Overcoming the Worst-Case Curse for Cryptographic Constructions

Shafi Goldwasser[⋆] Yael Kalai[†] Raluca Ada Popa[⋆]
Vinod Vaikuntanathan[⋈] Nickolai Zeldovich[⋆]

⋆ MIT CSAIL † Microsoft Research ⋈ University of Toronto

Abstract

Modeling efficient algorithms as polynomial size circuits rather than as polynomial time Turing machines has been the rule with few exceptions in cryptographic constructions which provide "secure versions" of general efficient algorithms. A consequence of this modeling is that the resulting "secure version" of an efficient algorithm A incurs the worst-case runtime of A over all inputs of a certain length, rather than the runtime of A on specific inputs.

In this work, we address the challenge of achieving input-specific runtime rather than worst-case runtime for a wide variety of cryptographic tasks. In particular, we construct (under cryptographic assumptions detailed below):

- An attribute-based encryption (ABE) scheme for any polynomial-time Turing and RAMs (including those with non-uniform advice), where the length of the function keys (or Turing machine keys) depends on the *size* of the Turing machine (and does not depend on its runtime). Moreover, the decryption algorithm has input-specific runtime (as opposed to worst-case).
- A single-key functional encryption scheme (FE) for any polynomial-time Turing machines (uniform or non-uniform), where the length of the function keys (or Turing machine keys) depends only on the *size* of the Turing machine independent of its runtime. In addition, we construct a decryption algorithm that has input-specific runtime (at the price of revealing this runtime).
- A reusable garbling scheme for arbitrary Turing machines (uniform or non-uniform), where the size of the garbling depends only on the size of the Turing machine.

Previously, it was known how to construct all these objects for depth d circuits, where all the parameters grow with d. Our constructions remove this depth d restriction, and moreover, avoid the worst-case "curse".

We also show a fully homomorphic encryption scheme for Turing machines (including those with non-uniform advice), where given a ciphertext $\mathsf{Enc}(x)$ and any Turing machine M, one can compute $\mathsf{Enc}(M(x))$ in time that is dependent on the runtime of M *on input* x as opposed to the worst-case runtime. Previously, such a result was known only for a restricted class of Turing machines and it required an expensive preprocessing phase (with worst-case runtime). Our result is for any class of polynomial time Turing machines and removes the expensive preprocessing.

Our results are obtained via a reduction from (a variant of) the witness encryption scheme, recently introduced by Garg *et al.* (STOC 2013) and the existence of SNARKs (Bitansky *et al.* STOC 2013). In particular, when instantiating our schemes using the witness encryption construction proposed by Garg *et al.*, the security of our schemes relies on a strengthening of their assumption. We thus view our results as a "proof of concept". We note that previously, no proposals or even heuristics for such schemes existed. We also point out the connection between this variant of witness encryption and the obfuscation of point filter functions as defined by Goldwasser and Kalai in 2005.

Keywords: Functional encryption; Fully homomorphic encryption; Turing machines; Input-specific running time.

1 Introduction

Modeling general efficient algorithms as polynomial size circuits is prominent in cryptographic constructions.[1] A consequence of this modeling is that all these *secure* variants of efficient algorithms incur worst-case performance regardless of the running time of the computation on the actual inputs, namely, the *input-specific performance* of the underlying algorithm. The reason is that the known transformations from algorithms modeled as Turing machines to algorithms modeled as circuits essentially work by unrolling computations to their worst case runtime; the outcome is that the runtime on any input of length n is as long as the time required by the worst case input of length n – implying a major slow down of secure variants of algorithms (even without taking into account the cost of additional cryptographic building blocks). To exemplify, consider algorithms (or even heuristics) which may run in polynomial time on all instances one encounters in practice, but run in exponential time on rare inputs, such as the simplex algorithm for linear programming.

The earliest example of this phenomenon is Yao's secure function evaluation protocol [Yao86] which takes as input any polynomial time computable function f – specified by a Boolean circuit – and outputs a "garbled circuit" with the same input-output functionality. In this example, it is thus entirely possible that the performance of the best algorithm (or heuristic) for computing f on most inputs x would take linear time, and yet the size of the garbled circuit for f on inputs of length n is n^{100}. This worst case run-time "curse" affects known two-party and multi-party protocols for general secure function evaluation [Yao86, GMW87, BGW88, CCD88], constructions of fully homomorphic encryption schemes [Gen09, BV11a, BV11b, BGV12, Bra12], constructions of attribute-based encryption [GVW13, SW12, GGH12], and constructions of functional encryption schemes for general functions [SS10, AGVW12, GKP+13].

In this work we address this challenge. Stated broadly and informally, we consider a variety of cryptographic tasks all of which (in a sense) "compile" a given generic Turing machine algorithm M into an "encrypted" algorithm which works on "encrypted" data using circuits. For each such task, we show how to ensure that the latter algorithms achieve the runtime of M on input x rather than the worst-case runtime of M on all inputs of length n where $n = |x|$. As a necessary by product of our results, the new encrypted algorithms will leak the runtime of M on x. The cryptographic tasks we address include the construction of an FHE scheme, a (non-adaptive) ABE scheme, (single-key or bounded-collusion) functional encryption scheme, and (reusable) computation garbling schemes, where we model functions as *Turing machines*.

Interestingly, the worst-case curse can be easily overcome for interactive tasks such as two-party and multi-party protocols: in order to securely compute a Turing machine M, first compute Turing machines $M_1, \ldots, M_{\omega(\log n)}$ sequentially, one after the other, where M_i runs the Turing machine M for 2^i steps and outputs the answer if M halts within 2^i steps, and $\perp$ otherwise. To compute M_i, we use any multi-party computation protocol and note that the circuit size for M_i is $\mathrm{poly}(2^i)$. Since we halt the computation as soon as we get an answer (which is not $\perp$), this protocol runs in input-specific time. More generally, the reason we are able to overcome the worst-case curse in the case of two-party and multi-party computation setting is precisely the interactive nature of these tasks. In this work, we focus on non-interactive tasks such as attribute-based encryption, fully homomorphic encryption, functional encryption and reusable garbled schemes.

Throughout this introduction, we assume that Turing machines may have a non-uniform tape, and thus such Turing machines can in particular compute any circuit.

Our schemes (listed explicitly below) are all based on a strong (and new) computational assumption which is a strengthening of the assumption made in a recent paper of Garg *et al.* [GGSW13]. Given that this assumption is a newcomer to the cryptographic scheme, we view our contribution mainly as a "proof of concept". We are not aware of even heuristic constructions for such schemes.

[1] See Sec. 1.4 for a discussion of some exceptions.

1.1 Our results

Let us now explain our specific results in detail.

Attribute-based encryption (ABE) for Turing and RAM machines. Attribute-based encryption schemes, originally defined by Sahai and Waters [SW05], allow a user holding the secret key sk to generate function-keys sk_f for any function f of his choice. The user can generate many such function keys. Given the corresponding public key pk, and given an 'attribute" x, any user can encrypt a message m with respect to the (public) attribute x. Such ciphertext is denoted by $\mathsf{Enc}(x; m)$. The ciphertext $\mathsf{Enc}(x; m)$ does not hide x, and hides only m. Given a function key sk_f and a ciphertext $\mathsf{Enc}(x; m)$ one can compute m only if $f(x) = 1$. On the other hand if $f(x) = 0$ then no information about m is leaked, and semantic security is achieved.

Attribute-based encryption is a powerful primitive with many applications, and has deserved significant attention [GPSW06, LOS+10, LW12], culminating with the work of Gorbunov, Vaikuntanathan and Wee [GVW13], who constructed under the LWE assumption an attribute-based encryption scheme for the class of all circuits of depth at most d, where d is an arbitrary parameter. However, the efficiency of the scheme (such as the size of the ciphertexts) deteriorates with d. In concurrent work, Sahai and Waters [SW12] and Garg, Gentry, Sahai, and Waters [GGSW13] have constructed attribute-based encryption schemes with similar properties from multi-linear maps. The recent work of Garg *et al.* [GGSW13] constructs an ABE scheme for all poly-size circuit, however their scheme is not succinct (the ciphertexts grow with the worst-case runtime of the circuits).

In this work, we construct an attribute-based encryption scheme for all circuits, with no restriction on the depth. Moreover, and arguably more importantly, in our ABE scheme we model the functions as (possibly non-uniform) Turing machines, as opposed to circuits as in all previous work. Computing a function-key sk_M corresponding to a Turing machine M, takes time that is polynomial to the *size* of M, *independent of the runtime* of M. Moreover, given sk_M and $\mathsf{Enc}(x; m)$ where $f(x) = 1$, one can compute m in time that depends only on the time it takes to compute M *on input* x (as opposed to the worst-case running time of M). We prove the security of our scheme with respect to a non-adaptive simulation-based definition (we refer the reader to Section 3 for details). Our ABE construction also carries through for RAMs.

Theorem 1.1 (Informal)**.** *There exists a secure attribute-based encryption scheme (as per Def. C.2) for the class of all (uniform or non-uniform) polynomial time Turing and RAMs from the assumptions in Sec. 1.2.*

Functional encryption (FE) for Turing machines. Functional encryption, formalized by Boneh, Sahai and Waters [BSW11], is a generalization of attribute-based encryption. In a functional encryption scheme a user holding the secret key sk can generate a function-key sk_f corresponding to any function f, and then any user who holds a ciphertext $\mathsf{Enc}(m)$ and a key sk_f can compute $f(m)$, but learns nothing else about the message m.

Previously, the only functional encryption schemes known (where the user is allowed to release many keys), was for the inner product functionality [KSW08, LOS+10]. Agrawal *e*t al. [AGVW12] showed that there does not exist an arbitrary number of keys functional encryption scheme achieving a simulation-based definition for general functions.[2] This led to the quest of single-key functional encryption (or bounded-collusion functional encryption). Indeed, Sahai and Seyalioglu [SS10] constructed a *single-key* functional encryption for arbitrary circuits of size at most s, followed by Gorbunov, Vaikuntanathan and Wee [GVW12] for a bounded number of keys. However, the size of the ciphertexts in both schemes grow with the circuit size s. Thus these schemes are not *succinct*.

Very recently, Goldwasser *et al.* [GKP+13], constructed a *succinct* single-key functional encryption scheme for any depth d circuit, where the parameters of the scheme grow with d (but are independent of the

[2] We note however, that the negative result of Agrawal *et al.* is only for a simulation-based definition, and we do not know of a negative result for indistinguishability based definitions.

circuit size).

In this work we remove this depth d restriction. Moreover, we model the functions as (possibly non-uniform) Turing machines, as opposed to circuits as in all prior work. In our scheme computing the function-key of a Turing machine M, depends only on the size of M and does not depend on the run-time of M. We note that in all previous schemes the size of a function-key for a function f grows (at least linearly) with the worst-case runtime of f. We note however, that as opposed to our ABE scheme, in a functional encryption scheme, given $\mathsf{Enc}(x)$ and sk_M, the time it takes to compute $M(x)$ must be proportional to the worst-case runtime of M, since the runtime of M on input x may leak sensitive information about x. However, if one is willing to slightly relax security and allow to leak the runtime of M on the secret input x, then we also show how to construct a decryption algorithm that has input-specific runtime (i.e., runs in time polynomial in the runtime of M on input x) – we denote this *input-specific runtime functional encryption*.

Theorem 1.2 (Informal)**.** *There exists a single-key functional encryption scheme and a single-key input-specific runtime functional encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines from the assumptions in Sec. 1.2.*

FHE for Turing machines. We construct an FHE scheme where one can evaluate a program P on a ciphertext $\mathsf{Enc}(x)$ in time that depends on the runtime of P *on the specific input* x. At first glance, this may seem impossible, since revealing the runtime of P on input x may reveal secret information about x. However, for many Turing machines M, revealing the runtime of M on various inputs is not harmful, and is many times essential to make the FHE usable in practice.

