input bits and one output bit and an associated 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 $P\in\mathcal{P}_n$, for some n, outputs a key* fsk_P *corresponding to P.*
- $\mathsf{ABE.Enc}(\mathsf{fmpk},x,M)$*: Takes as input the public key* fmpk*, an attribute $x\in\{0,1\}^n$, for some n, and a message $M\in\mathcal{M}$ 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 $x\in\{0,1\}^n$, and messages $M\in\mathcal{M}$:*

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

The space $\{0,1\}^n$ is referred to as the *attribute space* (with an attribute size of n) and $\mathcal{M}$ is referred to as the *message space*.

Intuitively, the security of ABE is that M is revealed only if $P(x)=1$. Regarding the attribute x, ABE's security does not require any secrecy of the attribute, so x may leak no matter what is the value of $P(x)$. Many ABE schemes have been proven secure under indistinguishability-based definitions. Despite being weaker than simulation-based definitions, such definitions suffice for the security of our construction, so we present them here. Two notions of security have been used in previous work: full and selective security. Full security allows the adversary to provide the challenge ciphertext after seeing the public key, whereas in selective security, the adversary must provide the challenge ciphertext before seeing the public key. We present both in the full security and selective security cases, because the ABE primitive we use [GVW13] achieves them with different parameters of the gapSVP assumption. We only provide the security definition for the case when the adversary can ask for *a single key* because this is all we need for our results.

Definition 2.9 (Attribute-based encryption security)**.** *Let* ABE *be an attribute-based encryption scheme for a class of predicates* $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$*, and an associated message space* $\mathcal{M}$*, and let* $A = (A_1, A_2, A_3)$ *be a triple of p.p.t. adversaries. Consider the following experiment.*

$\underline{\mathsf{Exp}_{\mathsf{ABE}}(1^\kappa)}$**:**

1: $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$
2: $(P, \mathsf{state}_1) \leftarrow A_1(\mathsf{fmpk})$
3: $\mathsf{fsk}_P \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}, P)$
4: $(M_0, M_1, x, \mathsf{state}_2) \leftarrow A_2(\mathsf{state}_1, \mathsf{fsk}_P)$
5: Choose a bit b at random and let $c \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}, x, M_b)$.
6: $b' \leftarrow A_3(\mathsf{state}_2, c)$. If $|M_0| = |M_1|$, $P(x) = 0$, and $b = b'$, output 1, else output 0.

We say that the scheme is a single-key fully-secure attribute-based encryption if for all p.p.t. adversaries A*, and for all sufficiently large* κ*:*

$$\Pr[\mathsf{Exp}_{\mathsf{ABE},A}(1^\kappa) = 1] \leq 1/2 + \mathrm{negl}(\kappa).$$

We say that the scheme is single-key selectively secure if the same statement holds for a slightly modified game in which A *provides* x *before receiving* fmpk.

Attribute-based encryption schemes have been constructed for the class of Boolean formulas [GPSW06, LOS$^+$10] and most recently for the class of all polynomial-size circuits: Gorbunov, Vaikuntanathan and Wee [GVW13] based on the subexponential Learning With Errors (LWE) intractability assumption, and Sahai and Waters [SW12] based on the k-Multilinear Decisional Diffie-Hellman (see [SW12] for more details). Our reduction can start from any of these schemes, but in this paper, we choose [GVW13] because it is based on LWE, which is a more standard assumption and is also the assumption for our other building block, FHE.

Before we state the results of Gorbunov, Vaikuntanathan and Wee [GVW13], we will set up some notation. Let d and p be two univariate polynomials. Define $\mathcal{C}_{n,d(n),p(n)}$ to be the class of all boolean circuits on n inputs of depth at most $d(n)$ and size at most $p(n)$. Let $\mathcal{C}_{n,d(n)} := \bigcup_{\text{polynomial p}} \mathcal{C}_{n,d(n),p(n)}$. An attribute-based encryption or functional encryption scheme that supports circuits in $\mathcal{C}_{n,d(n)}$ is called a d-leveled attribute-based encryption or functional encryption scheme, respectively. We also refer to an ABE or FE scheme as leveled, if it is d-leveled for some d. We are now ready to state the theorem of [GVW13].

Theorem 2.4 ([GVW13])**.** *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 by a polynomial algorithm to within a* $2^{O(\ell^\epsilon)}$ *factor in the worst case. Then, for every* n *and every polynomial* $d = d(n)$*, there is a selectively secure* d*-leveled attribute-based encryption scheme where encrypting* n *bits produces ciphertexts of length* $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$.

Furthermore, assuming that gapSVP *in* ℓ *dimensions is hard to approximate to within a* $2^{O(\ell^{\epsilon})}$ *factor* in time $2^{O(\ell^{\epsilon})}$*, the scheme is fully secure with ciphertexts of length* $\mathsf{poly}(n, \kappa, d^{1/\epsilon^2})$.

In either case, the scheme is secure with polynomially many secret-key queries.

2.5.1 Two-Outcome Attribute-Based Encryption

We use an attribute-based encryption scheme with a slightly modified definition. The setup and key generation algorithms are the same as in previous schemes. The difference is in the encryption and decryption algorithms: instead of encrypting one message M in one ciphertext, we encrypt two messages M_0 and M_1 in the same ciphertext such that M_0 is revealed if the predicate evaluates to zero on the attribute, and M_1 is revealed if the predicate evaluates to one. Since there are two possible outcomes of the decryption algorithm, we call the modified scheme *a two-outcome attribute-based encryption* scheme. Such a variant of ABE has been used for other purposes by [PRV12].

Definition 2.10 (Two-Outcome Attribute-Based Encryption)**.** *A two-outcome attribute-based encryption scheme (*ABE_2*) for a class of predicates* $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ *represented as boolean circuits with* n *input bits, and a message space* $\mathcal{M}$ *is a tuple of algorithms* ($\mathsf{ABE}_2.\mathsf{Setup}$, $\mathsf{ABE}_2.\mathsf{KeyGen}$, $\mathsf{ABE}_2.\mathsf{Enc}$, $\mathsf{ABE}_2.\mathsf{Dec}$) *as follows:*

- $\mathsf{ABE}_2.\mathsf{Setup}(1^\kappa)$*: Takes as input a security parameter* 1^κ *and outputs a public master key* fmpk *and a master secret key* fmsk.
- $\mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{fmsk}, P)$*: Given a master secret key* fmsk *and a predicate* $P \in \mathcal{P}$*, outputs a key* fsk_P *corresponding to* P.
- $\mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}, x, M_0, M_1)$*: 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.
- $\mathsf{ABE}_2.\mathsf{Dec}(\mathsf{fsk}_P, c)$*: 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* $M_0, M_1 \in \mathcal{M}$*:*

$$\Pr\left[\begin{array}{l}(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{ABE}_2.\mathsf{Setup}(1^\kappa);\\ \mathsf{fsk}_P \leftarrow \mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{fmsk}, P);\\ c \leftarrow \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}, x, M_0, M_1);\\ M^* \leftarrow \mathsf{ABE}_2.\mathsf{Dec}(\mathsf{fsk}_P, c):\\ M^* = M_{P(x)}\end{array}\right] = 1 - \mathrm{negl}(\kappa).$$

We now define the security for single-key two-outcome attribute-based encryption. Intuitively, the security definition requires that, using a token for a predicate P, an adversary can decrypt one of the two messages encrypted in C based on the evaluation of P on the attribute, but does not learn anything about the other message.

Definition 2.11 (Two-outcome attribute-based encryption security)**.** *Let* ABE_2 *be a two-outcome attribute-based encryption scheme for the class of predicates* $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ *and associated message space* $\mathcal{M}$ *and let* $A = (A_1, A_2, A_3)$ *be a triple of p.p.t. adversaries. Consider the following experiment.*

$\underline{\mathsf{Exp}_{\mathsf{ABE}_2}(1^\kappa)\textbf{:}}$

1: $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{ABE}_2.\mathsf{Setup}(1^\kappa)$
2: $(P, \mathsf{state}_1) \leftarrow A_1(\mathsf{fmpk})$
3: $\mathsf{sk}_P \leftarrow \mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{fmsk}, P)$
4: $(M, M_0, M_1, x, \mathsf{state}_2) \leftarrow A_2(\mathsf{state}_1, \mathsf{sk}_P)$
5: Choose a bit b at random. Then, let

$$c = \begin{cases} \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}, x, M, M_b), & \text{if } P(x) = 0, \\ \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}, x, M_b, M), & \text{otherwise.} \end{cases}$$

6: $b' \leftarrow A_3(\mathsf{state}_2, c)$. If $b = b'$, $\exists\, n$ such that, for all $P \in \mathcal{P}_n$, messages $M, M_0, M_1 \in \mathcal{M}$, $|M_0| = |M_1|$, $x \in \{0,1\}^n$, output 1, else output 0.

We say that the scheme is a fully-secure single-key two-outcome ABE if for all p.p.t. adversaries A, and for all sufficiently large security parameters κ:

$$\Pr[\mathsf{Exp}_{\mathsf{ABE}_2,A}(1^\kappa) = 1] \le 1/2 + \mathrm{negl}(\kappa).$$

The scheme is single-key selectively secure if A needs to provide x before receiving fmpk.

As before, we need only a single-key ABE_2 scheme for our construction.

A class of predicates $\{\mathcal{P}_n\}_n$ is closed under negation if for all input sizes n and for all predicates $p \in \mathcal{P}_n$, we have $\bar{p} \in \mathcal{P}_n$; $\bar{p}$ is the negation of p, namely $\bar{p}(y) = 1 - p(y)$ for all y.

Claim 2.5. *Assuming there is an ABE scheme for a class of predicates closed under negation, there exists a two-outcome ABE scheme for the same class of predicates.*

The proof of this claim is immediate and we present it in Appendix B, for completeness.

2.6 Functional Encryption (FE)

We recall the functional encryption definition from the literature [KSW08, BSW, GVW12] with some notational changes.

Definition 2.12 (Functional Encryption)**.** *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 (*FE.Setup, FE.KeyGen, FE.Enc, 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 $x \in \{0,1\}^n$,*

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

2.6.1 Security of Functional Encryption

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 (the adversary can learn the circuit C). As mentioned, two notions of security have been used in previous work: full and selective security, with the same meaning as for ABE. We present both definitions because we achieve them with different parameters of the gapSVP assumption. Our definitions are simulation-based: 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 2.13. *(*FULL-SIM*-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:*

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa)}$:	$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},A,S}(1^\kappa)}$:
1: $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$	
2: $(f, \mathsf{state}_A) \leftarrow A_1(\mathsf{fmpk})$	
3: $\mathsf{fsk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, f)$	
4: $(x, \mathsf{state}'_A) \leftarrow A_2(\mathsf{state}_A, \mathsf{fsk}_f)$	
5: $c \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, x)$	*5:* $\tilde{c} \leftarrow S(\mathsf{fmpk}, \mathsf{fsk}_f, f, f(x), 1^{\lvert x\rvert})$
6: Output (state'_A, c)	*6:* Output $(\mathsf{state}'_A, \tilde{c})$

The scheme is said to be (single-key) FULL-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:*

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

We now define selective security, which is a weakening of full security, by requiring the adversary to provide the challenge input x before seeing the public key or any other information besides the security parameter. We simply specify the difference from full security.

Definition 2.14 (SEL-SIM-Security)**.** *The same as Def. 2.13, but modify the game so that the first step consists of A specifying the challenge input x given only the security parameter.*

It is easy to see that the full simulation definition (FULL-SIM-security) implies the selective definition (SEL-SIM-security).

The literature [BSW, AGVW12] has considered another classification for simulation-based definitions: adaptive versus non-adaptive security. In the adaptive case, the adversary is allowed to ask for a function f after seeing the ciphertext c for an input x. In the non-adaptive case, the adversary must first provide f

and only then ask for encryptions of inputs x. Our definition falls in the non-adaptive category. Boneh et al. [BSW] have shown that adaptive simulation-based security is unachievable even for single-key functional encryption for the simple functionality of identity-based encryption. As such, the adaptive definition appears too strong and is unachievable for general functionalities, so we use non-adaptive security.

Remark 2.6. *Attribute-based encryption can be viewed as functional encryption for a specific class of functionalities, where the additional information leaked is part of the output to the function. Namely, consider a class of functions $\mathcal{F}$ whose plaintext space consists of pairs of values from $\{0,1\}^n \times \mathcal{M}$, where $\{0,1\}^n$ is the attribute space (with an attribute size of n) and $\mathcal{M}$ is the message space. The class of functions for ABE is more specific: there exists an associated predicate class $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ to $\mathcal{F}$ such that for every n, for every $f \in \mathcal{F}_n$, there is an associated predicate $P \in \mathcal{P}_n$ to f such that*

$$f(x, M) = \begin{cases} (x, M), & \text{if } P(x) = 1, \\ (x, \perp), & \text{otherwise.} \end{cases}$$

Since the attribute x is in the output of the function no matter what P is, x leaks from the scheme no matter what (x is public). Therefore, this functionality leads to weaker security guarantees than functional encryption in a conceptual way: the value to be computed on, x, leaks with ABE (whereas the value M on which P does not compute remains secret when $P(x) = 0$), whereas the input x to the computation is hidden with FE.

3 Our Functional Encryption Scheme

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) two-outcome attribute-based encryption scheme ABE_2, and a Yao garbling scheme Gb.