Our construction is an improvement of the one of Goldwasser *et* al. [GKP+13] who showed how to construct input-specific runtime FHE from single-key functional encryption. In our scheme, one encrypts a Turing machine M and x together into a token $\mathsf{tk}_{M,x}$. Producing such a token only depends on the size of x and M, and not on the running time of M. The evaluator can use $\mathsf{tk}_{M,x}$ and public information to compute $M(x)$ in input-specific time. There are two reasons why M and x are combined in a token $\mathsf{tk}_{M,x}$: (1) For security, the FHE evaluator must no longer be able to evaluate TMs of its choice on the encrypted inputs because the running time of those TMs can leak the input. Therefore, the evaluator must receive a token permitting it to run a certain TM M; (2) Since the FE scheme is only secure for a single-key, we encrypt both M and x in the FE ciphertext, and generate a FE key for the universal Turing machine U.

Comparing to the scheme of Goldwasser et al. [GKP+13], we make the following improvements:

- *Remove costly preprocessing.* [GKP+13] had an expensive preprocessing phase taking as long as the worst-case runtime. With our scheme, the preprocessing is cheap: polynomial in the size of the TMs and independent of the worst-case runtime (so in fact it can be performed in the online phase).
- *Works for any class of polynomial time Turing machines.* Because ciphertexts depended on the depth of the worst case circuit representation of the class of Turing machines in [GKP+13], they only allowed a restricted class of Turing machines: the class of TMs that can be expressed by shallow depth circuits (e.g., log-space Turing machines). Our result does not have the depth dependency and thus applies to any class of polynomial time Turing machines.

Theorem 1.3 (Informal)**.** *There exists an input-specific runtime fully homomorphic encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines from the assumptions in Sec. 1.2.*

Reusable garbling scheme for Turing machines. Garbling schemes, introduced in the seminal work of Yao [Yao86], have been proven to be very useful in cryptography. In such schemes, a user can "garble" a function f, such that the garbling itself does not yield any information about f. Given any input x the garbler (who has secret key) can release a token tk_x. Given a garbling of f and a token tk_x one can

compute $f(x)$, but learns nothing else about f or x. Various works have also considered an authenticity property [BHR12, GVW13] on which we do not delve.

In known garbling schemes (even non reusable garbling), the garbled circuits evaluation and program garbling are as large as the worst-case runtime of f. Often, the reason is that the programs are modeled as circuits, and the size of the garbling is at least the size of the corresponding circuit. In this work we construct a (reusable) garbling scheme for (uniform or non-uniform) Turing machines, where the size of the garbling depends only on the size of the Turing machine, and is *independent of its runtime*.

As in our FHE and FE schemes, if one allows to leak the runtime of M on input x, we can additionally avoid worst-case evaluation time: then a garbling for a Turing machine M, and given a token tk_x corresponding to an input x, the time it takes to compute $M(x)$ is polynomial in the runtime of M on the specific input x. Goldwasser *et* al. [GKP+13] only provide such a scheme for depth bounded circuits, and we thus remove the depth dependency, and moreover, can additionally avoid worst-case running time.

Theorem 1.4 (Informal)**.** *There exists a reusable garbling scheme for the class of all (uniform or non-uniform) polynomial time Turing machines from the assumptions in Sec. 1.2.*

1.2 Our Assumptions

Our schemes rely on two assumptions: extractable witness encryption and the existence of SNARKs.

Extractable Witness Encryption. To describe our first assumption, we first refer to a recent work of Garg *et al.* [GGSW13]. This work constructs a new beautiful primitive called a witness encryption scheme. A witness encryption scheme is associated with some **NP** complete language L. Any user given an instance x and a message m, can encrypt m with respect to x; this is denoted by $\mathsf{Enc}_x(m)$. Any user given $\mathsf{Enc}_x(m)$ and a valid witness w of x can decrypt x efficiently. On the other hand, if x is not in the language then semantic security is ensured. The construction of Garg *et al.* relies on a new computational assumption called *Decision Graded Encoding No-Exact-Cover* or shortly *DGE No-Exact-Cover* assumption.

In our work, we additionally assume that the [GGSW13] scheme is extractable – and such an extractable scheme can be constructed from an extractable version of their assumption; thus, we strengthen their assumption. Loosely speaking, we require that any adversary that can decrypt $\mathsf{Enc}_x(m)$ with non-negligible advantage, must "know" a valid witness w corresponding to x. Specifically, we require that there exists a PPT extractor E such that for every PPT (possibly non-uniform) adversary $\mathcal{A}$, if $\mathcal{A}$ on input $\mathsf{Enc}_x(m)$ outputs m with non-negligible advantage, then the extractor $E(x, \mathcal{A})$ extracts a valid witness w from $\mathcal{A}$ with non-negligible probability. Thus, our witness encryption scheme is with respect to an **NP** relation (as opposed to an **NP** language). We call this assumption the *extractable* DGE No-Exact-Cover assumption.

We refer the reader to Section 2 for more details on the assumption, and emphasize that we view our result as a reduction from any extractable witness encryption scheme, as opposed to a result that is tied to the specific computational assumption.

Interestingly, we remark that extractable witness encryption is highly related to another task that was already well-known in the cryptographic literature, namely (weakly) obfuscating point-filter functions, first defined in 2005 by Goldwasser and Kalai [GK05]. Informally, point-filter functions for a language $L \in \mathbf{NP}$ with witness relation R_L are a class of functions $\{\delta_{x,b}\}$, indexed by a string $x \in \{0,1\}^n$ and a bit $b \in \{0,1\}$ that behave as follows:

$$\delta_{x,b}(w) = \begin{cases} (x, b) & \text{if } (x, w) \in R_L \\ (x, \perp) & \text{otherwise} \end{cases}$$

In other words, the function reveals the "secret bit" b only when given a witness w such that $(x, w) \in R_L$. It can be shown that (weak)[3] extractable witness encryption is indeed equivalent to (weakly) obfuscating point filter function. Thus, the former implies the consequences of the later regarding the impossibility of

[3] A relaxation of extractable witness encryption

obfuscation for a wide range of natural tasks [GK05]. See Sec. B for more details on this equivalence and its consequences.

The existence of SNARKs, Succinct Non-interactive Arguments of Knowledge (SNARK) as defined in [GLR11, BCCT12, DFH12]. Bitansky et al. [BCCT13] construct SNARKs in a generic way (via a reduction from weaker SNARKs). Their work is based on non-falsifiable assumptions, such as "knowledge of exponent assumptions" as in [Gro10] or [BCI+13], and the existence of collision resistant hash functions.

Our functional encryption schemes, FHE for Turing machines scheme, and reusable garbling scheme, additionally rely on the existence of a fully homomorphic encryption scheme, which can be obtained from the LWE assumption with circular security [BGV12].

1.3 Our Techniques

ABE for Turing machines. The main technical challenge in this work is constructing an ABE scheme for Turing machines. The other primitives use the ABE scheme as a building block, similarly to the recent work of Goldwasser *et al.* [GKP+13], who showed how to construct a functional encryption scheme and a reusable garbling schemes from an ABE scheme.

Our ABE construction relies on a variant of the recent work of Garg *et al.* [GGSW13], who defined and constructed a witness encryption scheme (described above). The basic idea is the following: The secret key corresponding to a Turing machine M is simply a signature of M (using any signature scheme that is existentially unforgeable against adaptive chosen message attacks). The master secret and public keys generated during the setup algorithm are the secret and verification keys $(\mathsf{SigSK}, \mathsf{VK})$ for the signature scheme. To encrypt a bit b with respect to a (public) attribute x, we compute a witness encryption $\mathsf{Enc}_{x^*}(b)$, where $x^* = (x, \mathsf{VK})$ and where a valid witness for x^* is a tuple (M, σ, π), where M is a Turing machine, σ is a signature of M using SigSK, and π the tableau of the computation, which is a "proof" that $M(x) = 1$.

Loosely speaking, the security proof proceeds as follows: Suppose there exists an adversary $\mathcal{A}$ that given $\mathsf{Enc}_{x^*}(b)$, which is an ABE encryption of a random bit b w.r.t. public attribute x, and given several secret keys $\mathsf{sk}_{M_i} = \sigma_i$ such that $M_i(x) = 0$ (where σ_i is a signature of M_i), succeeds in guessing b with non-negligible advantage. The security of the extractable witness encryption implies that there exists a poly-time extractor that extracts a valid witness from $\mathcal{A}$ with non-negligible probability. Recall that a valid witness is a triplet the form (M^*, σ^*, π^*) where σ^* is a valid signature of the Turing machine M^* and π^* is a proof that $M^*(x) = 1$. Note that since $M_i(x) = 0$ for every i, it must be the case that $M^* \neq M$, which contradicts the security of the underlying signature scheme.

Unfortunately, this idea does not quite give us the results we want. The reason is that a witness for an instance $x^* = (x, \mathsf{VK})$ is very long since it consists of a triplet (M^*, σ^*, π^*), where π^* is the entire tableau of M^* on input x. The witness encryption of Garg *et al.* [GGSW13], as well as our stronger version, is not "succinct", in the sense that the size of the ciphertext $\mathsf{Enc}_{x^*}(b)$ grows with the witness size. Thus, the approach above gives us a non-succinct ABE scheme, where the size of a ciphertext corresponding to a public-attribute x depends on the worst-case runtime of any (allowed) Turing machine on input x.

In order to get the succinctness property we use a SNARG (succinct non-interactive argument) scheme. A SNARG has a common-reference string crs, which is assumed to be securely generated. Then any user can prove any $\mathbf{NP}$ statement, by computing a *short* proof π. Both the length of the crs and the length of valid proofs depend only on the security parameter, and are independent of the witness size. Moreover, verification can be done efficiently (in time that is independent of the witness size).

$\mathsf{Enc}_{x^*}(b)$ now proceeds as follows. We generate a crs corresponding the underlying SNARG scheme. To encrypt a bit b w.r.t. a public-attribute x, simply compute $\mathsf{Enc}_{x^*}(b)$, where now $x^* = (x, \mathsf{crs}, \mathsf{VK})$. A valid witness for x^* is a tuple of the form (M, σ, π) where σ is a valid signature of the Turing machine M, and π is a *succinct* proof that $M(x) = 1$. Note that the fact that π is succinct makes the witness succinct (independent of the computation time), as desired.

This indeed gives us an ABE for Turing machines. There is a slight drawback to this scheme, which is that it is succinct only for uniform Turing machines. If the Turing machine is non-uniform, and is of the size of the runtime then again the witness becomes non-succinct, and the resulting ABE becomes non-succinct. We would like our ABE scheme to be a generalization of previous work, and in particular to be succinct for any non-uniform Turing machine.

To this end, instead of using a SNARG scheme we use a SNARK (succinct non-interactive argument of knowledge) scheme. In a SNARK scheme the soundness requirement is strengthened, and the security guarantee is that if an adversary $\mathcal{A}$ succeeds in proving that $x \in L$ then an extractor can extract a corresponding witness w from $\mathcal{A}$.

The ABE scheme is exactly as before, except that now a valid witness for $x^* = (x, \mathsf{crs}, \mathsf{VK})$ is a pair (π, t) (without the Turing machine and the signature), where π is a proof-of-knowledge of a Turing machine M and a signature σ such that σ is a valid signature of M and $M(x) = 1$. Now the witness size and the verification time is efficient (independent of the size of the Turing machine or its runtime). We refer the reader to Section 3 for more details on our ABE scheme and the security proof.

To extend the ABE scheme to RAMs, the key for a RAM M with memory D also contains a per-entry signature of the database D: $(M, D, \sigma_M, \{\sigma_{D_i}\}_i)$, where σ_M is a signature on M and $\{\sigma_{D_i}\}_i$ represents an individual signature on every element in the database. The rest of the details are similar.

Functional encryption for Turing machines. To construct a (single-key) functional encryption scheme, we use the reduction of Goldwasser *et al.* [GKP+13] that shows how to construct a (single-key) functional encryption scheme from an ABE scheme. In their work the functions were modeled as circuits, but a similar reduction works when functions are modeled as Turing machines. Namely, in the resulting functional encryption scheme, the secret key size of a Turing machine depends only on the *size* of the Turing machine (as opposed to depending on its worst-case runtime, as in previous constructions). In addition the ciphertext size depends only on the size of the message x being encrypted, and thus is succinct.

Note however, that the time it takes to compute $M(x)$, from sk_M and $\mathsf{Enc}(x)$, does depend on the worst case running time of M. This is inherent to the security requirement, which asserts that the only thing that is leaked about x is $f(x)$, and nothing else. In particular the runtime of M on input x is not leaked.

In many applications it is strongly desired to avoid the worst-case running time during the decryption. Moreover, often releasing the runtime does not compromise security. We show how to convert a (single-key) functional encryption for Turing machines into one where the decryption algorithm, on input sk_M and $\mathsf{Enc}(x)$, runs in time that is polynomial in the runtime of M on input x. This comes at the price of revealing the runtime of f on input x.

The idea is simple, as in [GKP+13]: instead of generating a secret key sk_M for a Turing machine M, we generate many keys $\mathsf{sk}_{M_1}, \ldots, \mathsf{sk}_{M_{\log B_n}}$, where M_i is the Turing machine that runs M for 2^i time steps, and outputs $\perp$ if the computation did not halt in 2^i steps, and otherwise outputs the output of M. The parameter B_n is a global bound on the worst-case runtime of the Turing machines we consider. For polynomial time Turing machines, we can set $B_n = n^{\log n}$. To be able to generate $\log B_n$ keys, we use $\log B_n$ instances of our single-key functional encryption scheme above, by generating fresh keys for every instance of it. Moreover, note that since the underlying functional encryption scheme is for Turing machines, generating sk_{M_i} can be done very efficiently, in time polynomial in the *description size* of M_i, independent on the runtime of M_i.

Encrypting a message x is done exactly as in the underlying scheme. The decryption algorithm, on input a ciphertext $\mathsf{Enc}(x)$ and a secret key $(\mathsf{sk}_{M_1}, \ldots, \mathsf{sk}_{M_{\log B}})$ for the Turing machine M, first tries to decrypt with sk_{M_1}, then tries with sk_{M_2}, and so on. The first time that it succeeds it stops. Note that the runtime of this decryption algorithm depends on the runtime of M on the *specific input* x, denoted by t_x. This is the case since it runs the original decryption algorithm (which runs in the worst-case) only with the secret keys $\mathsf{sk}_{M_1}, \ldots, \mathsf{sk}_{M_{\log t_x}}$, and all the Turing machines $M_1, \ldots, M_{\log t_x}$ run in time at most t_x.

Other primitives. Finally, we show how to use any (single-key) functional encryption scheme for Turing

machines to construct an FHE for Turing machines and a reusable garbling scheme for Turing machines. This again follows the reduction of Goldwasser *et al.* in a straightforward manner. We will not elaborate on the construction of garbling schemes in the rest of this paper because the construction is almost exactly the same as the one in [GKP+13].

1.4 Other Related Work

Perhaps the two main cryptographic constructions that deal with Turing machines and RAM machines directly instead of going through circuits are (1) Oblivious RAMs, first proposed by Goldreich and Ostrovsky [GO96] and studied in many subsequent works [OS97, Ajt10, DMN11, PR10, LO11, GMOT11, GMOT12, LO13]; and (2) succinct arguments, originating from the works of Kilian and Micali [Kil92, Mic00]. In particular, our constructions will make extensive use of succinct non-interactive arguments of knowledge (SNARKs) [GW11, BCCT12, GLR11, DFH12, BCCT13], which are direct descendants of CS proofs.

Oblivious RAM has also been used to perform some two-party computation tasks on the RAM machine model [OS97, GKK+12, LO11], sometimes with input-specific runtimes, but always in an interactive setting. In contrast, the main focus of this work is non-interactive secure computation tasks, where input-specific runtimes are more challenging.

In a recent paper, Lu and Ostrovsky [LO12] consider another limitation of circuits versus uniform computational models, in their case RAM machines. They observe that there are tasks in the RAM machine model that take an exponentially large hit when converted to circuits, their most prominent example being binary search. They show how to compile a RAM machine (with a worst-case running time t) into a garbled RAM program whose size and running time are polynomially related to t. Our work and theirs are like apples and oranges. On the one hand, we consider Turing machines whereas their focus is on the RAM machine model. On the other hand, in their case, both the size of their garbled machines and the time required for the garbled machine evaluation depend polynomially on a *worst-case upper bound* t on the running time of the program, whereas we achieve input-specific running times. Furthermore, their garbled circuits are not reusable. We achieve reusability, albeit at the expense of much stronger cryptographic assumptions. We consider the question of achieving the "best of both works" very interesting.

There has been a large body of work on fully homomorphic encryption [Gen09, BV11b, BV11a, GH11, BGV12, Bra12], attribute-based encryption [SW05, GPSW06, GVW13, SW12] and functional encryption [KSW08, LOS+10, LW12, BSW11, GKP+13], all of which use the Boolean circuit model of computation. A singular exception is the recent work of Waters [Wat12] that constructs an attribute-based encryption scheme for finite state machines, a uniform model of computation.

The problem of designing secure solutions avoiding worst-case running times, and working directly with natural models of computation such as RAM machines has been recently investigated in the security community as well. Fletcher, van Dijk and Devadas [FvD12] show solutions for performing specific computations, such as Euclid's algorithm and SAT solvers, on encrypted data avoiding the worst-case curse. Mitchell et al. [MSSZ12] design a programming language for computations on encrypted data, where the computations are represented as programs and not circuits.

1.5 Paper Roadmap

The rest of this paper is organized as follows. Sec. 2 describes extractable witness encryption and how to obtain it from a strengthening of the Garg *et* al. assumption. Next, Sec. 3 presents our ABE for Turing machines scheme. Finally, Sec. 4 shows how to construct functional encryption for Turing machines.

Because of space constraints, we delegate certain sections to the appendix. Our preliminaries are in appendix A: they contain notation and with definitions of witness encryption, signature schemes, and SNARKs. In appendix B, we show that an extractable witness encryption scheme implies a (weak) obfuscator for

the class of point-filter functions. Based on [GK05], this implies wide-ranging impossibility results for the obfuscation of natural functionalities such as pseudo-random functions. Then, in appendix C, we provide definitions for ABE and functional encryption for Turing machines; these are natural adaptations of the circuit-based equivalents with improved efficiency requirements. Appendices D–F provide additional proofs and details referenced in the rest of the paper. Finally, we show how to construct FHE for Turing machines in appendix G.

2 Extractable Witness Encryption

In this section we define and construct an extractable witness encryption under a new computational assumption. This notion is a strengthening of witness encryption as recently defined by Garg *et al.* [GGSW13] (see Definition 2.1).

Recall that a witness encryption scheme is associated with some **NP** complete language L. Any user given an instance x and a message m, can encrypt m with respect to x; this is denoted by $\mathsf{Enc}_x(m)$. Any user given $\mathsf{Enc}_x(m)$ and a valid witness w of x can decrypt x efficiently. On the other hand, if x is not in the language then semantic security is ensured.

An extractable witness encryption scheme is a witness encryption scheme (with syntax defined in Def. A.1) with a strong soundness property. In particular, it is required that any adversary that can decrypt $\mathsf{Enc}_x(m)$ with non-negligible advantage, must "know" a valid witness w corresponding to x.

Definition 2.1 (Extractable security)**.** *A witness encryption scheme for a language $L \in NP$ is secure if for all p.p.t. adversaries A, there exists a p.p.t. extractor E, for all poly q, there exists a poly p, such that for all auxiliary inputs z, such that for all $x \in \{0,1\}^*$, the following holds:*

$$\Pr[b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\kappa, x, b) : A(x, \mathsf{ct}, z) = b] \geq 1/2 + 1/q(|x|)$$
$$\Rightarrow \quad \Pr[E(x,z) = w : \ (x,w) \in R_L] \geq 1/p(|x|).$$

Remark 2.2. *We note that the probability that E outputs a valid witness can be amplified to $1 - \mu(|x|)$ for any negligible function μ simply by running E with independent random coins $p(|x|) \cdot \log(1/\mu(|x|))$ times.*

We next claim that under an appropriate assumption the witness encryption scheme of Garg *et al.* is *extractable*. To this end, let us first recall their scheme.

The witness encryption scheme of Garg *et al.* The witness encryption scheme of Garg *et al.* is w.r.t. the exact set cover language, where instances are of the form $x = (n, T_1, \ldots, T_\ell)$ where each T_i is a subset of $\{1, \ldots, n\}$. The instance x is in the language if and only if there is a subset $\{T_{i_1}, \ldots, T_{i_k}\} \subseteq \{T_1, \ldots, T_\ell\}$ that covers the set $\{1, \ldots, n\}$ exactly.

Their scheme uses a group $\mathbb{G}$ with multi-linear maps. Namely, they assume the existence of groups $\mathbb{G}_1, \ldots, \mathbb{G}_n$, with corresponding generators $g_1, \ldots, g_n$, and a bilinear map e such that for every i, j with $i + j \leq n$, and for every $\alpha, \beta \in \mathbb{Z}_p$ where p is the order of $\mathbb{G}$, it holds that $e(g_i^\alpha, g_j^\beta) = g_{i+j}^{\alpha \cdot \beta}$.

The encryption algorithm $\mathsf{Enc}_x(m)$, where $x = (n, T_1, \ldots, T_\ell)$, is defined as follows: Choose at random $a_1, \ldots, a_n \leftarrow \mathbb{Z}_p$, and set

$$\mathsf{Enc}_x(m) = \left(\left(g_{|T_1|}\right)^{\prod_{j \in T_1} a_j}, \ldots, \left(g_{|T_\ell|}\right)^{\prod_{j \in T_\ell} a_j}, g_n^{\prod_{j=1}^n a_j} \cdot m \right)$$

Given an exact cover $\{T_{i_1}, \ldots, T_{i_k}\}$ of $\{1, \ldots, n\}$ one can efficiently compute the mask $g_n^{\prod_{j=1}^n a_j}$, by pairing the elements $\left(g_{|T_{i_1}|}\right)^{\prod_{j \in T_{i_1}} a_j}, \ldots, \left(g_{|T_{i_k}|}\right)^{\prod_{j \in T_{i_k}} a_j}$.

The DGE No-Exact-Cover assumption. Garg *et al.* proved security based a new computational assumption. Their assumption is on the multi-linear group $\mathbb{G}$, and depends on the actual **NP** complete language L.[4] More specifically, their assumption is the following: For any instance $x = (n, T_1, \ldots, T_\ell)$, if $x \notin L$ then given

$$\left(g_{|T_1|}\right)^{\prod_{j\in T_1} a_j}, \ldots, \left(g_{|T_\ell|}\right)^{\prod_{j\in T_\ell} a_j}$$

it is hard to distinguish $(g_n)^{\prod_{j=1}^{n} a_j}$ from a uniform element in $\mathbb{G}_n$, where $a_1, \ldots, a_n \leftarrow \mathbb{Z}_p$. This assumption immediately implies the security of their scheme (since this assumption essentially assumes security).

The Extractable DGE No-Exact-Cover assumption. We require a stronger security guarantee (since we need an extractable witness encryption), and therefore we rely on an even stronger computational assumption. Our assumption states that if there exists an adversary who can distinguish between the above two distributions, then there exists an extractor who can extract a witness from him. More formally, our assumption states that there exists a poly-time extractor E and for any polynomial p there exists a polynomial q such that for any instance $(n, T_1, \ldots, T_\ell)$ and any poly-size adversary $\mathcal{A}$, if

$$\Pr\left[\mathcal{A}\left(\left(g_{|T_1|}\right)^{\prod_{j\in T_1} a_j}, \ldots, \left(g_{|T_\ell|}\right)^{\prod_{j\in T_\ell} a_j}, (g_n)^{\prod_{j=1}^{n} a_j}\right) = 1\right] -$$
$$\Pr\left[\mathcal{A}\left(\left(g_{|T_1|}\right)^{\prod_{j\in T_1} a_j}, \ldots, \left(g_{|T_\ell|}\right)^{\prod_{j\in T_\ell} a_j}, (g_n)^{r}\right) = 1\right] \geq \frac{1}{p(k)}$$

where $a_1, \ldots, a_n, r$ are uniformly generated in $\mathbb{Z}_p$ and k is the security parameter (which is assumed to be polynomially related to the instance size), then

$$\Pr\left[E(n, T_1, \ldots, T_\ell, \mathcal{A}) = w \text{ s.t. } w \text{ is a valid witness}\right] \geq \frac{1}{q(k)}.$$

This assumption immediately implies the security guarantee of an extractable witness encryption (since this assumption essentially assumes security).