We let $\mathsf{FHE.Eval}_f(\mathsf{hpk}, \bar{\psi})$ denote the circuit that performs homomorphic evaluation of the function f on the vector of ciphertexts $\bar{\psi} := (\psi_1, \ldots, \psi_n)$ using the public key hpk, and we will let $\mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi)$ denote the predicate that computes the i-th output bit of $\mathsf{FHE.Eval}_f(\mathsf{hpk}, \bar{\psi})$. Namely,

$$\mathsf{FHE.Eval}_f(\mathsf{hpk}, \bar{\psi}) = \left(\mathsf{FHE.Eval}^1_f(\mathsf{hpk}, \bar{\psi}), \ldots, \mathsf{FHE.Eval}^\lambda_f(\mathsf{hpk}, \bar{\psi})\right),$$

where $\lambda = \lambda(\kappa) = \left|\mathsf{FHE.Eval}_f(\mathsf{hpk}, \bar{\psi})\right|$. Our main theorem then says:

Theorem 3.1. *There is a (fully/selectively secure) single-key functional encryption scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *for any class of circuits $\mathcal{C}$ that take n bits of input and produce a one-bit output, assuming the existence of the following primitives:*

- *an IND-CPA-secure $\mathcal{C}$-homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$*;*
- *a (fully/selectively secure) single-key attribute-based encryption scheme* $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ *for the class of predicates $\mathcal{P} = \mathcal{P}_{\mathcal{C},\mathsf{FHE}}$ where*

$$\mathcal{P}_{\mathcal{C},\mathsf{FHE}} = \{\mathsf{FHE.Eval}^i_C, 1 - \mathsf{FHE.Eval}^i_C :\ C \in \mathcal{C} \text{ and } i \in \{1, \ldots, \lambda\}\}; \quad \text{and}$$

- *a Yao garbling scheme* $\mathsf{Gb} = (\mathsf{Gb.Garble}, \mathsf{Gb.Enc}, \mathsf{Gb.Eval})$ *that is input- and circuit-private.*

The succinctness property of the functional encryption scheme is summarized as follows: the size of the ciphertexts $\mathsf{ctsize}_{\mathsf{FE}}(n)$ *in the resulting scheme for* n *bits of input is*

$$2 \cdot \mathsf{ctsize}_{\mathsf{FHE}} \cdot \left[\mathsf{ctsize}_{\mathsf{ABE}}(n \cdot \mathsf{ctsize}_{\mathsf{FHE}} + \mathsf{pksize}_{\mathsf{FHE}})\right] + \mathrm{poly}(\kappa, \mathsf{ctsize}_{\mathsf{FHE}}, \mathsf{sksize}_{\mathsf{FHE}}).$$

where $\mathsf{ctsize}_{\mathsf{ABE}}(k)$ *denotes the size of the ciphertexts in the attribute-based encryption scheme for a* k*-bit attribute and a* $\mathrm{poly}(\kappa)$*-bit message,* $\mathsf{ctsize}_{\mathsf{FHE}}$ *denotes the size of the ciphertexts in the fully homomorphic encryption scheme for a single-bit message and* $\mathsf{pksize}_{\mathsf{FHE}}$ *(resp.* $\mathsf{sksize}_{\mathsf{FHE}}$*) denotes the size of the public key (resp. secret key) in the fully homomorphic encryption scheme.*

Since garbling schemes can be constructed from one-way functions, our theorem says that we can move from attribute-based encryption, in which the part of the input that the function computes on leaks, to a functional encryption scheme, in which no part of the input leaks using fully homomorphic encryption and Yao garbled circuits.

We can see that if the ciphertext size in the ABE scheme and the fully homomorphic encryption scheme does not depend on the circuit size (and thus, those schemes are by themselves succinct), then neither will the resulting ciphertexts of the FE scheme depend on the circuit size; namely, the reduction does not blow up the ciphertexts and is "succinctness-preserving". We know of both a leveled FHE scheme and a leveled ABE scheme ([GVW13]) with ciphertext lengths independent of the size of the circuits to evaluate; the ciphertext size in these schemes just depends on the depth of the circuits.

We note that fully homomorphic encryption schemes with succinct ciphertexts that are also independent of depth are known, albeit under the stronger assumption of circular security of the underlying schemes. Thus, if the result of [GVW13] can be improved to remove the depth dependency of the ciphertexts in the ABE scheme, one automatically obtains a corresponding result for ABE using our reduction.

Our theorem needs the ABE scheme to be secure only with a single key, even though the recent constructions [GVW13] and [SW12] can tolerate an arbitrary number of keys.

Our main theorem is thus a reduction, which has a number of useful corollaries. The first and perhaps the most important one shows how to combine the leveled fully homomorphic encryption scheme from [BV11a, BGV12] with the recent construction of a leveled attribute-based encryption scheme from [GVW13] to obtain a leveled functional encryption scheme based solely on the hardness of LWE. In other words, the corollary says that for every depth d, there is a functional encryption scheme for the class of all Boolean circuits of (arbitrary) polynomial size and depth at most d. The size of the ciphertexts in the scheme grows with d, and is of course independent of the size of the circuits it supports.

Let d and p be polynomial functions. Define $\mathcal{C}_{n,d(n),p(n)}$ to be the class of all Boolean circuits on n inputs of depth at most $d(n)$ and size at most $p(n)$. Let $\mathcal{C}_{n,d(n)} := \bigcup_{\text{polynomial p}} \mathcal{C}_{n,d(n),p(n)}$.

Corollary 3.2 (The LWE Instantiation)**.** *We have the following two constructions of functional encryption based on the* worst-case *hardness of lattice problems:*

- *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 polynomial time) in the worst case. Then, for every* n *and every polynomial* $d = d(n)$*, there is a selectively-secure (succinct single-key) functional encryption scheme for the class* $\mathcal{C}_{n,d(n)}$ *where encrypting* n *bits produces ciphertexts of length* $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$.

- *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 time $2^{O(\ell^\epsilon)}$ *in the worst case. Then, for every n and every polynomial $d = d(n)$, there is a fully-secure (succinct single-key) functional encryption scheme for the class $\mathcal{C}_{n,d(n)}$ where encrypting n bits produces ciphertexts of length* $\mathsf{poly}(n^{1/\epsilon}, \kappa, d^{1/\epsilon^2})$.

The corollary follows directly from Theorem 3.1, by invoking the leveled fully homomorphic encryption scheme of [BV11a] (see Theorem 2.1) and the leveled attribute-based encryption scheme of [GVW13] (see Theorem 2.4). The concrete constructions and proofs in fact go through the learning with errors (LWE) problem; we refer to [BV11a, GVW13] for the concrete setting of parameters.

Letting *universal* attribute-based encryption or functional encryption denote a single attribute-based encryption or functional encryption scheme scheme, respectively, that supports the class of all polynomial-size circuits, we have the following corollary:

Corollary 3.3 (Universal Functional Encryption)**.** *Assuming that fully homomorphic encryption schemes exist and universal single-key attribute-based encryption schemes exist, there is a universal single-key functional encryption scheme.*

Of the two prerequisites mentioned above, we know that fully homomorphic encryption schemes exist (albeit under stronger assumptions than merely LWE). Thus, the corollary provides a way to immediately translate any universal attribute-based encryption scheme into a functional encryption scheme. We point out that universal functional encryption schemes, by definition, have succinct ciphertexts.

A recent result of Gorbunov, Vaikuntanathan and Wee [GVW12] shows how to generically convert single-key functional encryption schemes into q-keys functional encryption schemes for any bounded q, where the latter provide security against an attacker that can obtain secret keys of up to q functions of her choice. The size of the ciphertexts in the q-keys scheme grows polynomially with q.

Corollary 3.4 (Many queries, using [GVW12])**.** *For every $q = q(\kappa)$, there is a (fully/selectively-secure) q-keys succinct functional encryption scheme for any class of circuits $\mathcal{C}$ that take n bits of input and produce a one-bit output, assuming the existence of primitives as in Theorem 3.1. The size of the ciphertexts* $\mathsf{ctsize}_{\mathsf{FE}}(n)$ *in the resulting scheme is q times as large as in Theorem 3.1.*

Finally, a functional encryption scheme for circuits that output multiple bits can be constructed by thinking of the circuit as many circuits each with one-bit output, and modifying the key generation procedure to produce keys for each of them. This gives us the following corollary although we remark that more efficient methods of achieving this directly are possible using homomorphic encryption schemes that pack multiple bits into a single ciphertext [SV11, BGV12, GHS12a].

Corollary 3.5 (Many queries, many output bits)**.** *For every $q = q(\kappa)$ and $k = k(n)$, there is a (fully/selectively secure) q-keys functional encryption scheme for any class of circuits $\mathcal{C}$ that take n bits of input and produce k bits of output, assuming the existence of primitives as in Theorem 3.1. The size of the ciphertexts* $\mathsf{ctsize}_{\mathsf{FE}}(n)$ *in the resulting scheme is qk times as large as in Theorem 3.1.*

Remark 3.6 (On the necessity of single-key security)**.** *We note that even though the work of [GVW13] provides an attribute-based scheme that is secure even if the adversary obtains secret keys for polynomially many functions, our theorem gives us only a single-key secure scheme. Indeed, this is inherent by the impossibility result of [AGVW12] if we ask for (even a very weak notion of) simulation security, as we do here. Corollary 3.4 gives us a way to get (simulation-)security with q queries for any a priori bounded q, albeit at the expense of the ciphertext growing as a function of q.*

Remark 3.7 (On composing our functional encryption scheme)**.** *One might wonder if chaining is possible with our FE scheme. Namely, one could try to generate keys for a function f that computes another function f_1 on an input x and then outputs $f_1(x)$ together with a new encryption of x under a different public key for the FE scheme. The new encryption of x could be used to compute a second function $f_2(x)$ and an encryption of x under yet another public key. This chain could potentially repeat and its benefit is that it allows us to compute multiple functions on x (and overcome the single-key property). However, this approach allows only a very small number of iterations because, in order to produce one bit of output from* $\mathsf{FE.Dec}$*, the ciphertexts output by* $\mathsf{FE.Enc}$ *are polynomial in κ. To obtain an* FE *ciphertext as result of* $\mathsf{FE.Dec}$*, one needs to have started with ciphertexts of size quadratic in the first polynomial. If we want to chain the scheme q times, the original ciphertext must have been exponential in q.*

3.1 Construction

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.

From Claim 2.5, the existence of a secure single-key ABE scheme implies the existence of a two-outcome single-key ABE scheme, which we denote ABE_2. Let $\lambda = \lambda(\kappa)$ be the length of the ciphertexts in the FHE scheme (both from encryption and evaluation). The construction of 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 two-outcome ABE scheme λ times:

$$(\mathsf{fmpk}_i, \mathsf{fmsk}_i) \leftarrow \mathsf{ABE}_2.\mathsf{Setup}(1^\kappa) \text{ for } i \in [\lambda].$$

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, recall that $\mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi_1, \ldots, \psi_n)$ is 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_2 for the functions $\mathsf{FHE.Eval}^i_f$ (under the different master secret keys) to construct secret keys:

$$\mathsf{fsk}_i \leftarrow \mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{fmsk}_i, \mathsf{FHE.Eval}^i_f) \text{ for } i \in [\lambda].$$

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_i^b for $i \in [\lambda]$ and $b \in \{0,1\}$. Namely,
$$\left(\Gamma, \{L_i^0, L_i^1\}_{i=1}^{\lambda}\right) \leftarrow \mathsf{Gb.Garble}(1^\kappa, \mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)),$$
where Γ is the garbled circuit and the L_i^b are the input labels.

3. Produce encryptions $c_1, \ldots, c_\lambda$ using the ABE_2 scheme:
$$c_i \leftarrow \mathsf{ABE}_2.\mathsf{Enc}\left(\mathsf{fmpk}_i,\ (\mathsf{hpk}, \psi), L_i^0, L_i^1\right) \text{ for } i \in [\lambda],$$
where (hpk, ψ) comes from the first step, and the labels (L_i^0, L_i^1) 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_2 decryption algorithm on the ciphertexts $c_1, \ldots, c_\lambda$ to recover the labels for the garbled circuit. In particular, let
$$L_i^{d_i} \leftarrow \mathsf{ABE}_2.\mathsf{Dec}(\mathsf{fsk}_i, c_i) \text{ for } i \in [\lambda],$$
where d_i is equal to $\mathsf{FHE.Eval}_f^i(\mathsf{hpk}, \psi)$.

2. Now, armed with the garbled circuit Γ and the labels $L_i^{d_i}$, run the garbled circuit evaluation algorithm to compute
$$\mathsf{Gb.Eval}(\Gamma, L_1^{d_1}, \ldots, L_\lambda^{d_\lambda}) = \mathsf{FHE.Dec}(\mathsf{hsk}, d_1 d_2 \ldots d_\lambda) = f(x).$$

3.2 Proof

We now proceed to prove Theorem 3.1 by proving that the theorem holds for our construction above.

Proof of Theorem 3.1. We first argue correctness.

Claim 3.8. *The above scheme is a correct functional encryption scheme (Def. 2.12).*

Proof. Let us examine the values we obtain in $\mathsf{FE.Dec}(\mathsf{fsk}_f, c_1, \ldots, c_\lambda, \Gamma)$. In Step (1), by the correctness of the ABE_2 scheme used, d_i is the i-th bit of $\mathsf{FHE.Eval}_f(\mathsf{hpk}, \psi)$.

Therefore, the inputs to the garbled circuit Γ in Step (2) are the labels corresponding to $\mathsf{FHE.Eval}_f(\mathsf{hpk}, \psi)$. By the correctness of the FHE scheme, decrypting $\mathsf{FHE.Eval}_f(\mathsf{hpk}, \psi)$ results in $f(x)$. Finally, by the correctness of the garbling scheme, the FHE ciphertext gets decrypted correctly, yielding $f(x)$ as the output of $\mathsf{FE.Dec}$. □

We now prove the succinctness property which follows directly from our construction. The output of $\mathsf{FE.Enc}$ consists of λ ABE_2 ciphertexts and a garbled circuit. First, λ equals $\mathsf{ctsize}_{\mathsf{FHE}}$. Second, each ABE_2 ciphertext consists of two ABE ciphertexts generated by $\mathsf{ABE.Enc}$ on input $n\mathsf{ctsize}_{\mathsf{FHE}} + \mathsf{pksize}_{\mathsf{FHE}}$ bits. The labels of the garbled circuit are $\mathrm{poly}(\kappa)$ in size. Third, the garbled circuit is the output of $\mathsf{Gb.Garble}$ so its size is polynomial in the size of the input circuit, which in turn is polynomial in $\mathsf{sksize}_{\mathsf{FHE}}$ and $\mathsf{ctsize}_{\mathsf{FHE}}$. Therefore, overall, we obtain $2\mathsf{ctsize}_{\mathsf{FHE}} \cdot \mathsf{ctsize}_{\mathsf{ABE}}(n\,\mathsf{ctsize}_{\mathsf{FHE}} + \mathsf{pksize}_{\mathsf{FHE}}) + \mathrm{poly}(\kappa, \mathsf{sksize}_{\mathsf{FHE}}, \mathsf{ctsize}_{\mathsf{FHE}})$.

We can thus see that if FHE and ABE produce ciphertexts independent of the circuit size, then so will our functional encryption scheme.

We focus on the full security case: namely, assuming ABE_2 is fully secure, we show that the resulting FE scheme is fully secure. We then discuss the proof for the selective case.

For full security, we construct a p.p.t. simulator S that achieves Def. 2.13. 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. 2.13 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 Def. 2.7) 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 $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$ 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_2 scheme in the following way. Let

$$\tilde{c}_i \leftarrow \mathsf{ABE}_2.\mathsf{Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \hat{0}), \tilde{L}_i, \tilde{L}_i\right),$$

   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. 2.13), we define a sequence of hybrid experiments, and then invoke the security definitions of the underlying schemes (FHE, garbled circuit, and ABE_2 respectively) to show that the outcome of the hybrid experiments are computationally indistinguishable.

Hybrid 0 is the output of the ideal experiment from Def. 2.13 for our FE construction with simulator S. We denote it $\mathsf{Exp}_{\mathsf{FE},A}^{H_0}$ (= $\mathsf{Exp}_{\mathsf{FE},A,S}^{\mathsf{ideal}}$).

Hybrid 1 ($\mathsf{Exp}_{\mathsf{FE},A}^{H_1}$) 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:

$$\tilde{c}_i^{(1)} \leftarrow \mathsf{ABE}_2.\mathsf{Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), \tilde{L}_i, \tilde{L}_i\right),$$

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 ($\mathsf{Exp}^{H_2}_{\mathsf{FE},A}$) 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 $d_i = \mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi)$. In Step (3), include L^{d_i} twice in the ABE encryption; namely:

$$\tilde{c}_i^{(2)} \leftarrow \mathsf{ABE}_2.\mathsf{Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), L_i^{d_i}, L_i^{d_i}\right), \text{ and}$$

$$\tilde{c}^{(2)} = (\tilde{c}_1^{(2)}, \ldots, \tilde{c}_\lambda^{(2)}, \Gamma).$$

Hybrid 3 ($\mathsf{Exp}^{H_3}_{\mathsf{FE},A}$) is the output of the real experiment from Def. 2.13 for our FE construction.

We prove each pair of consecutive hybrids to be computationally indistinguishable in the following three lemmas, Lemmas 3.9, 3.10, and 3.11.

Lemma 3.9. *Assuming* FHE *is IND-CPA–secure, Hybrid 0 and Hybrid 1 are computationally indistinguishable.*

Proof. We proceed by contradiction. We assume that there exist p.p.t. adversaries $A = (A_1, A_2)$ and a p.p.t. distinguisher D such that D (with A) can distinguish between Hybrid 0 and Hybrid 1 above. Namely, there exists a polynomial $p(\cdot)$ such that, for infinitely many κ,

$$|\Pr[D(\mathsf{Exp}^{H_0}_{\mathsf{FE},A}(1^\kappa)) = 1] - \Pr[D(\mathsf{Exp}^{H_1}_{\mathsf{FE},A}(1^\kappa)) = 1]| \geq 1/p(\kappa). \tag{1}$$

We construct a p.p.t. adversary $R = (R_1, R_2)$ that can break the semantic security of FHE. Adversary R_1 outputs an n-bit value x for some n, and adversary R_2 receives as input either homomorphic encryption of x or of 0^n, and it will distinguish between these two. Distinguishing successfully implies that there is an adversary that can distinguish successfully in Def. 2.5, by a standard hybrid argument.

To determine x, adversary R_1 works as follows:

1. Run $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},A,S}(1^\kappa)$ (Def. 2.13) from Step (1) to Step (4) and let x be the output of A_2 in Step (4).
2. Output x.

To distinguish between encryption of x or 0^n, adversary R_2 receives input hpk^*, the FHE public key, and an encryption E^* of x or 0^n and works as follows:

1. Run a modified algorithm of S by using hpk^* instead of generating fresh FHE keys and using E^* instead of encrypting 0^n. Namely:
   (a) Generate $\left(\tilde{\Gamma}, \{\tilde{L}_i\}_{i=1}^{\lambda}\right)$ as in Step (2) of S.
   (b) Output $c^* = (c_1^*, \ldots, c_\lambda^*)$ for $c_i^* = \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}_i, ((\mathsf{hpk}^*, E^*), \tilde{L}_i, \tilde{L}_i))$.
2. Feed $(c^*, \tilde{\Gamma})$ to D and output the decision of D.

Notice that if E^* is encryption of 0^n, R_2 simulates Hybrid 0 perfectly; when E^* is encryption of x, R_2 simulates Hybrid 1 perfectly. Therefore, D must have a probability of distinguishing between the two cases of at least $1/p(\kappa)$ (Eq. (1)); moreover, whenever D distinguishes correctly, R also outputs the correct decision. Therefore:

$$\begin{aligned}&|\Pr[x \leftarrow R_1(1^\kappa); (\mathsf{hsk}^*, \mathsf{hpk}^*) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa) : R_2(\mathsf{hpk}^*, \mathsf{FHE.Enc}(\mathsf{hpk}^*, x)) = 1] -\\ &\ \Pr[(\mathsf{hsk}^*, \mathsf{hpk}^*) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa) : R_2(\mathsf{hpk}^*, \mathsf{FHE.Enc}(\mathsf{hpk}^*, 0^n)) = 1]| =\\ &|\Pr[D(\mathsf{Exp}^{H_0}_{\mathsf{FE},A}(1^\kappa)) = 1] - \Pr[D(\mathsf{Exp}^{H_1}_{\mathsf{FE},A}(1^\kappa)) = 1]| \geq 1/p(\kappa),\end{aligned}$$

which contradicts the IND-CPA security of the FHE scheme. □

Lemma 3.10. *Assuming the garbled circuit is circuit- and input-private (Def. 2.7), Hybrid 1 and Hybrid 2 are computationally indistinguishable.*

Proof. We proceed by contradiction. Assume there exist p.p.t. adversaries $A = (A_1, A_2)$ and a p.p.t. distinguisher D such that D (with A) can distinguish Hybrid 1 and Hybrid 2 above. Namely, there exists a polynomial p such that, for infinitely many κ,

$$|\Pr[D(\mathsf{Exp}^{H_1}_{\mathsf{FE},A}(1^\kappa)) = 1] - \Pr[D(\mathsf{Exp}^{H_2}_{\mathsf{FE},A}(1^\kappa)) = 1]| \geq 1/p(\kappa). \tag{2}$$

We construct a stateful p.p.t. adversary $R = (R.A, R.D)$ that can break the security of the garbling scheme from Def. 2.7. The adversary $R.A$ has to provide a circuit G and an input I and then $R.D$ needs to distinguish between the simulated and the real garbled circuits and input encodings.

The adversary $R.A$ computes I and G as follows.

1. Run Steps (1)–(4) from Def. 2.13, which are the same in Hybrid 1 and Hybrid 2 and obtain f from A_1 and x from A_2.
2. Generate $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$ and let $\psi \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, x)$.
3. Output $G(\cdot) := \mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$ and $I := \mathsf{FHE.Eval}_f(\mathsf{hpk}, \psi)$ and the following state for $R.D$: $\alpha = (\psi, \mathsf{fmpk}_i, \mathsf{hpk})$.

The adversary $R.D$ receives as input a garbled circuit Γ^* and a set of labels, one for each i: $\{L^*_i\}^\lambda_{i=1}$. These could be outputs of either $\mathsf{Sim}_{\mathsf{Garble}}$ or of $\mathsf{Gb.Garble}/\mathsf{Gb.Enc}$ and $R.D$ decides which is an output of as follows:

1. Compute $c^* = \big(\{\mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}_i, ((\mathsf{hpk}, \psi), L^*_i, L^*_i))\}^\lambda_{i=1}, \Gamma^*\big)$.
2. Run D on c^* and output what D outputs.

Notice that if $(\Gamma^*, \{L^*_i\}^\lambda_{i=1})$ are outputs of $\mathsf{Sim}_{\mathsf{Garble}}$, R simulates Hybrid 1 perfectly; when $(\Gamma^*, \{L^*_i\}^\lambda_{i=1})$ are outputs of the real garbling scheme, R simulates Hybrid 2 perfectly. Therefore, the probability that D distinguishes between the two cases at least is $1/p(\kappa)$ (Eq. (2)); moreover, whenever D distinguishes correctly, R also outputs the correct decision. Therefore:

$$\begin{aligned}&|\Pr[(G, I) \leftarrow R.A(1^\kappa) : R.D(\tilde{\Gamma}, \{\tilde{L}_i\}^\lambda_{i=1}) = 1] - \Pr[(G, I) \leftarrow R.A(1^\kappa) : R.D(\Gamma, \{L_i\}^\lambda_{i=1}) = 1]| =\\ &|\Pr[D(\mathsf{Exp}^{H_1}_{\mathsf{FE},A}(1^\kappa)) = 1] - \Pr[D(\mathsf{Exp}^{H_2}_{\mathsf{FE},A}(1^\kappa)) = 1]| \geq 1/p(\kappa),\end{aligned}$$

where, $(\tilde{\Gamma}, \{\tilde{L}_i\}^\lambda_{i=1})$ are outputs of $\mathsf{Sim}_{\mathsf{Garble}}$ and $(\Gamma, \{L_i\}^\lambda_{i=1})$ are outputs of $\mathsf{Gb.Garble}/\mathsf{Gb.Enc}$. This relation contradicts the security of the garbling scheme Def. 2.7. □

Lemma 3.11. *Assuming the underlying* ABE_2 *scheme is fully secure, Hybrid 2 and Hybrid 3 in the fully secure setting above are computationally indistinguishable.*

Proof. In Hybrid 2 and Hybrid 3, there are λ ABE_2 encryptions, each with a pair of independent ABE_2 keys. First, we would like to prove that if Hybrid 2 and Hybrid 3 are computationally indistinguishable with only one of these encryptions, then they are computationally indistinguishable with λ encryptions. This would enable us to focus on only one ABE_2 ciphertext for the proof.

The argument proceeds in a standard way with a set of sub-hybrids, one for each index $i = 0 \dots \lambda$. The argument is straightforward because $\tilde{c}_i$ and $\tilde{c}_j$ (for $i \neq j$) use independently generated keys and the values encrypted with these keys are known to R. Hence, we present the hybrid argument briefly. Sub-hybrid 0 corresponds to Hybrid 2 and sub-hybrid λ corresponds to Hybrid 3. Sub-hybrid i has the first i ciphertexts as in Hybrid 2 and the rest $\lambda - i$ as in Hybrid 3.

If an adversary A can distinguish between sub-hybrids $i-1$ and i, for some i, then he can distinguish Hybrid 2 and Hybrid 3 for only one pair of ciphertexts (c_i^2, c_i^3); the reason is that we can build an adversary B: B places the challenge ciphertext in slot i of the challenge to A and produces the ciphertexts for all other slots $j \neq i$ with the correct distribution; B can do so because these ciphertexts are encrypted with fresh ABE_2 keys and B has all the information it needs to generate them correctly.

Now we are left to prove that Hybrid 2 and Hybrid 3 are indistinguishable when there is only one ciphertext, say the ℓ-th ciphertext. Namely, we need to prove that:

$$\left\{(\mathsf{state}'_A, \tilde{c}_\ell^{(2)}) \leftarrow \mathsf{Exp}_{\mathsf{FE},A}^{H_2}(1^\kappa)\right\} \stackrel{c}{\approx} \left\{(\mathsf{state}'_A, \tilde{c}_\ell^{(3)}) \leftarrow \mathsf{Exp}_{\mathsf{FE},A}^{H_3}(1^\kappa)\right\}. \tag{3}$$

We prove this statement by contradiction. Assume there exist p.p.t. adversaries $A = (A_1, A_2)$ and distinguisher D that can distinguish the distributions in (3); namely, there exists a polynomial $p(\cdot)$ such that, for infinitely many κ,

$$|\Pr[D(\mathsf{Exp}_{\mathsf{FE},A}^{H_2}(1^\kappa)) = 1] - \Pr[D(\mathsf{Exp}_{\mathsf{FE},A}^{H_3}(1^\kappa)) = 1]| \geq 1/p(\kappa). \tag{4}$$

We construct a p.p.t. adversary $R = (R_1, R_2, R_3)$ that breaks the security of ABE_2 from Def. 2.11. R_1, R_2 and R_3 send state to each other as in Def. 2.11, but for simplicity we will not denote this explicitly. R_3 aims to guess b in this definition.

Intuition. A and D can distinguish between Hybrid 2 and Hybrid 3. The only difference between these hybrids is that $\tilde{c}_\ell$ contains encryption of $(L_\ell^{d_\ell}, L_\ell^{d_\ell})$ versus $(L_\ell^{d_\ell}, L_\ell^{1-d_\ell})$. However, the ABE_2 scheme does not decrypt $L_\ell^{1-d_\ell}$ by the definition of d_ℓ, so its security hides the value of $L_\ell^{1-d_\ell}$. Since A and D can distinguish between these hybrids, they must be breaking the security of ABE_2. Therefore, R will use L_ℓ^ℓ and $L_\ell^{1-\ell}$ as part of its answers to C and then use D to distinguish its challenge.

Specifically, the adversary R_1 receives as input fmpk^* in Step 2 of Def. 2.11 and computes P as follows:

1. Interact with adversary A_1 by running Steps (1)–(2) from Defs. 2.13 as follows.
   (a) Let $\mathsf{fmpk}_\ell := \mathsf{fmpk}^*$. Generate the rest of ABE_2 keys using the $\mathsf{ABE}_2.\mathsf{Setup}$ algorithm: $(\mathsf{fmpk}_i, \mathsf{fmsk}_i) \leftarrow \mathsf{ABE}_2.\mathsf{Setup}(1^\kappa)$ for $i \neq \ell$.
   (b) Receive f from A_1 and output $P := \mathsf{FHE.Eval}_f^\ell$.

Adversary R_2 receives sk_P^* in Step 4 of Def. 2.11 and computes M, M_0, M_1, x_c as follows:

1. Continue interaction with A_2. To provide fsk_f to A_2, compute $\mathsf{fsk}_i \leftarrow \mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{fmsk}_i, \mathsf{FHE.Eval}_f^i)$ for $i \neq \ell$, and let $\mathsf{fsk}_\ell := \mathsf{sk}_P^*$.

2. Receive x from A_2.

3. Run the real garbled circuit generation as in Hybrid 2 and 3. Let $L_\ell^{d_\ell}$ be defined as in Hybrid 2. Provide $M := L_\ell^{d_\ell}$, $M_0 := L_\ell^{d_\ell}$ and $M_1 := L_\ell^{1-d_\ell}$.