3 Attribute-based Encryption for Turing Machines and RAM machines

We construct an ABE scheme for Turing machines based on three ingredients:

1. an extractable witness encryption scheme $\mathsf{WE} = (\mathsf{WE.Enc}, \mathsf{WE.Dec})$ based on the work of [GGSW13], on which we elaborate in Sec. 2,
2. a succinct argument of knowledge scheme, $\mathsf{SNARK} = (\mathsf{SNARK.Gen}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$, based on the work of [BCCT13] defined in Sec. A.4,
3. an existentially unforgeable signature scheme $\mathsf{SIG} = (\mathsf{SIG.KeyGen}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ [GMR88].

Theorem 3.1. *Assuming the above three primitives, there exists a secure attribute-based encryption scheme (as per Def. C.2) for the class of all (uniform or non-uniform) polynomial time Turing machines and for all polynomial time RAM machines.*

Corollary 3.2. *There exists a secure attribute-based encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines and polynomial time RAM machines under extractable DGE No-Exact-Cover assumption (Sec. 2), "knowledge of exponent assumption" (Sec. 1.2), and the existence of collision-resistant hash functions.*

We will now focus on the construction and proof for Turing machines, and later explain how to obtain ABE for RAM machines, which only entails a few modifications.

[4] Garg *et al.* show that the fact that the assumption depends on the **NP** complete language is inherent. We refer the reader to [GGSW13] for details.

3.1 The Setup

Recall the intuition provided in technique overview, Sec. 1.3; we also refer the reader to appendix C for a definition of ABE for Turing machines.

The language L for SNARK. We define L by defining its relation, R_L. Let R_L be the following instance-witness relation: the instance is of the form $y = (\mathsf{VK}, x, t)$ (a verification key VK for a signature scheme, an input x, and a time bound t) and the witness is of the form $w = (M, \sigma)$, for M a Turing machine and σ a signature. Then, $(y, w) \in R_L$ iff $\mathsf{SIG.Verify}(\mathsf{VK}, M, \sigma) = 1$ and M halts on x in at most t steps and outputs one.

Note that this does not map directly onto the precise format of the universal language $R_{U,c}$ provided in Sec. A.4 which requires the TM to take as input the rest of the instance and the witness; however, it can be easily mapped to that setting using a universal TM U, and we prefer this formulation for simplicity.

Let $(\mathsf{Gen}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$ be a SNARK system for L and let (SIG.KeyGen,SIG.Sign, SIG.Verify) be an existentially unforgeable signature scheme.

The Language L^* for WE. Based on the above language L and the SNARK system (SNARK.Gen, SNARK.Prover, SNARK.Verify) for L, we define a language L^* for the witness encryption scheme using the witness relation R_{L^*} as follows:

$$R_{L^*}\left[x^* = (x, \mathsf{crs}, \mathsf{VK}), w^* = (\pi, t)\right] = 1 \text{ if and only if } \mathsf{SNARK.Verify}(\mathsf{crs}, (\mathsf{VK}, x, t), \pi) = 1.$$

Let $\mathsf{WE} = (\mathsf{WE.Enc}, \mathsf{WE.Dec})$ be an extractable witness encryption scheme for the witness relation R_{L^*}.

Our construction of a Turing machine ABE scheme, $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$, proceeds as follows.

3.2 Construction of ABE for Turing machines

Let $\mathcal{T}$ be the class of (uniform or non-uniform) polynomial time Turing machines for the ABE scheme.

Setup $\mathsf{ABE.Setup}(1^\kappa)$ where κ is the security parameter:

1. Sample a verification key / signing key pair $(\mathsf{VK}, \mathsf{SigSK}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$, and output $\mathsf{mpk} := \mathsf{VK}$ and $\mathsf{msk} := \mathsf{SigSK}$.

Encryption $\mathsf{ABE.Enc}(\mathsf{mpk}, x, b)$ where $\mathsf{mpk} = \mathsf{VK}$, $x \in \{0,1\}^*$ and $b \in \{0,1\}$:

1. Run the SNARK generator SNARK.Gen to get $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa)$.
2. Let $x^* = (x, \mathsf{crs}, \mathsf{VK})$. Compute $\mathsf{ct}_{\mathsf{WE}} \leftarrow \mathsf{WE.Enc}(1^\kappa, x^*, b)$.
3. Output $\mathsf{ct} := (x^*, \mathsf{ct}_{\mathsf{WE}})$.

Key generation $\mathsf{ABE.KeyGen}(\mathsf{msk}, M)$ where M is a Turing machine:

1. Compute $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{SigSK}, M)$ and output $\mathsf{sk}_M := (M, \sigma)$.

Decryption $\mathsf{ABE.Dec}(\mathsf{sk}_M, \mathsf{ct})$ where $\mathsf{sk}_M = (M, \sigma)$ and $\mathsf{ct} = (x^* = (x, \mathsf{crs}, \mathsf{VK}), \mathsf{ct}_{\mathsf{WE}})$:

1. Run M on x and let t be the number of steps after which M halts (note that M is a polynomial time Turing machine so it must halt within a polynomial number of steps).
2. If $M(x) = 0$, output $\perp$ and exit.
3. Otherwise, let $w := (M, \sigma)$ and note that $\big((\mathsf{VK}, x, t), w\big) \in R_L$.
4. Run SNARK.Prover to obtain a proof $\pi \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, (\mathsf{VK}, x, t), w)$.
5. Let $w^* = (\pi, t)$. Compute and output $\mathsf{WE.Dec}(w^*, \mathsf{ct}_{\mathsf{WE}})$.

Proof Intuition. In appendix Sec. D, we formally prove Th. 3.1. Here we only give intuition into the security proof. We start by assuming the ABE scheme is not secure, and reach a contradiction by showing that one can forge signatures using the extractability properties of the WE and SNARK schemes. Therefore, assume there is an adversary for ABE, $A_{\mathsf{ABE}} = (A_{\mathsf{ABE},1}, A_{\mathsf{ABE},2})$. We will show how to construct an adversary A_{WE} for the WE scheme: A_{WE} simply embeds its challenge ciphertext into the ciphertext for A_{ABE} and lets A_{ABE} decide.

Once we have the adversary A_{WE}, by the security definition of WE, we also have an extractor E_{WE} which on input x^*, outputs a valid witness $w^* = (\pi, t)$ of $(x^*, w^*) \in R_{L^*}$.

Using E_{WE}, we construct a prover P^* for the SNARK system that is able to construct an instance $y = (\mathsf{VK}, x, t)$ and a proof π for which the SNARK verifier accepts. By the proof of knowledge property of the SNARK, there exists an extractor E_{SNARK} that outputs a witness for the SNARK language L, namely $w = (M, \sigma)$, such that $(y, w) \in R_L$. This means that $M(x) = 1$ and that σ is a correct signature on M; thus we can use P^* and E_{SNARK} to forge a signature and reach a contradiction.

3.3 ABE for RAMs

In this section, we discuss how to construct ABE for RAMs. This construction is similar to our construction for Turing machines, so we only mention the main differences here: the language L for the SNARK and key generation. Let (M, D) be a RAM pair: a RAM machine M and a memory D.

The language L for SNARK. Let R_L be the following instance-witness relation: the instance is of the form $y = (\mathsf{VK}, x, t)$ (a verification key VK for a signature scheme, an input x, and a time bound t) and the witness is of the form $w = (r, M, \sigma_{(r,M)}, S, \{i, D_i, \sigma_{(r,i,D_i)}\}_{i \in S})$, where r is a nonce, M is a RAM, $\sigma_{(M,r)}$ is a signature on the description of the machine M and the nonce r, S is a set of accesses M makes to the memory, D_i is the value in the i-th slot of memory and σ_{r,i,D_i} is a signature on r and D_i. Then, $(y, w) \in R_L$ iff

1. $\mathsf{SIG.Verify}(\mathsf{VK}, (r, M), \sigma_{(r,M)}) = 1$,
2. $\mathsf{SIG.Verify}(\mathsf{VK}, (r, i, D_i), \sigma_{(r,i,D_i)}) = 1$ for all $i \in S$,
3. M halts on x in at most t, all of its memory queries are in S, and outputs one.

Key generation $\mathsf{ABE.KeyGen}(\mathsf{msk}, M, D)$ where M is a RAM and D its memory:

1. Choose $r \leftarrow \{0,1\}^{\mathrm{poly}(\kappa)}$.
2. Compute $\sigma_{(r,M)} \leftarrow \mathsf{SIG.Sign}(\mathsf{SigSK}, (r, M))$.
3. For every $i \in 1 \ldots |D|$, compute $\sigma_{(r,i,D_i)} \leftarrow \mathsf{SIG.Sign}(\mathsf{SigSK}, (r, i, D_i))$.
4. Output $(r, M, \sigma_{(r,M)}, \{D_i, \sigma_{(r,i,D_i)}\}_{i=1}^{|D|})$.

Key generation runtime and the key size are polynomial in the description of the RAM machine and the size of memory, but they do not depend on the runtime of the RAM machine. The time to decrypt also only depends on the time to run the RAM and not on its worst case running time or on the memory size.

4 Functional encryption for Turing machines

In this section we construct a (single-key) functional encryption scheme for Turing machines from any ABE scheme for Turing machine (such as from the one constructed in Section 3). In a recent work, Goldwasser

et al. [GKP+13] showed how to construct a (single-key) functional encryption scheme from any ABE and FHE scheme, where functions are modeled as circuits. This reduction also gives a (single-key) functional encryption scheme for Turing machines if the ABE we started with is for Turing machines. For completeness, we include the reduction below.

Theorem 4.1. *Assuming we have:*

- *a secure attribute-based encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines, and*
- *a fully homomorphic encryption scheme,*

there is a (single-key) functional encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines.

Theorem 4.2. *Assuming there exists a (single-key) functional encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines, there is a (single-key) input-specific runtime functional encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines.*

Corollary 4.3. *There exists a secure (single-key) functional encryption scheme* FE *and a (single-key) input-specific runtime functional encryption scheme* FE^* *for the class of all (uniform or non-uniform) polynomial time Turing machines under the extractable DGE No-Exact-Cover assumption (Sec. 2), "knowledge of exponent assumption" (Sec. 1.2), and the LWE assumption with circular security.*

We refer the reader to appendix C for a definition of FE for Turing machines. For simplicity, we construct a (single-key) functional encryption scheme for functions outputting one bit; functions with larger outputs can be handled by repeating our scheme below for every output bit. Our construction uses not only an ABE scheme, but also uses an FHE scheme $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ and Yao's garbling scheme $\mathsf{Gb} = (\mathsf{Gb.Garble}, \mathsf{Gb.Enc})$.

Our construction of $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ proceeds similarly to the [GKP+13] construction. The main difference is that instead of circuits we deal with Turing machines, and we need to describe how to obtain a Turing machine M_{FHE} that runs the FHE evaluation of a Turing machine. We only present here this difference and delegate the full construction to appendix F.1.

4.1 FHE.Eval of a Turing machine

We describe a compiler $\mathsf{Compile}_{\mathsf{FHE}}$ that takes as input a Turing machine M and a number of steps t and produces a Turing machine that computes the FHE evaluation of M for t steps. In the following, let $\hat{x}$ denote the FHE encryption of x.

Algorithm 1 ($\mathsf{Compile}_{\mathsf{FHE}}(M, t)$)**.**

1. Transform M into an oblivious Turing machine M_O by applying the Pippenger-Fischer transformation [PF79] for time bound t. This transformation results in a new Turing machine M_O and a transition function δ for M_O. Namely, δ takes as input a tape input bit b, a state state and outputs a new state state', new content b' for the tape location, and a bit next indicating whether to move left or right; namely $\delta(b, \mathsf{state}) = (\mathsf{state}', b', \mathsf{next})$. Let the movement function next be such that $\mathsf{next}(i)$ indicates whether the head on the input tape of M_O should move left or right after step i.
2. Based on (M_O, next), construct a new Turing machine M_{FHE} that takes as input a FHE public key hpk and an input encryption $\hat{x}$. M_{FHE} applies the transition function δ_{FHE} (the FHE evaluation of δ using hpk) t times. Each cell of the tapes of M_O corresponds to an FHE encrypted value for M_{FHE}. The state of M_{FHE} at time i is the FHE encryption of the state of M_O at time i. At step i, the transition function δ_{FHE} takes as input the encrypted bit from the input tape $\hat{b}$ that the head currently points at, the current

encrypted state $\widehat{\mathsf{state}}$, and outputs an encrypted new state $\widehat{\mathsf{state}'}$ and a new content $\widehat{b'}$. To determine whether to move the head left or right, compute $\mathsf{next}(i)$.