4. Let $x_c := (\mathsf{hpk}, \psi)$ where $\psi \leftarrow (\mathsf{FHE.Enc}(\mathsf{hpk}, x_1), \ldots, \mathsf{FHE.Enc}(\mathsf{hpk}, x_n))$, the bitwise FHE encryption of x.

5. Output (M, M_0, M_1, x_c).

Adversary R_3 receives as input a challenge ciphertext c^* and decides if it corresponds to M_0 or to M_1 as follows:

1. Let $\tilde{c}_\ell := c^*$ and provide $(\mathsf{state}'_A, \tilde{c}_\ell)$ to D.

2. Output D's guess.

In order for D to distinguish (as in Eq. (4)), the input distribution to A must be the one from Hybrid 2 or 3. We can see that this is the case: if $b = 0$, R simulates Hybrid 2 perfectly, and if $b = 1$, R simulates Hybrid 3 perfectly. Moreover, whenever D distinguishes correctly, R also outputs the correct decision. Therefore, by a simple calculation, we can see that

$$\Pr[\mathsf{Exp}_{\mathsf{ABE}_2,R}(1^\kappa) = 1] \geq 1/2 + 1/2p(\kappa),$$

which contradicts the security of the ABE_2 scheme, Def. 2.11. □

Returning to the proof of our theorem, by transitivity of computational indistinguishability, we showed that Hybrid 0 (the ideal experiment) is equivalent to Hybrid 3 (the real experiment), thus concluding our proof.

Selective security. The proof for the selective case follows similarly. The simulator S and the four hybrids are the same. Lemmas 3.9 and 3.10 proceed similarly, except that R now interacts with A as in the selective FE definition Def. 2.14 rather than Def. 2.13. The argument of Lemma 3.11 is the same, except that the order of some operations changes. This lemma makes the resulting FE scheme selective if one starts from a selective ABE_2 scheme. □

4 Reusable Garbled Circuits

In this section, we show how to construct garbled circuits that can be reused; namely, a garbled circuit that can run on an arbitrary number of encoded inputs without compromising the privacy of the circuit or of the input. For this goal, we build on top of our functional encryption scheme.

The syntax and correctness of the reusable garbling schemes remains the same as the one for one-time garbling schemes (Def. 2.6). In Sec. 2.4, we provided the one-time security definition for circuit and input privacy, Def. 2.7. We begin by defining security for more than one-time usage.

Definition 4.1 (Input and circuit privacy with reusability)**.** *Let* RGb *be a garbling scheme for a family of circuits* $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$*. For a pair of p.p.t. algorithms* $A = (A_1, A_2)$ *and a p.p.t. simulator* $S = (S_1, S_2)$*, consider the following two experiments:*

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGb},A}(1^\kappa)}$**:**	$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGb},A,S}(1^\kappa)}$**:**
1: $(C, \mathsf{state}_A) \leftarrow A_1(1^\kappa)$	*1:* $(C, \mathsf{state}_A) \leftarrow A_1(1^\kappa)$
2: $(\mathsf{gsk}, \Gamma) \leftarrow \mathsf{RGb.Garble}(1^\kappa, C)$	*2:* $(\tilde{\Gamma}, \mathsf{state}_S) \leftarrow S_1(1^\kappa, 1^{\lvert C \rvert})$
3: $\alpha \leftarrow A_2^{\mathsf{RGb.Enc}(\mathsf{gsk},\cdot)}(C, \Gamma, \mathsf{state}_A)$	*3:* $\alpha \leftarrow A_2^{\mathsf{O}(\cdot,C)[[\mathsf{state}_S]]}(C, \tilde{\Gamma}, \mathsf{state}_A)$
4: Output α	*4:* Output α

In the above, $\mathsf{O}(\cdot, C)[[\mathsf{state}_S]]$ *is an oracle that on input* x *from* A_2*, runs* S_2 *with inputs* $C(x)$*,* $1^{\lvert x \rvert}$*, and the latest state of* S*; it returns the output of* S_2 *(storing the new simulator state for the next invocation).*

We say that the garbling scheme RGb *is input- and circuit-private with reusability if there exists a p.p.t. simulator* S *such that for all pairs of p.p.t. adversaries* $A = (A_1, A_2)$*, the following two distributions are computationally indistinguishable:*

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

We can see that this security definition enables reusability of the garbled circuit: A_2 is allowed to make as many queries for input encodings as it wants.

From now on, by *reusable garbling scheme*, we will implicitly refer to a garbling scheme that has input and circuit privacy with reusability as in the definition above, Def. 4.1.

Remark 4.1. *We can provide an alternate syntax for a reusable garbling scheme, and we can also construct a scheme with this syntax (and a similar security definition) from our functional encryption scheme. This syntax has an additional setup algorithm (separate from the garble algorithm) that produces the secret key necessary for encoding and for circuit garbling; such a syntax would allow the garbled circuit to be generated after the encodings.*

Remark 4.2. *We do not provide a definition of authenticity because it is a straightforward extension of our scheme and is already achieved by [GVW13]. We focus on circuit and input privacy, which have not been achieved by previous work.*

Recall the class of circuits $\mathcal{C}_{n,d(n)}$ defined for Corollary 3.2.

Theorem 4.3. *There exists a polynomial* p*, such that for every depth* $d = d(n)$ *function of the input size* n*, there is a reusable garbling scheme for any class of boolean circuits* $\{\mathcal{C}_{n,d}\}_{n\in\mathbb{N}}$*, assuming there is a fully secure single-key functional encryption scheme for any class of boolean circuits* $\{\mathcal{C}_{n,p(d)}\}_{n\in\mathbb{N}}$*.*

Corollary 4.4 (The LWE Instantiation)**.** *For every integer* $n \in \mathbb{N}$*, polynomial function* $d = d(n)$*, there is a reusable garbling scheme for the class* $\mathcal{C}_{n,d(n)}$*, under the following assumption: 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 time* $2^{O(\ell^\epsilon)}$ *in the worst case.*

The proof of this corollary follows from Theorem 4.3 when instantiating the functional encryption scheme with the one from Corollary 3.2.

Denote by *universal reusable garbling scheme*, a reusable garbling scheme for the class of all polynomial-sized circuits. Then, the following corollary follows directly from Theorem 4.3:

Corollary 4.5 (Universal reusable garbled circuits)**.** *If there is a universal single-key fully secure functional encryption scheme, there is a universal reusable garbling scheme.*

Notice that our functional encryption tool (FE) already gives reusable garbled circuits with input privacy but no circuit privacy: the garbling of C is $\mathsf{FE.KeyGen}(\mathsf{fmsk}, C)$, whereas the encoding of the input x is $\mathsf{FE.Enc}(\mathsf{fmpk}, x)$. The fact that our scheme is single-key does not pose a limitation because the single-key corresponds to the circuit to garble (and any input encoding need only work with one garbled circuit). Since the single-key for one function works with an arbitrary number of encrypted inputs, the resulting garbled circuit is reusable.

However, the problem is that FE does not hide the circuit C, which is a required property of garbling schemes. The insight in achieving circuit privacy is to use the input-hiding property of the FE scheme to hide the circuit as well. The first idea that comes to mind is to hide C by including it in the ciphertext together with the input x. Specifically, instead of providing a key for circuit C, the encryptor runs FE.KeyGen on a universal circuit U that on input (C, x) computes $C(x)$. Notice that U can be public because it carries no information about C other than its size. Now the encryption of x consists of an encryption of (C, x) using FE.Enc. In this way, we can see that the resulting garbled circuit satisfies the correctness property. Moreover, for security, FE hides the input (C, x) so it would hide the circuit C as well.

Nevertheless, this approach is not useful because the encoding is as large as the circuit C (in particular, RGb.Enc no longer satisfies the efficiency property in Def. 2.6). Moreover, in this case, the standard one-time garbling schemes would be enough because one could produce a fresh garbled circuit with each ciphertext.

To overcome this problem, the idea is to provide, together with the ciphertext of x, *the ability to decrypt* C rather than the entire description of C. Specifically, let E be the encryption of the circuit C with a semantically secure symmetric encryption scheme under a secret key sk. The garbling of C consists of running the key generation FE.KeyGen on a circuit U_E that includes E and works as follows. On input (x, sk) the circuit U_E decrypts E to obtain C, and outputs the result of running C on x. Even though $\mathsf{FE.KeyGen}(\mathsf{fmsk}, U_E)$ does not hide U_E, the description of U_E does not leak C because C is encrypted. An encoding by RGb.Enc of x thus consists of running the encryption algorithm FE.Enc on (x, sk).

4.1 Construction

We construct a reusable garbling scheme RGb = (RGb.Garble, RGb.Enc, RGb.Eval) as follows. Let E = (E.KeyGen, E.Enc, 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 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 [BGI+01, GK05] (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 RGb.Garble is public key. We prove in Sec. 4.3 that this is also not possible based on the impossibility result of [AGVW12]; hence, with regard to public versus private key, our reusable garbling result is tight.

4.2 Proof

Proof of Theorem 4.3. We first argue the scheme satisfies the correctness and efficiency properties in Def. 2.6.

Claim 4.6. *The above scheme* RGb *is a correct and efficient garbling scheme.*

Proof. We can easily see correctness of RGb.Eval:

$$\begin{aligned} \mathsf{RGb.Eval}(\Gamma, c_x) &= \mathsf{FE.Dec}(\Gamma, c_x) \quad \text{(by the definition of RGb.Eval)} \\ &= U_E(\mathsf{sk}, x) \qquad \text{(by the correctness of FE)} \\ &= C(x) \text{ (by the definition of } U_E). \end{aligned}$$

The efficiency of RGb depends on the efficiency of the FE.Enc algorithm and the length of gsk depends on the FE.Setup. If the runtime of FE.Enc does not depend on the class of circuits to be computed at all, the same holds for RGb.Enc's efficiency. If FE.Enc and FE.Setup depend on the depth of the circuits to be computed, as is the case in our LWE instantiation, RGb.Enc's runtime and $|\mathsf{gsk}|$ also depend on the depth of the circuits, but still remain independent of the size of the circuits, which could potentially be much larger. □

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.

To prove security, we need to construct a simulator $S = (S_1, S_2)$ satisfying Def. 4.1, assuming there is a simulator $\mathsf{Sim}_{\mathsf{FE}}$ that satisfies Def. 2.13.

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 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 $\mathsf{Sim}_{\mathsf{FE}}$ of the FE scheme and outputs

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

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 $\mathsf{Sim}_{\mathsf{FE}}$ may not simulate correctly. However, this is not a problem because, by semantic security, E and $\tilde{E}$ are computationally indistinguishable so $\mathsf{Sim}_{\mathsf{FE}}$ must work correctly, otherwise it breaks semantic security of the encryption scheme E.

We now prove formally that the simulation satisfies Def. 4.1 for any adversary $A = (A_1, A_2)$. Let us assume that the α output of A_2 is its view, namely, all the information A_2 receives in the protocol, $(C, \mathsf{state}_A, \Gamma, \{x_i, c_{x_i}\}_{i=1}^t)$. If the outcome of the real and ideal experiments are computationally indistinguishable in this case, then they are computationally indistinguishable for any other output strategy of A_2 because D can always run A_2 on its view since A_2 is p.p.t.. Therefore, we would like to show that:

$$\left\{(C, \mathsf{state}_A, \Gamma, \{x_i, c_{x_i}\}_{i=1}^t) \leftarrow \mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGb},A}(1^\kappa)\right\}_\kappa \overset{c}{\approx}$$
$$\left\{(C, \mathsf{state}_A, \tilde{\Gamma}, \{x_i, \tilde{c}_{x_i}\}_{i=1}^t) \leftarrow \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGb},A,S}(1^\kappa)\right\}_\kappa.$$

Game 0: The ideal game of Def. 4.1 with simulator S; we recall that the output distribution in this case is

$$\big(C, \mathsf{state}_A, \mathsf{FE.KeyGen}(\mathsf{fmsk}, U_{\tilde{E}}), \{x_i, \mathsf{Sim}_{\mathsf{FE}}(\mathsf{fmpk}, \mathsf{fsk}_{U_{\tilde{E}}}, U_{\tilde{E}}, C(x_i), 1^{|x_i|+|\mathsf{sk}|})\}_{i=1}^t\big).$$

Game 1: The same as Game 0, but $\tilde{E}$ is replaced with $E = \mathsf{E.Enc}(\mathsf{sk}, C)$. That is, the output distribution is

$$\big(C, \mathsf{state}_A, \Gamma, \{x_i, \mathsf{Sim}_{\mathsf{FE}}(\mathsf{fmpk}, \mathsf{fsk}_{U_E}, U_E, C(x_i), 1^{|x_i|+|\mathsf{sk}|})\}_{i=1}^t\big).$$

Game 2: The real game with our construction for RGb. It consists of the output distribution

$$\big(C, \mathsf{state}_A, \Gamma, \{x_i, c_{x_i}\}_{i=1}^t\big).$$

First, let us argue that the distributions output by Game 0 and Game 1 are computationally indistinguishable. Note that these two distributions differ only in E and $\tilde{E}$. Since these distributions do not contain sk or any other function of sk other than $E/\tilde{E}$, by semantic security of the encryption scheme, we can show these two distributions are computationally indistinguishable. Finally, Lemma 4.7 proves that the outputs of Game 1 and Game 2 are also computationally indistinguishable, which concludes our proof.

Lemma 4.7. *Assuming* FE *is* $\mathsf{FULL\text{-}SIM}$*-secure, the outputs of Game 1 and Game 2 are computationally indistinguishable.*

Proof. The proof of the lemma is by contradiction. We assume there exist p.p.t. adversaries $A = (A_1, A_2)$ and p.p.t. distinguisher D such that D with A can distinguish Game 1 and Game 2. Namely, there exists a polynomial $p(\cdot)$ such that, for infinitely many κ,

$$|\Pr[D(\mathsf{Exp}^{\mathrm{Game1}}_{\mathsf{FE},A}(1^\kappa)) = 1] - \Pr[D(\mathsf{Exp}^{\mathrm{Game2}}_{\mathsf{FE},A}(1^\kappa)) = 1]| \geq 1/p(\kappa). \quad (5)$$

We construct adversaries that break the full security of the functional encryption scheme Def. 2.13. We call these adversaries $A^{\mathsf{FE}} = (A_1^{\mathsf{FE}}, A_2^{\mathsf{FE}})$ and D^{FE} using the "FE" superscript to differentiate them from the adversaries distinguishing Game 1 and 2. In fact, we construct adversaries A^{FE} and D^{FE} that break a modified version of Def. 2.13: the modification is that A^{FE} can repeat Steps (4–5) as many times as it wishes and adaptively; more precisely, for the i-th repetition of Steps (4–5), A_2^{FE} can ask for an encryption of an input x_i where x_i could be determined based on the previous values and encryptions of $x_1, \ldots, x_{i-1}$; A_2^{FE} receives either a real encryption or a simulated encryption as in Step (5), but either all encryptions are real or all are simulated. We can see that if A^{FE} and D^{FE} break this modified definition, then they must break the original definition (with a polynomially smaller advantage): this implication follows from a standard hybrid argument possible because the encryption of x_i is public key.