3. Output the description of M_{FHE}.

The running time of $\mathsf{Compile}_{\mathsf{FHE}}$ and M_{FHE} is polynomial in t.

5 Input-specific runtime functional encryption for Turing machines

In what follows we show how to convert a (single-key) functional encryption scheme for Turing machines FE into one where the decryption algorithm, on input fsk_M and $\mathsf{FE.Enc}(\mathsf{MPK}, x)$, runs in time that depends on the runtime of M on input x. Denote by FE^* such a functional encryption scheme. (We refer the reader to appendix C.2.2 for the definition of input-specific runtime functional encryption, and to Sec. 1.3 for an overview of our construction).

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

1. Generate $\tau := \log B_n$ independent pair of keys for the FE scheme: $(\mathsf{msk}_i, \mathsf{mpk}_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$.
2. Output $\mathsf{MPK} := (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_\tau)$ and $\mathsf{MSK} := (\mathsf{msk}_1, \ldots, \mathsf{msk}_\tau)$.

Key Generation $\mathsf{FE}^*.\mathsf{KeyGen}(\mathsf{MSK}, M)$**:** with $\mathsf{MSK} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_\tau)$.

1. Let M_i be the Turing machine that runs M for 2^i steps and outputs $M(x)$ if M finishes in that number of steps, otherwise, it outputs $\perp$. Let t_i be the number of steps M_i runs for.[5]
2. Let $\mathsf{fsk}_{M_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}_i, M_i, t_i)$, for $i = 1 \ldots \tau$.
3. Output $\mathsf{fsk}_M := (\mathsf{fsk}_{M_1}, \ldots, \mathsf{fsk}_{M_\tau})$.

Encryption $\mathsf{FE}^*.\mathsf{Enc}(\mathsf{MPK}, x)$ with $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_\tau)$

1. Compute $\mathsf{ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}_i, x)$ for $i = 1 \ldots \tau$.
2. Output $\mathsf{ct} := (\mathsf{ct}_1, \ldots, \mathsf{ct}_\tau)$.

Decryption $\mathsf{FE}^*.\mathsf{Dec}(\mathsf{fsk}_M, \mathsf{ct})$**:** with $\mathsf{fsk}_M = (\mathsf{fsk}_{M_1}, \ldots, \mathsf{fsk}_{M_\tau})$ and $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\tau)$.

1. Starting with $i = 1$, repeat until $v \neq \perp$:
   (a) $v \leftarrow \mathsf{FE.Dec}(\mathsf{fsk}_{M_i}, \mathsf{ct}_i)$
   (b) $i \leftarrow i + 1$
2. Output v.

Based on this construction, we prove Th. 4.2 in appendix F.2.

References

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

[Ajt10] Miklós Ajtai. Oblivious RAMs without cryptographic assumptions. In *STOC*, pages 181–190, 2010.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *STOC*, 2013.

[5] Note that t_i may be slightly larger than 2^i, since t_i is the number of steps it takes to simulate a Turing machine that runs for 2^i steps.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

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

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

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

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

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[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.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74. 2012.

[DMN11] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious RAM without random oracles. In *TCC*, pages 144–163, 2011.

[FvD12] Christopher Fletcher, Marten van Dijk, and Srinivas Devadas. Towards an interpreter for efficient encrypted computation. In *ACM CCS Cloud Computing Security Workshop*, 2012.

[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. http://eprint.iacr.org/.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GH11] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *FOCS*, pages 107–109, 2011.

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

[GKK+12] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM CCS*, pages 513–524, 2012.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. In *STOC*, 2013.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. Cryptology ePrint Archive, Report 2011/456, 2011. http://eprint.iacr.org/.

[GMOT11] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In *CCSW*, pages 95–100, 2011.

[GMOT12] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *SODA*, pages 157–167, 2012.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, pages 281–308, 1988.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[Gol04] Oded Goldreich. *Foundations of Cryptography - Basic Applications*, volume 2. Cambridge University Press, 2004.

[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.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[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.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

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

[LO11] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. Cryptology ePrint Archive, Report 2011/384, 2011. http://eprint.iacr.org/.

[LO12] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. *Cryptology ePrint Archive, Report 2012/601*, 2012. http://eprint.iacr.org/.

[LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *TCC*, pages 377–396, 2013.

[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.

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

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MSSZ12] J.C. Mitchell, R. Sharma, D. Stefan, and J. Zimmerman. Information-flow control for programming on encrypted data. Cryptology ePrint Archive, Report 2012/205, 2012. http://eprint.iacr.org/.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.

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

[PR10] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *CRYPTO*, pages 502–519, 2010.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.

[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. http://eprint.iacr.org/.

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

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

A Preliminaries

A.1 Notation

We 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 $\mathrm{polylog}(\kappa)$.

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 $\{X_\kappa\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \{Y_\kappa\}_{\kappa\in\mathbb{N}}$) 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 $\kappa \in \mathbb{N}$.

Given a Turing machine M and an input x, let $\mathsf{runtime}(M, x)$ denote the running time of M on x.

A.2 Witness encryption (WE)

We now define the syntax and security definition of witness encryption as defined by Garg et al. [GGSW13], although we use a stronger security definition as detailed in Sec. 2.

Definition A.1 (Witness Encryption)**.** *A witness encryption for a language $L \in NP$ with corresponding witness relation R_L consists of two polynomial-time algorithms* $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$ *such that*

- *Encryption* $\mathsf{WE.Enc}(1^\kappa, x, b)$*: takes as input a security parameter κ, $x \in \{0,1\}^*$ and a bit b and outputs a ciphertext* ct*.*
- *Decryption* $\mathsf{WE.Dec}(w, \mathsf{ct})$*: takes as input $x \in \{0,1\}^*$ and a ciphertext* ct *and outputs a bit b.*

Correctness: *For all $(x, w) \in R_L$, for all bits b, for every sufficiently large security parameter κ*

$$\Pr[\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\kappa, x, b) : \mathsf{WE.Dec}(w, \mathsf{ct}) = b] = 1 - \mathrm{negl}(\kappa).$$

Definition A.2 (Security)**.** *A witness encryption scheme for a language $L \in NP$ is secure if for any $x \notin L$, for all p.p.t. adversaries A, for all sufficiently large security parameters κ:*

$$|\Pr[A(\mathsf{WE.Enc}(1^\kappa, x, 0)) = 1] - \Pr[A(\mathsf{WE.Enc}(1^\kappa, x, 1)) = 1]| = \mathrm{negl}(\kappa).$$

A.3 Signature schemes

For completeness, we briefly remind the security definition of existentially unforgeable signatures schemes (for more details, see [Gol04]).

Definition A.3. *A signature scheme $\mathsf{SIG} = (\mathsf{SIG.KeyGen}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ is unforgeable, if for all p.p.t. adversaries A, for all sufficiently large, letting $\mathcal{O}_{\mathsf{SigSK}}$ be a signing oracle that on input m outputs $\mathsf{SIG.Sign}(\mathsf{SigSK}, m)$, we have*

$$\Pr[(\mathsf{VK}, \mathsf{SigSK}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa); (m, \sigma) \leftarrow A^{\mathcal{O}_{\mathsf{SigSK}}}(\mathsf{VK}) : \\ \mathsf{SIG.Verify}(\mathsf{VK}, m, \sigma) = 1 \text{ and } A \text{ did not ask for } m \text{ to } \mathcal{O}_{\mathsf{SigSK}}] < \mathrm{negl}(\kappa).$$

A.4 Succinct arguments of knowledge (SNARK)

Using succinct arguments of knowledge (abbreviated SNARKs), a prover can prove to a verifier that an instance y is in a language. We use SNARKs that are publicly-verifiable, fully-succinct, and have adaptive soundness (see [BCCT13] for definitions of these properties). Bitansky, Canetti, Chiesa and Tromer [BCCT13] constructed such SNARKs.

We present the definition of SNARKs adapted from [BCCT13]. We define the universal relation as a canonical form to represent verification-of-computation problems.

Definition A.4. *The universal relation is the set R_U of instance-witness pairs $(y, w) = \big((U, x, t), w\big)$, where $|y|, |w| \leq t$ and U is a Turing machine, such that U accepts (x, w) after at most t steps. We denote by L_U the universal language corresponding to R_U. For any $c \in \mathbb{N}$, the universal NP relation is the set $R_{U,c}$ defined as R_U with the additional constraint that $t \leq |x|^c$.*

A SNARK is a triple of algorithms $(\mathsf{SNARK.Gen}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$ that works as follows.

- The generator $\mathsf{SNARK.Gen}$, on input the security parameter κ, samples a reference string crs (since we consider publicly-verifiable SNARKs, the crs can also contain the public verification state). $\mathsf{SNARK.Gen}$ also takes as input a time bound B but we set this to be $B = \kappa^{\log \kappa}$ which will never be achieved for an NP language. Therefore, for simplicity, we do not make B explicit from now on.
- The honest prover $\mathsf{SNARK.Prover}(\mathsf{crs}, y, w)$ produces a proof π for the statement $y = (U, x, t)$ given a valid witness w.
- The verifier $\mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi)$ takes as input the crs, the instance y and a proof π and deterministically verifies π.

The SNARG is adaptive if the prover may choose the statement after seeing crs.

Definition A.5 (SNARK)**.** *A triple of algorithms $(\mathsf{SNARK.Gen}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$ for the relation $R_{U,c}$, where $\mathsf{SNARK.Gen}$ is probabilistic and $\mathsf{SNARK.Verify}$ is deterministic, is a SNARG if the following conditions are satisfied:*

- *Completeness: For every large enough security parameter $\kappa \in \mathbb{N}$, and every instance-witness pair $(y, w) = \big((U, x, t), w\big) \in R_U$,*

$$\Pr[\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa); \pi \leftarrow P(\mathsf{crs}, y, w) : \mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi) = 1]$$

- *Proof of knowledge: For every polynomial-size prover P^* there exists a polynomial-size extractor E_{P^*} such that for every large enough security parameter $\kappa \in \mathbb{N}$, every auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\kappa)}$, and every constant c:*

$$\Pr[\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa); (y, \pi) \leftarrow P^*(\mathsf{crs}, z); w \leftarrow E_{P^*}(\mathsf{crs}, z) : \\ \mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi) = 1 \text{ and } (y, w) \notin R_{U,c}] = \mathrm{negl}(\kappa).$$

- *Efficiency: There exists a universal polynomial p such that, for every large enough security parameter κ, and every instance $y = (M, x, t)$,*
  - *the generator $\mathsf{SNARK.Gen}(1^\kappa)$ runs in time $p(\kappa)$;*
  - *the prover $\mathsf{SNARK.Prover}(\mathsf{crs}, y, w)$ runs in time $p(k + |U| + |x| + t)$;*
  - *the verifier $\mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi)$ runs in time $p(k + |U| + |x|)$;*
  - *an honestly generated proof has size $p(k)$.*

Theorem A.6 ([BCCT13])**.** *There exists a SNARK scheme as in Def. A.5 under "knowledge-of-exponent" assumptions.*

B Witness Encryption and Obfuscation of Point Filter Functions

It turns out that the notion of extractable witness encryption is very similar to another task that was already well-known in the cryptographic literature, namely obfuscating point-filter functions, first defined by Goldwasser and Kalai [GK05]. Informally, point-filter functions for a language $L \in \mathbf{NP}$ with witness relation R_L are a class of functions $\{\delta_{x,b}\}$, indexed by a string $x \in \{0,1\}^n$ and a bit $b \in \{0,1\}$ that behave as follows:

$$\delta_{x,b}(w) = \begin{cases} (x, b) & \text{if } (x, w) \in R_L \\ (x, \bot) & \text{otherwise} \end{cases}$$

In other words, the description of the function reveals the string x, but the function reveals the "secret bit" b only when given a witness w such that $(x, w) \in R_L$.

First, we formally show that an extractable witness encryption scheme for a language L can be used to construct a (weak) obfuscator for the class of point-filter functions for L.

Secondly, Goldwasser and Kalai [GK05] show that the existence of (even weak) obfuscators for the class of point-filter functions for some $\mathbf{NP}$-complete language L gives wide-ranging impossibility results for the obfuscation of natural functionalities such as pseudo-random functions. Thus, we recover these impossibilities under the assumption that an extractable version of the [GGSW13] assumption holds.

We start by defining the class of point-filter functions, and the notion of weak obfuscation with dependent auxiliary inputs from [GK05].

Definition B.1. *For every language $L \in \mathbf{NP}$, the class $\Delta^L = \{\Delta^L_n\}_{n \in \mathbb{N}}$ is a class of point-filter functions, where $\Delta^L_n = \{\delta_{x,b}\}_{x \in \{0,1\}^n, b \in \{0,1\}}$ and*

$$\delta_{x,b}(w) = \begin{cases} (x, b) & \textit{if } (x, w) \in R_L \\ (x, \bot) & \textit{otherwise} \end{cases}$$

Definition B.2 ([BGI+12, GK05]). *A probabilistic algorithm $\mathcal{O}$ is a weak obfuscator for a class of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ if the following three conditions are satisfied:*

- (Functionality) *For every $n \in \mathbb{N}$ and every $C \in \mathcal{C}_n$, $\mathcal{O}(C)$ describes a circuit that computes the same function as C.*
- (Polynomial Blowup): *There is a polynomial $\ell(\cdot)$ such that for every $n \in \mathbb{N}$ and $C \in \mathcal{C}_n$, $|\mathcal{O}(C)| \leq \ell(|C|)$.*
- (Weak "Virtual Black-box" Property with dependent auxiliary inputs): *For every PPT adversary $\mathcal{A}$ and every polynomial $s(\cdot)$, there exists a PPT simulator $\mathcal{S}$ such that for every $n \in \mathbb{N}$, every $C \in \mathcal{C}_n$, every polynomial-time computable predicate $\pi(\cdot)$, every polynomial-size auxiliary input z (note that z may depend on the circuit C), we have:*

$$\left|\Pr[\mathcal{A}(\mathcal{O}(C), z) = \pi(C, z)] - \Pr[\mathcal{S}^C(1^n, z) = \pi(C, z)]\right| < 1/s(n)$$

Lemma B.3. *For every language L, if there is an extractably secure witness encryption scheme for L, then the class of functions Δ^L is weakly obfuscatable with dependent auxiliary input.*

Proof. Assume that there is a witness encryption scheme $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$ for the language L. We construct an obfuscator for the class $\Delta^L = \{\Delta^L_n\}_{n\in\mathbb{N}}$.

On input a pair $(x, b) \in \{0,1\}^n \times \{0,1\}$, the obfuscator $\mathcal{O}$ for the circuit $\delta_{x,b} \in \Delta^L_n$ works as follows:

- Compute $c \leftarrow \mathsf{WE.Enc}(x, b)$.
- Output (x, c) as the description of the obfuscated circuit $\mathcal{O}(\delta_{x,b})$.

The obfuscated circuit $\mathcal{O}(\delta_{x,b})$ works as follows: on input $w \in \{0,1\}^*$, output $b' \leftarrow \mathsf{WE.Dec}(c, w)$.

The functionality property of the obfuscator follows from the (perfect) correctness of the witness encryption scheme. The polynomial blowup property follows from the fact that both the algorithms WE.Enc and WE.Dec run in polynomial time.

Let A be an obfuscation adversary that obtains an obfuscated circuit (x, c) and an auxiliary input z. Let $s(n)$ be a polynomial function. For every $x \in \{0,1\}^n$, $b \in \{0,1\}$ and $z \in \{0,1\}^*$, define the function

$$\boxed{\alpha(x, b, z) = \alpha_A(x, b, z) := \Pr[c \leftarrow \mathsf{WE.Enc}(x, b) : A(x, c, z) = 1]}$$

where the probability is over the coin-tosses of the encryption algorithm WE.Enc and the adversary A. In the sequel, we omit the subscript A when it is clear from the context.

We first show that there is an efficient way to test whether $\alpha(x, 0, z)$ and $\alpha(x, 1, z)$ are sufficiently far apart:

Claim B.4. *For every polynomial function s, there is a PPT algorithm* TestA *that, on input x and z, outputs:*

- `far` *if* $|\alpha(x, 0, z) - \alpha(x, 1, z)| \geq 2/s(n)$*; and*
- `close` *if* $|\alpha(x, 0, z) - \alpha(x, 1, z)| < 1/s(n)$.

with probability $1 - \mathrm{negl}(n)$.

Proof. The algorithm TestA repeats the following procedure $4n \cdot (s(n))^2$ times using independent randomness.

1. Expt0: Generate a ciphertext $c \leftarrow \mathsf{WE.Enc}(x, 0)$ and run A on input (x, c, z).

2. Expt1: Generate a ciphertext $c \leftarrow \mathsf{WE.Enc}(x, 1)$ and run A on input (x, c, z).

Let $\beta_0(n)$ be the number of times A outputs 1 in Expt0 and $\beta_1(n)$ be the number of times A outputs 1 in Expt1. If $|\beta_0 - \beta_1| \geq 6n \cdot s(n)$, output `far`, otherwise output `close`.

The expected number of times A outputs 1 in Expt_b is $4n \cdot s(n)^2 \cdot \alpha(x, b, z)$ (for $b \in \{0, 1\}$). By an application of the Chernoff-Hoeffding bound, we get

$$\Pr[|\beta_0(n) - 4n \cdot s(n)^2 \cdot \alpha(x, 0, z)| \geq n \cdot s(n)] \leq 2 \cdot e^{-2 \cdot 4n \cdot (s(n))^2 \cdot (1/4s(n))^2} = 2e^{-n/2}$$
$$\Pr[|\beta_1(n) - 4n \cdot s(n)^2 \cdot \alpha(x, 1, z)| \geq n \cdot s(n)] \leq 2 \cdot e^{-2 \cdot 8n \cdot (s(n))^2 \cdot (1/8s(n))^2} = 2e^{-n/2}$$

Suppose $|\alpha(x, 0, z) - \alpha(x, 1, z)| \geq 2/s(n)$. Then,

$$|\beta_1(n) - \beta_0(n)| \geq 4n \cdot s(n)^2 \cdot |\alpha(x, 1, z) - \alpha(x, 0, z)| - 2n \cdot s(n) \geq 6n \cdot s(n)$$

with probability $1 - 2 \cdot e^{-n/2} = 1 - \mathrm{negl}(n)$. Thus, the test outputs `far`.

Suppose $|\alpha(x, 0, z) - \alpha(x, 1, z)| < 1/s(n)$. Then,

$$|\beta_1(n) - \beta_0(n)| \leq 4n \cdot s(n)^2 \cdot |\alpha(x, 1, z) - \alpha(x, 0, z)| + 2n \cdot s(n) < 6n \cdot s(n)$$

with probability $1 - 2 \cdot e^{-n} = 1 - \mathrm{negl}(n)$. Thus, the test outputs `close`.

This finishes the proof of the claim. □

The next claim shows that if $\alpha(x, 0, z)$ and $\alpha(x, 1, z)$ are sufficiently different, then there is an extractor algorithm that outputs the witness for x.

Claim B.5. *For every polynomial function s, there is a PPT algorithm E and a negligible function μ such that if $|\alpha(x, 0, z) - \alpha(x, 1, z)| \geq 2/s(n)$, then,*

$$\Pr[E(x, z) = w : (x, w) \in R_L] \geq 1 - \mu(n)$$

Proof. If $|\alpha(x, 0, z) - \alpha(x, 1, z)| \geq 2/s(n)$, by definition

$$\left| \Pr[c \leftarrow \mathsf{WE.Enc}(x, 0) : A(x, c, z) = 1] - \Pr[c \leftarrow \mathsf{WE.Enc}(x, 1) : A(x, c, z) = 1] \right| \geq 2/s(n)$$

Then, there is an algorithm B that predicts the bit b given x, the ciphertext $c \leftarrow \mathsf{WE.Enc}(x, b)$ and the auxiliary input z. Namely,

$$\Pr[c \leftarrow \mathsf{WE.Enc}(x, b) : B(x, c, z) = b] \geq 1/2 + 1/s(n)$$

By the definition of extractable security for witness encryption and Remark 2.2, there is an extractor E and a negligible function μ such that

$$\Pr[E(x, z) = w : (x, w) \in R_L] \geq 1 - \mu(n)$$

This finishes the proof of the claim. □

We are now ready to construct the obfuscation simulator S for the adversary A. The simulator, on input z and access to an oracle for the function $\delta_{x,b}$, works as follows.

- Query the oracle for $\delta_{x,b}$ on input 0^n to get $(x, \perp)$.[6]

[6] If 0^n happens to be a witness for x, the simulator gets (x, b) which makes her life even easier.

- Run the algorithm TestA (from Claim B.4) on inputs x and z to get an answer which is either 'far' or 'close'.
- If TestA returns `close`, then run $A(\mathcal{O}(\delta_{x,0}), z))$ and output whatever A outputs.
- If TestA returns `far`, then do the following:
  - Run $E(x, z)$ to obtain a witness w. If $(x, w) \notin R_L$, output a uniformly random bit. Otherwise, proceed as below.
  - Query the oracle $\delta_{x,b}$ on input w and obtain an answer (x, b).
  - Run $A(\mathcal{O}(\delta_{x,b}), z)$ and output whatever A outputs.

We now show that this simulator satisfies the weak virtual black-box property for obfuscation. Consider two cases.

Case 1. TestA returns `close`: Then, we have:

$$\begin{aligned}
&\left|\Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : A(x,c,z) = \pi(x,b,z)] - \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : S^C(1^n, z) = \pi(x,b,z)]\right| \\
&= \left|\Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : A(x,c,z) = \pi(x,b,z)] - \Pr[c \leftarrow \mathsf{WE.Enc}(x,0) : A(x,c,z) = \pi(x,b,z)]\right| \\
&= \frac{1}{2} \cdot \left|\Pr[c \leftarrow \mathsf{WE.Enc}(x,1) : A(x,c,z) = \pi(x,1,z)] - \Pr[c \leftarrow \mathsf{WE.Enc}(x,0) : A(x,c,z) = \pi(x,1,z)]\right| \\
&= \frac{1}{2} \cdot |\alpha(x,1,z) - \alpha(x,0,z)| < 1/s(n) \qquad (1)
\end{aligned}$$

where the last inequality is because whenever TestA returns `close`, $|\alpha(x, 1, z) - \alpha(x, 0, z)| < 2/s(n)$.

Case 2. TestA returns `far`: Since TestA returns `far`, $|\alpha(x, 1, z) - \alpha(x, 0, z)| \geq 1/s(n)$. By Claim B.5, this means that there is a polynomial function p, a PPT algorithm E and a negligible function μ such that

$$\Pr[E(x, z) = w : (x, w) \in R_L] \geq 1 - \mu(n)$$

The extractor runs in time $p(s(n))$.

Note that as long as the extractor E returns a valid witness for x, the simulator S has all the information necessary to run a perfect simulation. Denote the event that $E(x, z)$ returns a valid witness for x as good. Now,

$$\begin{aligned}
&\Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : S^C(1^n, z) = \pi(x,b,z)] \\
&= \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : S^C(1^n, z) = \pi(x,b,z) \mid \mathsf{good}] \cdot \Pr[\mathsf{good}] \\
&\qquad + \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : S^C(1^n, z) = \pi(x,b,z) \mid \overline{\mathsf{good}}] \cdot \Pr[\overline{\mathsf{good}}] \\
&= \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : A(x,c,z) = \pi(x,b,z)] \cdot \Pr[\mathsf{good}] + \frac{1}{2} \cdot \Pr[\overline{\mathsf{good}}]
\end{aligned}$$

Thus,

$$\begin{aligned}
&\left|\Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : A(x,c,z) = \pi(x,b,z)] - \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : S^C(1^n, z) = \pi(x,b,z)]\right| \\
&= \Pr[\overline{\mathsf{good}}] \cdot (1/2 - \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : A(x,c,z) = \pi(x,b,z)]) \\
&\leq \Pr[\overline{\mathsf{good}}] \leq \mu(n) \qquad (2)
\end{aligned}$$

Combining Equations 1 and 2, we get

$$\big|\Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : A(x,c,z) = \pi(x,b,z)] \quad - \Pr[c \leftarrow \mathsf{WE.Enc}(x,b) : S^C(1^n,z) = \pi(x,b,z)]\big| \\ \leq \mathsf{max}(1/s(n), \mu(n)) = 1/s(n)$$

Note that since the simulator runs the extractor for the witness encryption scheme, it runs in time that depends polynomially on $s(n)$. This is essentially the reason that we are only able to show a weak obfuscator for Δ^L. □

Further Remarks. We note that a weaker variant of extractable security for witness encryption is sufficient for our obfuscator construction. In particular, the following notion of weak extractable security is sufficient for our proof:

Definition B.6 (Weak Extractable security)**.** *A witness encryption scheme for a language $L \in NP$ is weakly extractable if for all p.p.t. adversaries A, for all poly q, there exists a p.p.t. extractor E and a negligible function μ such that for all auxiliary inputs z, such that for all $x \in \{0,1\}^*$, the following holds:*

$$\Pr[b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{WE.Enc}(x,b) : A(x,\mathsf{ct},z) = b] \geq 1/2 + 1/q(|x|)$$

$$\Rightarrow \quad \Pr[E(x,z) = w] \geq 1 - \mu(|x|).$$

Here, the running time of the extractor can depend polynomially on $q(|x|)$.