On input fmpk, adversary A_1^{FE} works as follows:

1. Run A_1 on input 1^κ and obtain C and state_A.
2. Choose $\mathsf{sk} \leftarrow \mathsf{E.KeyGen}(1^\kappa)$, encrypt $E \leftarrow \mathsf{E.Enc}(\mathsf{sk}, C)$, and let U_E be the circuit described above.
3. Output function U_E and $\mathsf{state}_A^{\mathsf{FE}} := (\mathsf{sk}, U_E, \mathsf{state}_A)$.

On input $(\mathsf{fsk}_{U_E}, \mathsf{state}_A^{\mathsf{FE}})$, adversary A_2^{FE} works as follows:

1. Let $\Gamma := \mathsf{fsk}_{U_E}$.
2. Run A_2 on U_E, Γ and state_A by answering to its oracle queries as follows.
   (a) Consider the i-th oracle query (x_i, state_A). Output (x_i, sk).
   (b) Receive as input CT_i which is either the real ciphertext $c_i \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, (x_i, \mathsf{sk}))$ or the simulated ciphertext $\tilde{c}_i \leftarrow \mathsf{Sim}_{\mathsf{FE}}(\mathsf{fmpk}, \Gamma, U_E, C(x_i), 1^{|x_i|+|\mathsf{sk}|})$. Respond to A_2 with $(\mathsf{CT}_i, \mathsf{state}_A)$.
   (c) Repeat these steps until A_2 finishes querying for encodings, and outputs α.
3. Output α.

Adversary D^{FE} is the same as D.

When the encodings CT_i are the ideal ciphertexts, we can see that $(A_1^{\mathsf{FE}}, A_2^{\mathsf{FE}})$ simulate perfectly Game 1; hence

$$\Pr[D^{\mathsf{FE}}(\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},A^{\mathsf{FE}}}(1^\kappa)) = 1] = \Pr[D(\mathsf{Exp}^{\text{Game 1}}_{\mathsf{FE},A}(1^\kappa)) = 1].$$

When the encodings CT_i are the real ciphertexts, $(A_1^{\mathsf{FE}}, A_2^{\mathsf{FE}})$ simulate perfectly Game 2 and thus

$$\Pr[D^{\mathsf{FE}}(\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A^{\mathsf{FE}}}(1^\kappa)) = 1] = \Pr[D(\mathsf{Exp}^{\text{Game 2}}_{\mathsf{FE},A}(1^\kappa)) = 1].$$

By Eq. (5), we have

$$|\Pr[D^{\mathsf{FE}}(\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},A^{\mathsf{FE}}}(1^\kappa)) = 1] - \Pr[D^{\mathsf{FE}}(\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A^{\mathsf{FE}}}(1^\kappa)) = 1]| \geq 1/p(\kappa),$$

which contradicts FULL-SIM-security of FE. □

Having proved that Game 0 and Game 1 are computationally indistinguishable, and that Game 1 and Game 2 are computationally indistinguishable, we conclude that Game 0 and Game 2 are computationally indistinguishable, and therefore that garbling scheme RGb is input- and circuit-private with reusability. □

4.3 Impossibility of Public-Key Reusable Garbled Circuits

In this section, we show that a public-key reusable garbling scheme is impossible. Our argument is at a high level because it follows from existing results straightforwardly.

A public-key reusable garbling scheme would have the following syntax:

Definition 4.2 (Public-key garbling scheme)**.** *A public-key garbling scheme* PubGb *for the class of circuits* $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$*, with* $\mathcal{C}_n$ *a set of boolean circuits taking* n *bits as input, is a tuple of p.p.t. algorithms (*$\mathsf{PubGb.Setup}$*,* $\mathsf{PubGb.Garble}$*,* $\mathsf{PubGb.Enc}$*,* $\mathsf{PubGb.Eval}$*) such that*

- $\mathsf{PubGb.Setup}(1^\kappa)$*: Takes as input the security parameter* 1^κ *and outputs a secret key* gsk *and a public key* gpk*.*
- $\mathsf{PubGb.Garble}(\mathsf{gpk}, C)$*: Takes as input a public key* gpk *and a circuit* C*, and outputs the garbled circuit* Γ *of the circuit* C*.*
- $\mathsf{PubGb.Enc}(\mathsf{gsk}, x)$*: Takes as input the secret key* gsk *and an input* x*, and outputs an encoding* c_x*.*
- $\mathsf{PubGb.Eval}(\Gamma, c_x)$*: Takes as input a garbled circuit* Γ *and an encoding* c_x *and outputs a value* y*.*

Correctness. *For all polynomials* $n(\cdot)$*, for all sufficiently large security parameters* κ*, for* $n = n(\kappa)$*, for all circuits* $C \in \mathcal{C}_n$*, and for all* $x \in \{0,1\}^n$*,*

$$\Pr[(\mathsf{gsk}, \mathsf{gpk}) \leftarrow \mathsf{PubGb.Setup}(1^\kappa); \Gamma \leftarrow \mathsf{PubGb.Garble}(\mathsf{gpk}, C); c_x \leftarrow \mathsf{PubGb.Enc}(\mathsf{gsk}, x) : \\ \mathsf{PubGb.Eval}(\Gamma, c_x) = C(x)] = 1 - \mathrm{negl}(\kappa).$$

The natural security definition of circuit-private definition of this new scheme is similar in flavor to Def. 2.13, but we do not elaborate. (In fact, this definition can be relaxed to not require input privacy for the impossibility result to still hold.)

The first step in the impossibility argument is to note that the syntax and correctness of a public-key garbling scheme is the same as the syntax of a functional encryption scheme (Def. 2.12) with the following correspondence of algorithms: $\mathsf{PubGb.Setup}$ corresponds to $\mathsf{FE.Setup}$, $\mathsf{PubGb.Garble}$ corresponds to the encryption algorithm $\mathsf{FE.Enc}$, $\mathsf{PubGb.Enc}$ corresponds to $\mathsf{FE.KeyGen}$ and $\mathsf{PubGb.Eval}$ corresponds to $\mathsf{FE.Dec}$. Note that $\mathsf{PubGb.Enc}$ does not correspond to $\mathsf{FE.Enc}$ but to $\mathsf{FE.KeyGen}$ because $\mathsf{PubGb.Enc}$ is a secret key algorithm and $\mathsf{FE.Enc}$ is a public-key algorithm. Therefore, an encoding of an input x in the reusable garbling scheme corresponds to a secret key for a function f_x in the functional encryption scheme.

Moreover, considering this mapping, it is straightforward to show that a circuit-private public-key garbling scheme implies a secure functional encryption scheme. Since the reusable garbling scheme allows an arbitrary number of inputs being encoded, it implies that the functional encryption scheme can generate an arbitrary number of secret function keys sk_{f_x}; furthermore, in this functional encryption scheme, the size of the ciphertexts does not depend on the number of keys generated (because this number if nowhere provided as input in the syntax of the scheme). This conclusion directly contradicts the recent impossibility result of Agrawal et al. [AGVW12]: they show that any functional encryption scheme that can securely provide q keys must have the size of the ciphertexts grow in q; therefore, a reusable circuit-private public-key garbling scheme is unachievable.

5 Token-Based Obfuscation

Following the discussion of obfuscation in Sec. 1.1.2, the purpose of this section is to cast reusable garbled circuits in the form of obfuscation and to show that this provides a new model for obfuscation, namely *token-based obfuscation*.

Reusable garbled circuits come close to obfuscation: a reusable garbled circuit hides the circuit while permitting circuit evaluation on an arbitrary number of inputs. While they come close, reusable garbled circuits do not provide obfuscation, because the encoding of each input requires knowledge of the secret key: namely, to run an obfuscated program on an input, one needs to obtain a token for the input from the obfuscator. This requirement of our scheme is in fact necessary: as argued in the tightness discussion in Sec. 4, a scheme in which one can publicly encode inputs is impossible because it falls directly onto known impossibility results for obfuscation.

Therefore, we propose a new token-based model for obfuscation. The idea is for a program vendor to obfuscate his program and provide tokens representing rights to run this program on specific inputs. For example, consider the case when some researchers want to compute statistics on a database with sensitive information. The program to be obfuscated consists of the database service program with the secret database hardcoded in it, U_{DB}. When researchers want to compute statistics x, they request a token for x from the database owner. Using the obfuscated program and the token, the researchers can compute $U_{\mathsf{DB}}(x)$, the statistics result by themselves without having to contact the owner again. It is crucial that the time to compute the token for x is much smaller than the time to compute U_{DB} on x, so that the owner does not have to do a lot of work. We also note, that in certain cases, one has to anyways request such a token from the owner for other reasons: for example, the database owner can check that the statistics the researchers want to compute is not too revealing and grant a token only if this is the case.

Let us compare the token-based obfuscation model with the obfuscation model resulting from using FHE. With FHE, the obfuscation of a program is the FHE encryption of the program. When the client wants to feed an input to the obfuscated program, the client can encrypt this input by herself using the FHE public-key and does not need to obtain a token from the obfuscator. To run the program, the client performs FHE evaluation of a universal circuit on the encrypted program and the encrypted input, thus obtaining an encrypted result. The client cannot decrypt the result by herself and thus needs to contact the obfuscator for this decryption – this process consists of two messages. In our token-based model, if the obfuscator knows a priori the inputs for which to send tokens to the client (e.g., when distributing permissions for certain computations), the whole protocol consists of one message only because the client can compute and decrypt the result by herself. Another difference between these two obfuscation models is that, in the token-based model, the obfuscator needs to be available only at the beginning of the computation (when giving out tokens), whereas in the FHE model, the obfuscator has to be online at the end of the computation to decrypt the result.

5.1 Definition

We now provide the definition for token-based obfuscation and the desired simulation security. These definitions are very similar to the definitions for reusable garbled circuits (Def. 2.6 and Def. 4.1): the syntax, correctness and efficiency are the same except that garbling schemes have an additional Eval algorithm.

Definition 5.1 (Token-based Obfuscation). *A token-based obfuscation scheme for the class of circuits* $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *with* $\mathcal{C}_n : \{0,1\}^n \to \{0,1\}$ *is a pair of p.p.t. algorithms* $(\mathsf{tOB.Obfuscate}, \mathsf{tOB.Token})$ *such that*

- $\mathsf{tOB.Obfuscate}(1^\kappa, C)$*: Takes as input the security parameter* 1^κ*, and a circuit* $C \in \mathcal{C}_n$*, and outputs a secret key* osk *and the obfuscation* O *of the circuit* C*.*

- $\mathsf{tOB.Token}(\mathsf{osk}, x)$*: Takes as input the secret key* osk *and some input* $x \in \{0,1\}^n$*, and outputs* tk_x*.*

Efficiency. *The running time of* $\mathsf{tOB.Token}$ *is independent of the size of* C*.*

Correctness. *For all polynomials* $n(\cdot)$*, for all sufficiently large security parameters* κ*, if* $n = n(\kappa)$*, for all circuits* $C \in \mathcal{C}_n$*, and for all* $x \in \{0,1\}^n$*,*

$$\Pr[(\mathsf{osk}, O) \leftarrow \mathsf{tOB.Obfuscate}(1^\kappa, C); \mathsf{tk}_x \leftarrow \mathsf{tOB.Token}(\mathsf{osk}, x) : O(\mathsf{tk}_x) = C(x)] = 1 - \mathrm{negl}(\kappa).$$

Remark 5.1. *We could use an alternative definition of token-based obfuscation that separates the generation of* osk *(in an additional* $\mathsf{tOB.Setup}$ *algorithm with input the security parameter) from the* $\mathsf{tOB.Obfuscate}$ *algorithm. Such a formulation would force* osk *and thus the token computation* $\mathsf{tOB.Token}(\mathsf{osk}, x)$ *to be independent of the circuit obfuscated; moreover,* C *could be chosen later, even after all inputs* x *have been encrypted with* $\mathsf{tOB.Token}$*.*

Our construction satisfies this definition as well because it generates the secret key osk *independent of* C*. However, we did not choose such a formulation because we wanted to be consistent with the definition of obfuscation, which does not have a separate setup phase.*

Intuitively, in a secure token-based obfuscation scheme, an adversary does not learn anything about the circuit C other than $C(x)$ and the size of C.

Definition 5.2 (Secure token-based obfuscation)**.** *Let* tOB *be a token-based obfuscation scheme for a family of circuits* $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$*. For* $A = (A_1, A_2)$ *and* $S = (S_1, S_2)$*, pairs of p.p.t. algorithms, consider the following two experiments:*

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{tOB},A}(1^\kappa)}$**:**	$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{tOB},A,S}(1^\kappa)}$**:**
1: $(C, \mathsf{state}_A) \leftarrow A_1(1^\kappa)$	*1:* $(C, \mathsf{state}_A) \leftarrow A_1(1^\kappa)$
2: $(\mathsf{osk}, O) \leftarrow \mathsf{tOB.Obfuscate}(1^\kappa, C)$	*2:* $(\tilde{O}, \mathsf{state}_S) \leftarrow S_1(1^\kappa, 1^{\lvert C\rvert})$
3: $\alpha \leftarrow A_2^{\mathsf{tOB.Token}(\mathsf{osk},\cdot)}(C, O, \mathsf{state}_A)$	*3:* $\alpha \leftarrow A_2^{\mathsf{OS}(\cdot,C)[[\mathsf{state}_S]]}(C, \tilde{O}, \mathsf{state}_A)$
4: Output α	*4:* Output α

In the above, $\mathsf{OS}(\cdot, C)[[\mathsf{state}_S]]$ *is an oracle that on input* x *from* A_2*, runs* S_2 *with inputs* $C(x)$*,* $1^{|x|}$*, and the current state of* S*,* state_S*.* S_2 *responds with* tk_x *and a new state* state'_S *which* OS *will feed to* S_2 *on the next call.* OS *returns* tk_x *to* A_2*.*

We say that the token-based obfuscation tOB *is secure if there exists a pair of p.p.t. simulators* $S = (S_1, S_2)$ *such that for all pairs of p.p.t. adversaries* $A = (A_1, A_2)$*, the following two distributions are computationally indistinguishable:*

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

Note that, in this security definition, a token tk_x hides x as well because S_2 never receives x. This is usually not required of obfuscation, but we achieve this property for free.