The key point in the proof where this distinction appears is in Claim B.5 which only uses the weak extractability property above.

More interestingly, we are also able to show how to use a weak obfuscator for a class of point-filter functions Δ^L to construct a witness encryption scheme for L. This shows the equivalence between the two notions. We postpone the proof to the full version.

Implications to the Impossibility of Obfuscation. Goldwasser and Kalai [GK05] show that if there are weak obfuscators for the class Δ^L for any $\mathbf{NP}$-complete language L, then there are no weak obfuscators for a wide variety of cryptographic tasks including pseudorandom function generators, digital signature schemes, public-key decryption algorithms, and more generally any class of functions with sufficient pseudo-entropy. More formally:

Lemma B.7 ([GK05])**.** *Assume that for some $\mathbf{NP}$-complete language L, the class Δ^L is weakly obfuscatable with dependent auxiliary input. Then, no class of circuits $\mathcal{C}$ with super-polynomial pseudo-entropy is even weakly obfuscatable with dependent auxiliary input.*

As an immediate corollary of Lemma B.3 and Lemma B.7, we obtain the following theorem:

Theorem B.8. *Assume that there is an extractably secure witness encryption scheme for some $\mathbf{NP}$-complete language L. Then, no class of circuits $\mathcal{C}$ with super-polynomial pseudo-entropy is even weakly obfuscatable with dependent auxiliary input. In particular, under this assumption, no class of circuits implementing a pseudorandom function is weakly obfuscatable with dependent auxiliary input.*

C Definitions for Turing machines

C.1 Attribute-based encryption (ABE) for Turing machines

We define the syntax and security of ABE for Turing machines.

Definition C.1 (ABE for Turing machines)**.** *An attribute-based encryption scheme* ABE *for a class of Turing machines $\mathcal{T}$ is a tuple of four algorithms (*ABE.Setup*,* ABE.KeyGen*,* ABE.Enc*,* ABE.Dec*), the first three of which are p.p.t., such that:*

- $\mathsf{ABE.Setup}(1^\kappa)$ *takes as input the security parameter 1^κ and outputs a master public key* mpk *and a master secret key* msk*.*
- $\mathsf{ABE.KeyGen}(\mathsf{msk}, M)$ *takes as input the master secret key* msk*, a Turing machine $M \in \mathcal{T}$, and outputs a key sk_M.*
- $\mathsf{ABE.Enc}(\mathsf{mpk}, x, b)$ *takes as input the master public key* mpk*, an attribute $x \in \{0,1\}^*$, and a bit b and outputs a ciphertext* ct*.*
- $\mathsf{ABE.Dec}(\mathsf{sk}_M, \mathsf{ct})$ *takes as input a key sk_M and a ciphertext c and outputs a bit.*

Correctness. *For all Turing machines $M \in \mathcal{T}$, for all attributes $x \in \{0,1\}^*$, for all bits b, for κ sufficiently large,*

$$\Pr\Bigg[(\mathsf{mpk},\mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\kappa); \mathsf{sk}_M \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}, M); \mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}, x, b) : \mathsf{ABE.Dec}(\mathsf{sk}_M, \mathsf{ct}) = \begin{cases} b, & \text{if } M(x) = 1 \\ \bot, & \text{otherwise.} \end{cases} \Bigg] = 1 - \mathrm{negl}(\kappa).$$

Efficiency. *There exists a polynomial p such that the running time of $\mathsf{ABE.Dec}(\mathsf{sk}_M, \mathsf{ct})$ is at most $p(\kappa, \mathsf{runtime}(M, x))$.*

The efficiency property states that the work of the decryption depends on the run time of a Turing machine on the attribute. Since ABE.Setup, ABE.KeyGen and ABE.Enc are p.p.t.-s, their running time depends only on the security parameter and not on the running time of the Turing machines (except for a logarithmic dependency on it).

The literature considers two notions of security: full and selective security, depending on whether the adversary can choose the challenge attribute based on the public key or not, respectively. We only consider the full security definition here because it is the stronger one and we can achieve it.

The literature also considers adaptive versus non-adaptive notions of security indicating whether the adversary can choose the Turing machines M based on the challenge ciphertext or not, respectively. Our construction only provides non-adaptive security, so we only define the non-adaptive case here. Non-adaptive ABE suffices for our applications.

Definition C.2 (Attribute-based encryption security)**.** *Let* ABE *be an attribute-based encryption scheme for a class of Turing machines $\mathcal{T}$ and let $A = (A_1, A_2)$ be an adversary. Consider the following experiment.*

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

1: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$
2: $(x, \mathsf{state}) \leftarrow A_1^{\mathsf{ABE.KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$
3: Choose a bit b at random and let $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}, x, b)$.
4: $b' \leftarrow A_2(\mathsf{state}, \mathsf{ct})$.
5: If, $b = b'$ and for all Turing machines M that A requests to oracle $\mathsf{ABE.KeyGen}(\mathsf{msk}, \cdot)$, we have $M(x) = 0$, output 1, else output 0.

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

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

C.2 Functional Encryption for Turing machines

We define functional encryption for Turing machines, by adapting the definition of FE for circuits from the literature [KSW08, BSW11, GVW12] to Turing machines. We also define a second type of functional encryption, denoted FE^*, which avoids worst case running time during decryption, but necessarily also leaks the running time to the evaluator.

Definition C.3 (Functional Encryption (FE))**.** *A functional encryption scheme* FE *for a class of Turing machines $\mathcal{T}$ is a tuple of four algorithms (*FE.Setup, FE.KeyGen, FE.Enc, FE.Dec*), the first three of which are p.p.t.:*

- $\mathsf{FE.Setup}(1^\kappa)$ *takes as input the security parameter 1^κ and outputs a master public key* fmpk *and a master secret key* fmsk*.*
- $\mathsf{FE.KeyGen}(\mathsf{fmsk}, M, t)$ *takes as input the master secret key* fmsk*, a Turing machine $M \in \mathcal{T}$, a time bound t and outputs a key fsk_M.*
- $\mathsf{FE.Enc}(\mathsf{fmpk}, x)$ *takes as input the master public key* fmpk *and an input $x \in \{0,1\}^*$ and outputs a ciphertext c.*
- $\mathsf{FE.Dec}(\mathsf{fsk}_M, c)$ *takes as input a key fsk_M and a ciphertext c and outputs a value y.*

Correctness. *For every Turing machine $M \in \mathcal{T}$, for any time bound t, for all inputs $x \in \{0,1\}^{\mathrm{poly}(\kappa)}$, for every sufficiently large security parameter κ,*

$$\Pr\left[(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa); \mathsf{fsk}_M \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, M, t); c \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, x) : \mathsf{FE.Dec}(\mathsf{fsk}_M, c) = \begin{cases} M(x), & \text{if } M \text{ halts on } x \text{ in } t \text{ steps} \\ \perp, & \text{otherwise.} \end{cases}\right] = 1 - \mathrm{negl}(\kappa).$$

Efficiency. *There exists a polynomial p such that the running time of* FE.Dec *on a key for (M, t) is at most $p(\kappa, t)$.*

Note that t is usually the worst-case running time of M on inputs of a certain length. The efficiency property says that FE.Dec runs polynomial in the worst-case running time. Since FE.Setup, FE.Enc, and FE.Dec are p.p.t.-s, they only depend on the security parameter and not on the worst-case running time (except for a logarithmic dependency on this time).

C.2.1 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 $M(x)$, for some TM M for which a key was issued.

We adapt the simulation-based definition of FE from [GKP+13] to Turing machines:

Definition C.4 (FE Security)**.** *Let* FE *be a functional encryption scheme for the family of Turing machines* $\mathcal{T}$*. 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: $(M, 1^t, \mathsf{state}_A) \leftarrow A_1(\mathsf{fmpk})$
	3: $\mathsf{fsk}_M \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, M, t)$
	4: $(x, \mathsf{state}'_A) \leftarrow A_2(\mathsf{state}_A, \mathsf{fsk}_M)$
5: $c \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, x)$	5: $\tilde{c} \leftarrow S(\mathsf{fmpk}, \mathsf{fsk}_M, M, M(x), 1^{\lvert x\rvert})$
6: Output (state'_A, c)	6: Output $(\mathsf{state}'_A, \tilde{c})$

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

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

C.2.2 Input-specific runtime functional encryption

The syntax of an input-specific runtime FE scheme, denoted FE^*, is the same as the syntax of FE, except that $\mathsf{FE}^*.\mathsf{KeyGen}(\mathsf{msk}, M)$ does not take a time t as input, and $\mathsf{FE}^*.\mathsf{Dec}(\mathsf{sk}_M, \mathsf{ct})$ runs in time at most polynomial in $\mathsf{runtime}(M, x)$, and *does not depend on the worst-case running time (except logarithmically)*:

Definition C.5 (Efficiency of FE^* for a class of polynomial time Turing machines $\mathcal{T}$)**.** *There exists a polynomial* p *such that for all Turing machines* $M \in \mathcal{T}$*, for all* $x \in \{0,1\}^{\mathrm{poly}(\kappa)}$*, for every sufficiently large* κ*:*

$$\Pr\Bigg[(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE}^*.\mathsf{Setup}(1^\kappa); \mathsf{fsk}_M \leftarrow \mathsf{FE}^*.\mathsf{KeyGen}(\mathsf{fmsk}, M); \mathsf{ct} \leftarrow \mathsf{FE}^*.\mathsf{Enc}(\mathsf{fmpk}, x) : \\ \mathsf{runtime}(\mathsf{FE}^*.\mathsf{Dec}, (\mathsf{fsk}_M, \mathsf{ct})) < p\big(\kappa, |M|, \mathsf{runtime}(M, x)\big)\Bigg] = 1.$$

To enable the decryption function to run in input-specific time, one has to inherently reveal the running time. Therefore, the goal is for the functional encryption scheme to not reveal anything beyond the function result and the running time. The security definition now gives the simulator the running time as well. Let $\mathsf{runtime}(M, x)$ denote the running time of M on x.

Definition C.6 (Security of FE^*)**.** *Let* FE^* *be a functional encryption scheme for the family of Turing machines* $\mathcal{T}$*. 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: $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE}^*.\mathsf{Setup}(1^\kappa)$
	2: $(M, \mathsf{state}_A) \leftarrow A_1(\mathsf{fmpk})$
	3: $\mathsf{fsk}_M \leftarrow \mathsf{FE}^*.\mathsf{KeyGen}(\mathsf{fmsk}, M)$
	4: $(x, \mathsf{state}'_A) \leftarrow A_2(\mathsf{state}_A, \mathsf{fsk}_M)$
5: $c \leftarrow \mathsf{FE}^*.\mathsf{Enc}(\mathsf{fmpk}, x)$	5: $\tilde{c} \leftarrow S^*(\mathsf{fmpk}, \mathsf{fsk}_M, M, M(x), 1^{\lvert x\rvert}, \mathsf{runtime}(M, x))$
6: Output (state'_A, c)	6: Output $(\mathsf{state}'_A, \tilde{c})$

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

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

D Proof of Theorem 3.1

We prove Th. 3.1 with two lemmas showing that the construction above satisfies the desired properties for an ABE scheme for Turing machines (as defined in Sec. C.1).