5.2 Scheme

The construction of a token-based obfuscation scheme is very similar to the construction of reusable garbled circuits, the technical difference being minor: we need to specify how to construct the algorithm $\mathsf{tOB.Obfuscate}$ from $\mathsf{RGb.Garble}$ and $\mathsf{RGb.Eval}$. We construct a token-based obfuscation tOB =

(tOB.Obfuscate, tOB.Token) as follows based on a reusable garbled scheme RGb = (RGb.Garble, RGb.Enc, RGb.Eval). The token algorithm tOB.Token is the same as RGb.Enc.

Obfuscation tOB.Obfuscate$(1^\kappa, C \in \mathcal{C}_n)$**:**

1. Let $(\Gamma, \mathsf{sk}) \leftarrow \mathsf{RGb.Garble}(1^\kappa, C)$.
2. Construct the circuit O (the obfuscation of C) as follows. The circuit O has Γ hardcoded. It takes as input a token tk_x, computes $\mathsf{RGb.Eval}(\Gamma, \mathsf{tk}_x)$, and outputs the result.
3. Output sk as the secret key, and the description of O as the obfuscation of C.

Since the construction is essentially the same as the one of reusable garbled circuits and the security is the same, the same claims and proofs as for reusable garbled circuits hold here, based on Theorem 4.3 and Corollary 4.4. We state them here for completeness.

Claim 5.2. *Assuming a reusable garbling scheme for the class of circuits $\mathcal{C}$, there is a token-based obfuscation scheme for $\mathcal{C}$.*

Recall the class of circuits $\mathcal{C}_{n,d(n)}$ defined for Corollary 3.2.

Corollary 5.3 (The LWE Instantiation)**.** *For every integer $n \in \mathbb{N}$, polynomial function $d = d(n)$, there is a token-based obfuscation scheme for the class $\mathcal{C}_{n,d(n)}$, under the following assumption: 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 time $2^{O(\ell^\epsilon)}$ in the worst case.*

Denote by *universal token-based obfuscation scheme*, a token-based obfuscation scheme for the class of all polynomial-sized circuits. Then,

Corollary 5.4 (Universal token-based obfuscation)**.** *If there is a universal fully secure single-key functional encryption scheme, there is a universal token-based obfuscation scheme.*

6 Computing on Encrypted Data in Input-Specific Time

We initiate the study of fully homomorphic encryption where the runtime of the homomorphic evaluation is input-specific rather than worst-case time. We show how to use our functional encryption scheme to evaluate Turing machines on encrypted data in input-specific time.

Let us recall the setting of computation on encrypted data. A client gives various encrypted inputs and a function f to an evaluator. The evaluator should compute f on the encrypted inputs and return the encrypted result, while learning nothing about the inputs.

Fully homomorphic encryption has been the main tool used in this setting. It was first constructed in a breakthrough work by Gentry [Gen09] and refined in subsequent work [DGHV10, SS10b, BV11a, Vai11, BGV12, GHS12a, GHS12b]. Since then, FHE has found many great applications to various problems.

However, one of the main drawbacks of FHE is that when evaluating a Turing machine (TM) over encrypted data, the running time is at least the worst-case running time of the Turing machine over all inputs. The reason is that, one needs to transform the TM into a circuit. If t_{max} is the maximum running time of the TM on inputs of a certain size—namely, the running time on the worst-case input— then the size of the resulting circuit is at least t_{max}. Thus, even if the TM runs in a short time on most of the inputs, but for a very long time (t_{max}) on only one input, *homomorphic evaluation will still run in t_{max} for all inputs*. This property often results in inefficiency in practice; for example, consider a TM having a loop that depends on the input. For specific inputs, it can loop for a very long time, but for most inputs it does not loop at all.

As a result, researchers have tried to find input-specific schemes. A first observation is that this goal is impossible: input-specific evaluation implies that the evaluator learns the runtime of the TM on each input, which violates CPA-security of the homomorphic scheme (Def. 2.5). Hence, we must relax the security definition and allow the evaluator to learn the runtime for each input, but require that *the evaluator learns nothing else besides the running time*. This goal is not possible with FHE because the evaluator cannot decrypt any bit of information, so it cannot tell whether the computation finished or not; thus, we must look for new solutions.

A second observation is that the evaluator must no longer be able to evaluate TMs of his choice on the client's data: if he could, the evaluator would run TMs whose running times convey the value of the input x (for example, the evaluator could run $|x|$ TMs, where the i-th TM stops early if the i-th bit of x is zero, and otherwise, it stops later; in this way, the evaluator learns the exact value of x).

Based on these observations, we can see that functional encryption is the natural solution: it hides the inputs to the computation, enables the evaluator to decrypt the running time, and requires the evaluator to obtain a secret key from the client to evaluate each TM.

Due to the impossibility result for functional encryption [AGVW12] discussed in Sec. 1, the client cannot give keys for an arbitrary number of Turing machines to the evaluator. The best we can hope to achieve is for the client to provide a single key for a function to the evaluator (or equivalently, for a constant number q of keys if the client runs the scheme q times). Fortunately, the single-key restriction does not mean that the client can evaluate only one Turing machine. In fact, the client can give a key to the evaluator for a universal Turing machine U that takes as input a TM M and a value x, and outputs $M(x)$. Then, the client must specify together with each input x the TM M he wants to run on x. Such a strategy is even desirable in certain cases: the client may not want the evaluator to compute a TM on every input the client has provided and learn the running time on that input; the client may prefer to specify what inputs to run each Turing machine on.

Using our functional encryption scheme, we achieve a construction that enables computation in input-specific time. We call such a scheme *Turing machine homomorphic encryption*, or shortly TMFHE.

As discussed (Corollary 3.2), our functional encryption scheme is succinct in that the ciphertexts grow with the depth of the circuit rather than the size of the circuit. Therefore, our input-specific computation is useful only for Turing machines that can be represented in circuits whose depths are smaller than the running time – because otherwise the client would have to do a lot of work and could instead just run the Turing machine on its own. Moreover, for these machines, we cannot use the Pippenger-Fischer [PF79] transformation because the resulting circuits have depth roughly equal to the running time of the transformed machines. Specifically, our input-specific scheme makes sense for the following class of circuits, with a bound on their depth.

Definition 6.1 (d-depth-bounded class of Turing machines)**.** *A finite class of Turing machines $\mathcal{M}$ is d-depth-bounded for a function d, if there exists a class of efficiently computable transformations $\{\mathcal{T}_n\}_{n\in\mathbb{N}}$ with $\mathcal{T}_n : \mathbb{N} \to \{$all circuits$\}$ such that $\mathcal{T}_n(t) = C_{n,t}$ where $C_{n,t}$ is a circuit as follows.*

- *On input a Turing machine $M \in \mathcal{M}$ and a value $x \in \{0,1\}^n$, $C_{n,t}$ outputs $M(x)$ if M on input x stops in t steps, or $\perp$ otherwise.*
- *The depth of $C_{n,t}$ is at most $d(n)$ and the size of $C_{n,t}$ is $\tilde{O}(t)$.*

Remark 6.1. *Notice that, if we remove the depth constraint (but still keep the circuit size constraint), any finite class of Turing machines satisfies the definition because of the Pippenger-Fischer transformation* *applied to the universal circuit of this class of Turing machines. Specifically, let U_t be a universal Turing machine that runs any given machine $M \in \mathcal{M}$ for t steps. This machine has $O(t)$ running time and when applying the Pippenger-Fischer transformation [PF79] to it, we get a circuit of size $O(t \log t)$.*

We next present our construction. For completeness, we provide formal definitions and proofs of our theorems and claims in Appendix C. Our security notion (Def. C.2 in the appendix) is called *runtime-CPA security*, which straightforwardly captures the fact that the evaluator should learn nothing about the computation besides the running time.

6.1 Construction

A TMFHE scheme consists of four algorithms: TMFHE = (TMFHE.KeyGen, TMFHE.Enc, TMFHE.Eval, TMFHE.Dec). The client runs TMFHE.KeyGen once in an offline preprocessing stage. Later, in the online phase, the client sends a potentially large number of encrypted inputs to the evaluator. For every input (x, M) consisting of a value x and a Turing machine M, the client runs TMFHE.Enc to encrypt the input and then TMFHE.Dec to decrypt the result from the evaluator. The evaluator runs TMFHE.Eval to evaluate M on x homomorphically in input-specific running time. The work of the client in the offline phase is proportional to t_{max}, the worst-case input running time. However, for each input in the online phase, the client does little work (independent of the running time of M) and thus the cost is amortized.

We first provide intuition for our construction. As mentioned, we use our functional encryption scheme FE to enable the evaluator to determine at various intermediary points whether the computation finished or not. For each intermediary step, the client has to provide the evaluator with a function secret key fsk (using the FE scheme) for a function that returns a bit indicating whether the computation has finished. However, if the client provides a key for every computation step, the offline work of the client becomes quadratic in t_{max}, which can be very large in certain cases. The idea is to choose intermediary points spaced at exponentially increasing intervals. In this way, the client generates only a logarithmic number of keys, while the evaluator runs in roughly twice the time of M on an input.

As part of TMFHE.Enc, besides providing the FE encryptions for a pair (M, x), the client also provides a homomorphic encryption for x and the machine M, so that once the evaluator learns the running time of M on x, it can then perform the homomorphic computation on x in that running time.

We present our construction for a class of d-depth-bounded Turing machines. By Def. 6.1, such a class has a transformation $\mathcal{T}_n$ that enables transforming a universal TM into a circuit. Let FHE be any homomorphic encryption scheme (as defined in Sec. 2.3) for circuits of depth d and let FE be any functional encryption scheme for circuits of depth d. For simplicity, we present our scheme for Turing machines that output only one bit; we discuss in Sec. 6.3 multiple output bits and how to avoid having the output size be worst case.

Key generation TMFHE.KeyGen$(1^\kappa, 1^n, 1^{t_{\mathsf{max}}})$ takes as input the security parameter κ, an input size n, and a maximum time bound t_{max}.

1. Let $\tau = \lceil \log t_{\mathsf{max}} \rceil$. For each $i \in [\tau]$, let $D_i = \mathcal{T}_n(2^i)$ be the circuit that outputs $M(x)$ if M finishes in 2^i steps on input x or $\perp$ otherwise. Construct circuit C_i based on D_i: the circuit C_i, on input a TM M and a value x, outputs 1 if M finished in 2^i steps when running on input x or 0 otherwise; C_i is the same as circuit D_i but it just outputs whether the first output bit of C_i is non-$\perp$ or $\perp$, respectively.

2. Generate functional encryption secret keys for $C_1, \ldots, C_\tau$ by running:
$$(\mathsf{fmpk}_i, \mathsf{fmsk}_i) \leftarrow \mathsf{FE.Setup}(1^\kappa) \text{ and } \mathsf{fsk}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}_i, C_i) \text{ for } i \in [\tau].$$

3. Generate FHE keys $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$.

4. Output the tuple $\mathsf{PK} := (\mathsf{fmpk}_1, \ldots, \mathsf{fmpk}_\tau, \mathsf{hpk})$ as the public key, $\mathsf{EVK} := (\mathsf{fsk}_1, \ldots, \mathsf{fsk}_\tau, \mathsf{hpk})$ as the evaluation key, and $\mathsf{SK} := \mathsf{hsk}$ as the secret key.

Encryption $\mathsf{TMFHE.Enc}(\mathsf{PK}, M, x)$: takes as input the public key PK of the form $(\{\mathsf{fmpk}_i\}_i, \mathsf{hpk})$, a TM M and a value x of n bits long.

1. Let $\hat{x} \leftarrow (\mathsf{FHE.Enc}(\mathsf{hpk}, x_1), \ldots, \mathsf{FHE.Enc}(\mathsf{hpk}, x_n))$, where x_i is the i-th bit of x. Similarly, let $\hat{M} \leftarrow (\mathsf{FHE.Enc}(\mathsf{hpk}, M_1), \ldots, \mathsf{FHE.Enc}(\mathsf{hpk}, M_n))$, which is the homomorphic encryption of M (the string description of TM M) bit by bit.

2. Compute $c_i \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}_i, (M, x))$ for $i \in [\tau]$.

3. Output the ciphertext $c = (\text{“enc”}, \hat{x}, \hat{M}, c_1, \ldots, c_\tau)$.

Evaluation $\mathsf{TMFHE.Eval}(\mathsf{EVK}, c)$: takes as input an evaluation key EVK of the form $(\{\mathsf{fsk}_i\}_i, \mathsf{hpk})$ and a ciphertext c of the form $(\text{“enc”}, \hat{x}, \hat{M}, c_1, \ldots, c_\tau)$.

1. Start with $i = 1$. Repeat the following:
   (a) $b \leftarrow \mathsf{FE.Dec}(\mathsf{fsk}_i, c_i)$.
   (b) If $b = 1$, (computation finished and we can now evaluate homomorphically on $\hat{x}$)
      i. Compute D_i, the circuit that evaluates a Turing machine in $\mathcal{M}$ for 2^i steps, using $\mathcal{T}_n(2^i)$.
      ii. Evaluate and output $(\text{“eval”}, \mathsf{FHE.Eval}(\mathsf{hpk}, D_i, (\hat{M}, \hat{x})))$.
   (c) Else ($b = 0$), proceed to the next i.

Decryption $\mathsf{TMFHE.Dec}(\mathsf{SK}, c)$: takes as input a secret key $\mathsf{SK} = \mathsf{hsk}$ and a ciphertext c of the form $(\text{“enc”}, \hat{x}, \hat{M}, c_1, \ldots, c_\tau)$ or $(\text{“eval”}, c)$.

1. If the ciphertext is of type “enc”, compute and output $\mathsf{FHE.Dec}(\mathsf{hsk}, \hat{x})$.

2. Else (the ciphertext is of type “eval”), compute and output $\mathsf{FHE.Dec}(\mathsf{hsk}, c)$.

6.2 Results

We now state our results.

Theorem 6.2. *For any class of d-depth-bounded Turing machines that take n bits of input and produce one bit of output, there is a Turing machine homomorphic encryption scheme, assuming the existence of a fully secure functional encryption scheme* FE *for any class of circuits of depth d, and an d-leveled homomorphic encryption scheme* FHE*, where:*

- *The online work of the client is*

$$(n + \log t_{\mathsf{max}}) \cdot \mathrm{poly}(\kappa, d(n))$$

- *The online work of the server in evaluating M on an encryption of x is*

$$\mathrm{poly}(n, d(n), \mathsf{time}(M, x)),$$

where $\mathsf{time}(M, x)$ *is the runtime of M on x.*

This theorem shows that our TMFHE scheme comes as a reduction from any functional encryption scheme. The proof of this theorem is in Appendix C. We can see that the work of the client is indeed smaller than computing the circuit especially if the polynomial d is smaller than the running time. Moreover, we can also see that the server runs in input-specific time: the evaluation time depends on the actual running time and the depth of the circuit.

When instantiating our TMFHE construction with our functional encryption FE construction from Sec. 3 and using Corollary 3.2, we obtain a scheme under an LWE assumption.

Corollary 6.3 (LWE Instantiation)**.** *For every integer $n \in \mathbb{N}$ and polynomial function $d = d(n)$, there is a Turing machine homomorphic encryption scheme for any class of d-depth-bounded Turing machines, under the following assumption: 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 time $2^{O(\ell^\epsilon)}$ in the worst case.*

Remark 6.4. *If the underlying* FE *scheme is selectively secure (Def. 2.14), one can still obtain an input-specific homomorphic encryption scheme, but with selective security; namely, the scheme would achieve a modified version of Def. C.2 in Appendix C (the adversary A must choose x before seeing* EVK *and* PK*). The scheme would then be secure under the following assumption: 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 by polynomial-time adversaries.*

Let us discuss what kind of Turing machines classes are d-depth-bounded.

Fact 6.5. *The class of Turing machines running in* $\log$*-space is* $\log^2$*-depth-bounded.*

This fact follows directly from the known relation that the LOGSPACE complexity class is in NC2.

In general, the following pattern of computation would fit in d-depth-boundedness and would benefit from input-specific evaluation. Consider a computation that on different types of inputs, it performs different kinds of computation; all these computations are of the same (shallow) depth, but the computation can be much larger in one case.

A few remarks are in order:

Remark 6.6. *Denote by universal TMFHE scheme to be a scheme for any finite class of Turing machines. Based on Remark 6.1, we can see that if there is a universal succinct functional encryption scheme and a fully homomorphic scheme, there is a universal TMFHE scheme with online client and server work independent of depth:*

- *The online work of the client becomes*

$$(n + \log t_{\mathsf{max}}) \cdot \mathrm{poly}(\kappa)$$

- *The online work of the server in evaluating M on an encryption of x becomes*

$$\mathrm{poly}(n, \mathsf{time}(M, x)),$$

where $\mathsf{time}(M, x)$ *is the runtime of M on x.*

6.3 Input-Dependent Output Size

The construction above considered Turing machines that output only one bit. To allow TMs that output more than one bit, one can simply use the standard procedure of running one instance of the protocol for each bit of the output. However, as with running time, this would result in repeating the protocol as many times as the worst-case output size for every input. Certain inputs can result in small outputs while others can result in large outputs, so it is desirable to evaluate in *input-specific output size*.

We can use the same approach as above to obtain input-specific output size: The client provides keys to the evaluator to decrypt the size of the output. Then, the evaluator can simply use homomorphic evaluation on a circuit whose output size is the determined one.

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

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

[AGVW12] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. *Cryptology ePrint Archive, Report 2012/468*, 2012.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS*, pages 298–307, 2003.

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO*, pages 278–291, 1993.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Garbling schemes. *Cryptology ePrint Archive, Report 2012/265*, 2012.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, pages 868–886, 2012.

[BSW] Dan Boneh, Amit Sahai, and Brent Waters.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 28th IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[CKVW10] Ran Canetti, Yael Tauman Kalai, Mayank Varia, and Daniel Wichs. On symmetric encryption and point obfuscation. In *TCC*, pages 52–71, 2010.

[Dav12] Michael A. Davis. Cloud security: Verify, don't trust. *Information Week*, August 2012. http://reports.informationweek.com/abstract/5/8978/.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH12] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. Cryptology ePrint Archive, Report 2012/610, 2012.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099, 2012.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *EUROCRYPT*, pages 506–522, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP*, pages 579–591, 2008.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, pages 194–213, 2007.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, August 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22:161–188, April 2009.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, 2012.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358, 2010.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[Pri12] Privacy Rights Clearinghouse. Chronology of data breaches, 2012. `http://www.privacyrights.org/data-breach`.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[SS10a] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.

[SS10b] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, pages 377–394, 2010.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

[SV11] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW12] Amit Sahai and Brent Waters. Attribute-based encryption for circuits from multilinear maps. Cryptology ePrint Archive, Report 2012/592, 2012.

[Vai11] Vinod Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In *FOCS*, pages 5–16, 2011.

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

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

[Wat12] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[Yao86] Andrew C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A Detailed Background on Learning With Errors (LWE)

The LWE problem was introduced by Regev [Reg05] as a generalization of "learning parity with noise" [BFKL93, BKW03, Ale03]. Regev showed that solving the LWE problem *on the average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*. This result bolstered our confidence in the LWE assumption and generated a large body of work building cryptographic schemes under the assumption, culminating in the construction of a fully homomorphic encryption scheme [BV11a].

For positive integers ℓ and $q \geq 2$, a vector $\mathbf{s} \in \mathbb{Z}_q^\ell$, and a probability distribution χ on $\mathbb{Z}_q$, let $A_{\mathbf{s},\chi}$ be the distribution obtained by choosing a vector $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^\ell$ uniformly at random and a noise term $e \xleftarrow{\$} \chi$, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^\ell \times \mathbb{Z}_q$. A formal definition follows.

Definition A.1 (LWE). *For an integer $q = q(\ell)$ and an error distribution $\chi = \chi(\ell)$ over $\mathbb{Z}_q$, the learning with errors problem $\mathsf{LWE}_{\ell,m,q,\chi}$ is defined as follows: Given m independent samples from $A_{\mathbf{s},\chi}$ (for some $\mathbf{s} \in \mathbb{Z}_q^\ell$), output $\mathbf{s}$ with noticeable probability.*

The (average-case) decision variant of the LWE *problem, denoted $\mathsf{dLWE}_{\ell,m,q,\chi}$, is to distinguish (with non-negligible advantage) m samples chosen according to $A_{\mathbf{s},\chi}$ (for uniformly random $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^\ell$), from m samples chosen according to the uniform distribution over $\mathbb{Z}_q^\ell \times \mathbb{Z}_q$.*

We denote by $\mathsf{LWE}_{\ell,q,\chi}$ (resp. $\mathsf{dLWE}_{\ell,q,\chi}$) the variant where the adversary gets oracle access to $A_{\mathbf{s},\chi}$, and is not a priori bounded in the number of samples.

For cryptographic applications we are primarily interested in the average case decision problem dLWE, where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^\ell$. We will also be interested in assumptions of the form: no t-time adversary can solve dLWE with non-negligible advantage, which we will call the t-hardness of dLWE.

There are known quantum [Reg05] and classical [Pei09] reductions between $\mathsf{dLWE}_{\ell,m,q,\chi}$ and approximating short vector problems in lattices. Specifically, these reductions take χ to be (discretized versions of) the Gaussian distribution, which is B-bounded for an appropriate B. Since the exact distribution χ does not matter for our results, we state a corollary of the results of [Reg05, Pei09] in terms of the bound on the distribution.

Let $B = B(\ell) \in \mathbb{N}$. A family of distributions $\chi = \{\chi_\ell\}_{\ell \in \mathbb{N}}$ is called B-bounded if the support of χ_ℓ is (a subset of) $[-B(\ell), \ldots, B(\ell)]$. Then:

Lemma A.1 ([Reg05, Pei09]). *Let $q = q(\ell) \in \mathbb{N}$ be a product of co-prime numbers $q = \prod q_i$ such that for all i, $q_i = \mathrm{poly}(\ell)$, and let $B \geq \ell$. Then there exists an efficiently sampleable B-bounded distribution χ such that if there is an efficient algorithm that solves the (average-case) $\mathsf{dLWE}_{\ell,q,\chi}$ problem. Then:*

- *There is a quantum algorithm that solves SIVP with approximation factor $\widetilde{O}(\ell\sqrt{\ell} \cdot q/B)$ and gapSVP with approximation factor $\widetilde{O}(\ell\sqrt{\ell} \cdot q/B)$ on any ℓ-dimensional lattice, and runs in time $\mathrm{poly}(\ell)$.*
- *There is a classical algorithm that solves the ζ-to-γ decisional shortest vector problem $\mathsf{gapSVP}_{\zeta,\gamma}$, where $\gamma = \tilde{O}(\ell\sqrt{\ell} \cdot q/B)$, and $\zeta = \tilde{O}(q\sqrt{\ell})$, on any ℓ-dimensional lattice, and runs in time $\mathrm{poly}(\ell)$.*

We remark that this connection is time-preserving, in the sense that given an LWE algorithm that runs in time t, these reductions produce algorithms to solve lattice problems that run in time $\mathsf{poly}(t)$.

We refer the reader to [Reg05, Pei09] for the formal definition of these lattice problems, as they have no direct connection to this work. We only note here that the best known algorithms for these problems run in time nearly exponential in the dimension ℓ [AKS01, MV10]. More generally, the best algorithms that approximate these problems to within a factor of 2^k run in time $2^{\tilde{O}(\ell/k)}$. Specifically, given the current state of the art on lattice algorithms, the $\mathsf{LWE}_{\ell,q,\chi}$ assumption is quite plausible for a $\mathrm{poly}(\ell)$-bounded distribution χ and q as large as 2^{ℓ^ϵ} (for any constant $0 < \epsilon < 1$).

Given this state of affairs, we will abuse notation slightly and conflate the LWE dimension ℓ with the security parameter κ.

B Construction of Two-Outcome Attribute-Based Encryption

Let us construct a two-outcome attribute-based encryption scheme, denoted ABE_2, from an ABE scheme, ABE.

The idea is to use two ABE instantiations, one encrypting M_0 and the other M_1. To make sure that exactly one of these messages gets revealed when a predicate is evaluated, we provide secret keys for the predicate and the negation of the predicate for the two instantiations.

Setup $\mathsf{ABE}_2.\mathsf{Setup}(1^\kappa)$**:**

1. Run $(\mathsf{fmsk}_0, \mathsf{fmpk}_0) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$ and $(\mathsf{fmsk}_1, \mathsf{fmpk}_1) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$.
2. Let $\mathsf{fmsk} := (\mathsf{fmsk}_0, \mathsf{fmsk}_1)$ and $\mathsf{fmpk} := (\mathsf{fmpk}_0, \mathsf{fmpk}_1)$. Output fmsk and fmpk.

Key generation $\mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{fmsk}, P)$**:** Let $\mathsf{fsk}_0 \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}_0, \bar{P})$ and $\mathsf{fsk}_1 \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}_1, P)$, where $\bar{P}$ is the negation of P, namely $\bar{P}(x) = 1 - P(x)$. Output $\mathsf{fsk}_P = (\mathsf{fsk}_0, \mathsf{fsk}_1)$.

Encryption $\mathsf{ABE}_2.\mathsf{Enc}(\mathsf{fmpk}, x, M_0, M_1)$**:** Let $C_0 \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}_0, x, M_0)$ and $C_1 \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}_1, x, M_1)$. Output $C = (C_0, C_1)$.

Decryption $\mathsf{ABE}_2.\mathsf{Dec}(\mathsf{fsk}_P, C)$**:**

1. Parse $\mathsf{fsk}_P = (\mathsf{fsk}_0, \mathsf{fsk}_1)$ and $C = (C_0, C_1)$.
2. Run $M_0 \leftarrow \mathsf{ABE.Dec}(\mathsf{fsk}_0, C_0)$ and if $M_0 \neq \perp$, output M_0.
3. Run $M_1 \leftarrow \mathsf{ABE.Dec}(\mathsf{fsk}_1, C_1)$ and if $M_1 \neq \perp$, output M_1.

We next prove that this construction yields a secure two-outcome ABE scheme. Note that our construction requires an ABE scheme where the predicate class $\mathcal{P}_n$ is closed under negation: for every predicate $P \in \mathcal{P}_n$, the predicate $\bar{P}$ is also included in $\mathcal{P}_n$.

Proof of Claim 2.5. Correctness of ABE_2 is straightforward: If $P(x) = 0$, C_0 will decrypt to M_0 by the correctness of ABE, and mutatis mutandis for $P(x) = 1$.

We prove security by contradiction. Assume there exists p.p.t. $A = (A_1, A_2, A_3)$ that breaks the security of our ABE_2 construction: Def. 2.11; namely, there exists a polynomial p such that, for infinitely many κ,

$$\Pr[\mathsf{Exp}_{\mathsf{ABE}_2,A}(1^\kappa) = 1] \geq 1/2 + 1/p(\kappa). \tag{6}$$

We construct a p.p.t. adversary $R = (R_1, R_2, R_3)$ that breaks the security of ABE, Def. 2.9.

The adversary R_1 receives as input fmpk^* and outputs a predicate P^* as follows. The adversary A_1 expects two public keys. R_1 uses fmpk^* as one of these public keys and generates the other public key freshly $(\mathsf{fmsk}, \mathsf{fmpk}) \leftarrow \mathsf{ABE.KeyGen}(1^\kappa)$. The order in which R_1 provides these keys to A_1 depends on the value of $P(x)$ not known at this step. If $P(x)$ will be 0, R will have to give A the ability to decrypt a ciphertext encrypted with the first key. If that key is fmpk^*, R cannot accomplish this task because it does not have the corresponding secret key. Therefore, R will try to guess $P(x)$ by flipping a random coin. Concretely, R_1 runs:

1. Guess $P(x)$ at random, namely draw a random bit denoted guess. If guess is 0:
   (a) Provide $(\mathsf{fmpk}, \mathsf{fmpk}^*)$ to A_1.
   (b) Receive P from A_1 and output $P^* := P$.
2. Else [guess is 1]:
   (a) Provide $(\mathsf{fmpk}^*, \mathsf{fmpk})$ to A_1.
   (b) Receive P from A_1 and output $P^* := \bar{P}$.

Adversary R_2 receives as input fsk_{P^*} and generates M_0^*, M_1^*, and x^* as follows.

1. Generate $\mathsf{fsk}_{\bar{P}^*} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}, \bar{P}^*)$.
2. If guess is 0, provide $(\mathsf{fsk}_{\bar{P}^*}, \mathsf{fsk}_{P^*})$ to A_2, else (guess was 1) provide $(\mathsf{fsk}_{P^*}, \mathsf{fsk}_{\bar{P}^*})$ to A_2.
3. Receive (M, M_0, M_1, x) from A_2. Output $M_0^* := M_0$, $M_1^* := M_1$ and $x^* := x$.

Adversary R_3 receives as input c^* and outputs a guess bit as follows:

1. Check that $P(x)$ equals guess. If this is not the case, namely, R_1 had guessed incorrectly the value of $P(x)$, output a random bit and exit. Otherwise, continue.
2. Feed the following input to A_3: if guess $= 0$, feed inputs $(\mathsf{ABE.Enc}(\mathsf{fmpk}, (x, M)), c^*)$, else (guess $=$ 1), feed inputs $(c^*, \mathsf{ABE.Enc}(\mathsf{fmpk}, (x, M)))$. Output whatever A_3 outputs.

R_1 guesses $P(x)$ correctly with a chance of half. When R_1 does not guess $P(x)$ correctly, R_3 outputs a correct bit with chance $1/2$ (because it outputs a random guess). When R_1 guesses $P(x)$ correctly, we can see that R simulates the ABE_2 game with A correctly. Therefore, in this case, whenever A guesses correctly, R also guesses correctly. Using Eq. (6), we have

$$\Pr[\mathsf{Exp}_{\mathsf{ABE},R}(1^\kappa) = 1] \geq 1/2 \cdot 1/2 + 1/2(1/2 + 1/2p(\kappa)) = 1/2 + 1/2p(\kappa), \tag{7}$$

which provides the desired contradiction. □

C Homomorphic Encryption for Turing Machines: Definitions and Proofs

Let us first define the syntax of a Turing machine homomorphic encryption scheme.

Definition C.1. *A Turing machine homomorphic encryption scheme* TMFHE *for a class of Turing machines* $\mathcal{M}$ *is a quadruple of p.p.t. algorithms* $(\mathsf{TMFHE.KeyGen}, \mathsf{TMFHE.Enc}, \mathsf{TMFHE.Dec}, \mathsf{TMFHE.Eval})$ *as follows:*

- $\mathsf{TMFHE.KeyGen}(1^\kappa, 1^n, 1^{t_{\mathsf{max}}})$ *takes as input a security parameter* κ*, an input size* n*, and a time bound* t_{max}*, and outputs a public key* PK*, an evaluation key* EVK*, and a secret key* SK*.*
- $\mathsf{TMFHE.Enc}(\mathsf{PK}, M, x)$ *takes as input the public key* PK*, a Turing machine* M *with one bit of output, and an input* $x \in \{0,1\}^n$*, for some* n*, and outputs a ciphertext* c*.*
- $\mathsf{TMFHE.Dec}(\mathsf{SK}, c)$ *takes as input the secret key* SK *and a ciphertext* c*, and outputs a message* x*.*
- $\mathsf{TMFHE.Eval}(\mathsf{EVK}, c)$ *takes as input the evaluation key* EVK*, and a ciphertext* c*, and outputs a ciphertext* c'*.*

Correctness: *For every polynomial* $n(\cdot)$*, for every polynomial* $t_{\mathsf{max}}(\cdot)$*, for every sufficiently large security parameter* κ*, for* $n = n(\kappa)$*, for every Turing machine* $M \in \mathcal{M}$ *with upper bound on running time for inputs of size* n *of* $t_{\mathsf{max}}(n)$*, and for every input* $x \in \{0,1\}^n$*,*

$$\begin{aligned}
&\Pr[(\mathsf{PK}, \mathsf{EVK}, \mathsf{SK}) \leftarrow \mathsf{TMFHE.KeyGen}(1^\kappa, 1^n, 1^{t_{\mathsf{max}}(n)});\\
&\quad c \leftarrow \mathsf{TMFHE.Enc}(\mathsf{PK}, M, x);\\
&\quad c^* \leftarrow \mathsf{TMFHE.Eval}(\mathsf{EVK}, M, c):\\
&\quad \mathsf{TMFHE.Dec}(\mathsf{SK}, c^*) \neq M(x)] = \operatorname{negl}(\kappa).
\end{aligned}$$

Note that the correctness property constraints t_{max} to be a polynomial. However, t_{max} can still be a very large polynomial and we would like the server to not have to run in that time for all inputs. (In fact, this constraint can be eliminated if we use a FHE scheme and an ABE scheme that have no correctness error).

Definition C.2 (Runtime-CPA Security)**.** *Let* TMFHE *be an input-specific homomorphic encryption scheme for the class of Turing machines* $\mathcal{M}$*. For every p.p.t. adversary* $A = (A_1, A_2)$ *and p.p.t. simulator* S*, consider the following two experiments:*

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{TMFHE},A}(1^\kappa)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{TMFHE},A,S}(1^\kappa)$:
1: $(1^{t_{\mathsf{max}}}, 1^n, \mathsf{state}_A) \leftarrow A_1(1^\kappa)$.	
2: $(\mathsf{PK}, \mathsf{EVK}, \mathsf{SK}) \leftarrow \mathsf{TMFHE.KeyGen}(1^\kappa, 1^n, 1^{t_{\mathsf{max}}})$	
3: $(M, x, \mathsf{state}'_A) \leftarrow A_2(\mathsf{state}_A, \mathsf{PK}, \mathsf{EVK})$	
4: $c \leftarrow \mathsf{TMFHE.Enc}(\mathsf{PK}, M, x)$	*4:* $\tilde{c} \leftarrow S(M, 1^n, 1^t, \mathsf{EVK}, \mathsf{PK})$ with $t = \mathsf{time}(M, x)$
5: Output (state'_A, c)	*5:* Output $(\mathsf{state}'_A, \tilde{c})$

The scheme is said to be runtime-CPA-secure if there exists a p.p.t. simulator S such that for all pairs of p.p.t. adversaries $A = (A_1, A_2)$ for which A_2 outputs $M \in \mathcal{M}$ and $x \in \{0,1\}^n$, we have

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

This definition essentially captures our security goal: one can simulate any information learned from the scheme by using only the Turing machine M and the running time of M on x, but without any other information about x.

In fact, we can achieve a scheme that hides M as well in a straightforward way: since our construction passes M and x as inputs to universal circuits, M could also be hidden in the same way as x is.

C.1 Proof

Proof of Theorem 6.2. We first prove the correctness and efficiency claims of the theorem and then we prove security.

If the underlying FE scheme is correct, then TMFHE is correct; whenever 2^i for some i is an upper bound on the running time of M on x, then $C_i(M, x)$'s output is 1. Based on the correctness of the FHE scheme FHE, the evaluation of D_i on $\hat{M}, \hat{x}$ will be correct, so $\mathsf{FHE.Dec}$ will return $M(x)$.

Lemma C.1. *The online work of the client in the* TMFHE *scheme is* $(\log t_{\mathsf{max}} + n) \cdot \mathrm{poly}(\kappa, d(n))$.

Proof. The work of the client in the online phase consists of running $\mathsf{TMFHE.Enc}(\mathsf{PK}, x)$ and $\mathsf{TMFHE.Dec}(\mathsf{SK}, c)$. The work of the client for $\mathsf{TMFHE.Enc}(\mathsf{PK}, x)$ is $n\mathrm{poly}(d(\kappa))$ to compute the FHE ciphertexts and $(1 + \lceil \log t_{\mathsf{max}} \rceil) \cdot \mathrm{poly}(d(n), \kappa)$ to compute the FE ciphertexts. Since n depends polynomially in κ, we obtain that total cost is at most $(\log t_{\mathsf{max}} + n)\mathrm{poly}(\kappa, d(n))$ (where be incorporated the constant values in the poly notation).

The runtime of $\mathsf{TMFHE.Dec}(\mathsf{SK}, c)$ is $\mathrm{poly}(d(\kappa))$ because $\mathsf{FHE.Enc}$ runs polynomial in κ and $d(\kappa)$. Therefore, the total online work of the client is $(\log t_{\mathsf{max}} + n)\mathrm{poly}(d(\kappa), d(n), \kappa)$. □

Lemma C.2. *The work of the evaluator in the* TMFHE *scheme is* $\mathrm{poly}(n, d(n), \mathsf{time}(M, x))$.

Proof. The work of the evaluator consists of running $\mathsf{TMFHE.Eval}(\mathsf{EVK}, M, c)$. This depends on the number of times the loop in $\mathsf{TMFHE.Eval}$ is repeated and the cost within each loop. Let us evaluate the cost at the i-th repetition of the loop. Let $t_i = 2^i$.

By the properties of the transformation $\mathcal{T}_n$, the size of C_i is at most $t_i \mathrm{polylog}\ t_i$. The cost of evaluating $\mathsf{FE.Dec}(\mathsf{fsk}_i, c_i)$ is therefore $\mathrm{poly}(n, d(n), t_i \mathrm{polylog}\ t_i)$.

If t is the runtime of M on x, the index i at which the loop will halt (because the evaluator obtained a value the bit b being one) is at most $1 + \lceil \log t \rceil$. Therefore, the loop will repeat at most $1 + \lceil \log t \rceil$ times.

$$\begin{aligned} \text{Runtime of } \mathsf{TMFHE.Eval}(\mathsf{EVK}, c) &= \sum_{i=1}^{1+\lceil \log t \rceil} \mathrm{poly}(n, d(n), t_i \;\mathrm{polylog}\; t_i) \\ &\leq (1 + \lceil \log t \rceil)\mathrm{poly}(n, d(n), t \;\mathrm{polylog}\; t) \\ &\leq \mathrm{poly}(n, d(n), t \;\mathrm{polylog}\; t) = \mathrm{poly}(n, d(n), t), \end{aligned}$$

where the last equality comes from adjusting the implicit polynomial in poly. Note that even though EVK consists of $\log t_{\mathsf{max}}$ such fsk_i keys, $\mathsf{TMFHE.Eval}$ does not have to read all of EVK. □

Finally, we prove security of the scheme.

Lemma C.3. *The* TMFHE *protocol is runtime-CPA-secure.*

Proof. To prove that our TMFHE construction is secure, we provide a simulator S, as in Def. C.2. The simulator S invokes the simulator of the functional encryption scheme, as in Def. 2.13, which we denote $\mathsf{Sim}_{\mathsf{FE}}$. The simulator S receives inputs M, 1^n, 1^t, EVK, and PK, and proceeds as follows:

1. Compute $\hat{0^n} \leftarrow (\mathsf{FHE.Enc}(\mathsf{hpk}, 0), \ldots, \mathsf{FHE.Enc}(\mathsf{hpk}, 0))$ (n times).
2. For each $i \in [\tau]$, compute the circuits $D_i = T_n(2^i)$ and then C_i as before; we remind the reader that C_i, on input a TM M and a value x, outputs 1 if M finished in 2^i steps when running on input x or 0 otherwise.
3. For each i such that $2^i < t$:
   (a) Call the simulator $\mathsf{Sim}_{\mathsf{FE}}$ to simulate a computation result of 0 because M could not have finished its computation at step i. Specifically, compute $\tilde{c}_i \leftarrow \mathsf{Sim}_{\mathsf{FE}}(\mathsf{fmpk}_i, \mathsf{fsk}_i, C_i, 0, 1^{n+|M|})$.
4. For each i such that $2^i \geq t$:
   (a) Call the simulator $\mathsf{Sim}_{\mathsf{FE}}$ to simulate an answer of 1 because M finished computation on the input (unknown to S). Thus, compute $\tilde{c}_i \leftarrow \mathsf{Sim}_{\mathsf{FE}}(\mathsf{fmpk}_i, \mathsf{fsk}_i, C_i, 1, 1^{n+|M|})$.
5. Output $\tilde{c} = (\hat{0}, \tilde{c}_1, \ldots, \tilde{c}_\tau)$.

To prove that S satisfies Def. C.2, we use three hybrids:
Hybrid 0: The ideal experiment with simulator S.
Hybrid 1: The same as Hybrid 0 but $\hat{0^n}$ gets replaced with $\hat{x} = (\mathsf{FHE.Enc}(\mathsf{hpk}, x_1), \ldots, \mathsf{FHE.Enc}(\mathsf{hpk}, x_n))$.
Hybrid 2: The real experiment.
It is easy to see that the outcome of Hybrid 0 and the outcome of Hybrid 1 are computationally indistinguishable because FHE is semantically secure: the encryptions of 0^n in Hybrid 0 and the encryption of x in Hybrid 1 are both generated with fresh randomness, and the secret key hsk (or any function of hsk other than a fresh encryption) is never released to any adversary.

Now let us look at Hybrid 1 and Hybrid 2. These are computationally indistinguishable based on a standard hybrid argument invoking the security of $\mathsf{Sim}_{\mathsf{FE}}$ as follows.

The simulator $\mathsf{Sim}_{\mathsf{FE}}$ is called τ times. Let $\tilde{c}_i^{(1)}$ be the ciphertext output by the simulator for the i-th invocation in Hybrid 1, and let c_i be the ciphertext output in Hybrid 2 on the i-th invocation. It is enough to prove that the outcome of these two experiments consisting of state'_A and only one of the ciphertexts (e.g., $\tilde{c}_i^{(1)}$ or c_i) are computationally indistinguishable. The reason is that one can employ a standard hybrid argument consisting of $\tau + 1$ sub-hybrids, the 0-th sub-hybrid being Hybrid 1 and the τ-th sub-hybrid being Hybrid 2 and any intermediary sub-hybrid i has the first i ciphertexts as in Hybrid 2 and the rest as in Hybrid 1. Such an argument is possible because τ is polynomial in the security parameter and each ciphertext is encrypted with independently generated public keys.

Therefore, all we need to argue is that the outcome of Hybrid 1 and Hybrid 2 consisting of state'_A and only $\tilde{c}_i^{(1)}$ (c_i respectively) are computationally indistinguishable. This follows directly because $\mathsf{Sim}_{\mathsf{FE}}$ satisfies the $\mathsf{FULL\text{-}SIM}$-secure functional encryption definition, Def. 2.13. $\square$

The three lemmas above complete the proof of the theorem. $\square$