Lemma D.1. *The* ABE *construction above satisfies the correctness and efficiency property Def. C.1.*

Proof. We first prove correctness. Let $\mathsf{ct} = ((x, \mathsf{crs}, \mathsf{VK}), \mathsf{ct}_{\mathsf{WE}})$ be a ciphertext obtained by running $\mathsf{ABE.Enc}(\mathsf{mpk}, x, b)$, and let (M, σ) be a secret key obtained by running $\mathsf{ABE.KeyGen}(\mathsf{msk}, M)$. We have two cases:

1. *Case:* $M(x) = 1$.
   - Since M on input x accepts in t steps, we know $w = (M, \sigma)$ is a correct witness for $(\mathsf{VK}, x, t) \in L$; namely, $\big(\mathsf{VK}, x, t), (M, \sigma)\big) \in R_L$.
   - By SNARK's completeness, there is an accepting proof $\pi \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, (\mathsf{VK}, x, t), w)$, namely $\mathsf{SNARK.Verify}(\mathsf{crs}, (\mathsf{VK}, x, t), \pi) = 1$. Since $x^* := (x, \mathsf{crs}, \mathsf{VK})$ and $w^* := (\pi, t)$, we have $(x^*, w^*) \in R_{L^*}$.
   - By the correctness of the Witness Encryption scheme, $\mathsf{WE.Dec}(w^*, \mathsf{ct}_{\mathsf{WE}}) = b$.
2. *Case:* $M(x) = 0$.
   - The decryption algorithm determines that $M(x) = 0$ in Step 2 of ABE.Dec and outputs $\perp$.

This shows the correctness of the ABE decryption algorithm. □

Lemma D.2. *The above* ABE *scheme satisfies the security property in Def. C.2.*

Proof. Assume, for contradiction, that there is an adversary $A_{\mathsf{ABE}} = (A_{\mathsf{ABE},1}, A_{\mathsf{ABE},2})$ for the ABE scheme. Namely, there exists a polynomial α, such that for infinitely many κ:

$$\mathrm{Adv}_{\mathsf{ABE},A_{\mathsf{ABE}}} > 1/\alpha(\kappa). \tag{3}$$

Let $\mathcal{D}_\kappa$ be a distribution over pairs (x^*, z^*) as follows.

Algorithm 2 (Sampling from $\mathcal{D}_\kappa$)**.**
1. $(\mathsf{VK}, \mathsf{SigSK}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$.
2. Run $A_{\mathsf{ABE},1}$ on input VK: answer queries to the $\mathsf{ABE.KeyGen}(\mathsf{msk}, \cdot)$ oracle by simply computing signatures using SigSK.
3. Obtain result $(x, \mathsf{state}_{\mathsf{ABE}})$ from $A_{\mathsf{ABE},1}$.
4. Compute $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa)$.
5. Output $(x^* := (x, \mathsf{crs}, \mathsf{VK}), z^* := \mathsf{state}_{\mathsf{ABE}})$.

Let A_{WE} be an adversary for the WE scheme as follows.

Algorithm 3 ($A_{\mathsf{WE}}(x^* = (x, \mathsf{crs}, \mathsf{VK}), \mathsf{ct}, z^* = \mathsf{state}_{\mathsf{ABE}})$)**.**

1. Run $b' \leftarrow A_{\mathsf{ABE},2}(\mathsf{state}_{\mathsf{ABE}}, \mathsf{ct})$.
2. Output b'.

We can see that the input distribution to $A_{\mathsf{ABE},2}$ in the construction of A_{WE} is the same as in the ABE game, so $A_{\mathsf{ABE},2}$ guesses b with advantage $1/\alpha(\kappa)$ based on Eq. (3). Moreover, by construction, when $A_{\mathsf{ABE},2}$ guesses b correctly, so does A_{WE}; therefore,

$$\Pr[(x^*, z) \leftarrow D_\kappa; b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\kappa, x^*, b) : A_{\mathsf{WE}}(x^*, \mathsf{ct}, z) = b] > 1/2 + 1/\alpha(\kappa). \tag{4}$$

By the security definition of the WE scheme, Def. 2.1, for A_{WE}, there exists a p.p.t. E_{WE} such that for all poly q, there exists a polynomial p, such that for all inputs x^* and auxiliary inputs z^*:

$$\begin{aligned}&\Pr[b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\kappa, x^*, b) : A_{\mathsf{WE}}(x^*, \mathsf{ct}, z^*) = b] \geq 1/2 + 1/q(|x^*|)\\ &\quad \Rightarrow \quad \Pr[E_{\mathsf{WE}}(x^*, z^*) = w^*] \geq 1/p(|x^*|).\end{aligned}$$

This statement and Eq. (4) imply that there exists a polynomial β such that, for infinitely many κ,

$$\Pr[(x^*, z^*) \leftarrow \mathcal{D}_\kappa : E_{\mathsf{WE}}(x^*, z^*) = w^* \text{ such that } (x^*, w^*) \in L^*] > 1/\beta(\kappa). \tag{5}$$

This implication follows from a straightforward computation, included for completeness in Lemma E.1 of the appendix.

We now have an extractor E_{WE} that obtains a correct witness with non-negligible probability. Recall that the witness w^* for $x^* = (x, \mathsf{crs}, \mathsf{VK})$ is a SNARK proof $w^* = (\pi, t)$.

We would like to use the extractability property of the SNARK to extract a SNARK witness $w = (M, \sigma)$ for $y = (\mathsf{VK}, x, t)$. Let us now define a prover strategy P^* for which we will use the corresponding extractor E_{SNARK}.

Algorithm 4 ($P^*(\mathsf{crs}, z = (\mathsf{state}_{\mathsf{ABE}}, x, \mathsf{VK}))$)**.**

1. Let $x^* = (x, \mathsf{crs}, \mathsf{VK})$.
2. Run $E_{\mathsf{WE}}(x^*, \mathsf{state}_{\mathsf{ABE}})$ and obtain $w^* = (\pi, t)$.
3. Let $y = (\mathsf{VK}, x, t)$.
4. Output (y, π).

The input distribution to E_{WE} in the construction of P^* is the same as Eq. (4) so E_{WE} outputs a correct witness w^* with chance $\geq 1/\beta(\kappa)$. Whenever, E_{WE} outputs a correct witness, we have $\mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi) = 1$; therefore,

$$\begin{aligned}\Pr[(x, \mathsf{crs}, \mathsf{VK}, \mathsf{state}_{\mathsf{ABE}}) \leftarrow \mathcal{D}_\kappa; z := (\mathsf{state}_{\mathsf{ABE}}, x, \mathsf{VK}); (y, \pi) \leftarrow P^*(\mathsf{crs}, z) :\\ \mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi) = 1] > 1/\beta(\kappa).\end{aligned} \tag{6}$$

The proof of knowledge property of the SNARK says that, for P^* there exists an extractor E_{SNARK} such that

$$\Pr[\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa); (y,\pi) \leftarrow P^*(\mathsf{crs}, z); w \leftarrow E_{P^*}(\mathsf{crs}, z) : \mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi) = 1 \text{ and } (y,w) \notin R_{U,c}] = \mathrm{negl}(\kappa).$$

Combining this with Eq. (6), we obtain that there exists a polynomial γ such that, for infinitely many κ

$$\Pr[(x, \mathsf{crs}, \mathsf{VK}, \mathsf{state}_{\mathsf{ABE}}) \leftarrow \mathcal{D}_\kappa; z := (\mathsf{state}_{\mathsf{ABE}}, x, \mathsf{VK}); (y,\pi) \leftarrow P^*(\mathsf{crs}, z); w \leftarrow E_{\mathsf{SNARK}}(\mathsf{crs}, z) : (y,w) \in R_L] > 1/\gamma(\kappa). \quad (7)$$

This implication follows from a simple probability computation, included for completeness in Lemma E.3 in the appendix. We now know that the SNARK prover P^* and the SNARK extractor E_{SNARK} can produce together a x and (M,σ) such that σ is a correct signature on M and $M(x) = 1$, all with non-negligible probability. We now construct a signature adversary:

Algorithm 5 ($A_{\mathsf{SIG}}(\mathsf{VK})$)**.**

1. Run $A_{\mathsf{ABE},1}$ on input VK by implementing the $\mathsf{ABE.KeyGen}(\mathsf{msk}, \cdot)$ oracle as follows. Consider $A_{\mathsf{ABE},1}$ makes t queries.
   (a) on query M_i, send M_i to the signature challenger which responds with σ_{M_i}.
   (b) Return σ_{M_i} to $A_{\mathsf{ABE},1}$.
2. Let $(x, \mathsf{state}_{\mathsf{ABE}})$ be the outcome of $A_{\mathsf{ABE},1}$.
3. Let $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa)$ and $z := (\mathsf{state}_{\mathsf{ABE}}, x, \mathsf{VK})$.
4. Run $P^*(\mathsf{crs}, z)$ and obtain (y,π).
5. Run $E_{\mathsf{SNARK}}(\mathsf{crs}, z)$ and obtain $w = (M,\sigma)$.
6. Output (M,σ).

We can see that the input distributions to P^* and E_{SNARK} in A_{SIG} are the same as in Eq. (7). Therefore, there exists a polynomial γ such that $\big((\mathsf{VK}, x, t), (M,\sigma)\big) \in R_L$, which means that $M(x) = 1$ and $\mathsf{SIG.Verify}(\mathsf{VK}, M, \sigma) = 1$. But all the previous Turing machines M_i for which A_{SIG} asked for signatures had the property that $M_i(x) = 0$. This means that $M \notin \{M_1, \ldots, M_t\}$, which means that A_{SIG} forges a signature for a new Turing machine with an advantage of $1/\beta(\kappa)$.

Based on the unforgeability of the signature scheme, we reach a contradiction and thus complete the proof of the theorem. □

E Lemmas for proof of Theorem D.2

We used the following lemma in the security proof above. The lemma states that if for all inputs y, an extractor has a polynomial advantage is an adversary has a polynomial advantage on that y, then even for a given distribution of y-s, if the adversary has a polynomial advantage, so does the extractor on that distribution.

Lemma E.1. *Let X_A and X_E be functions mapping an input y to an indicator random variable. Assume that for all poly q, there exists a poly p such that for all y, if $\Pr[X_A(y)] \geq 1/2 + 1/q(|y|)$, then $\Pr[E(y)] \geq 1/p(|y|)$.*

Then, we have that for all poly q', there exists a poly p' such that for all distributions $\mathcal{D}$ of inputs y, if $\Pr[y \leftarrow \mathcal{D} : X_A(y)] \geq 1/2 + 1/q'(|y|)$, then $\Pr[y \leftarrow \mathcal{D} : X_E(y)] \geq 1/2 + 1/p'(|y|)$, for $|y|$ sufficiently large.

Proof. Let $S := \{y \in \mathcal{D} : \Pr[X_A(y)] \geq 1/2 + 1/q'^2(|y|)\}$; namely, S is the subset of the support of $\mathcal{D}$ consisting of values y such that $X_A(y)$ is true with an advantage of $1/q'^2(|y|)$.

Let's prove that the probability of a sample from $\mathcal{D}$ is in S is at least $1/q'(|y|)$. Intuitively, the average advantage over samples from $\mathcal{D}$ is $1/q'(|y|)$ so there has to be a non-negligible fraction of samples that have at least advantage $1/q'^2(|y|)$.

Claim E.2. $\Pr[y \leftarrow \mathcal{D} : y \in S] \geq 1/q'(|y|)$.

Proof. Assume for contradiction that $p := \Pr[y \leftarrow \mathcal{D} : y \in S] < 1/q'(|y|)$. Note that for $y \notin S$, $\Pr[X_A(y)] < 1/2 + 1/q'^2(|y|)$, by definition of S.

$$\begin{aligned}
\Pr[y \leftarrow \mathcal{D} : X_A(y)] &= p \Pr[y \leftarrow \mathcal{D} : X_A(y) \text{ given } y \in S] + (1-p) \Pr[y \leftarrow \mathcal{D} : X_A(y) \text{ given } y \notin S] \\
&< p + (1-p)(1/2 + 1/q'^2(|y|)) \\
&= 1/2 + p/2 + 1/q'^2(|y|) - p/q'^2(|y|) \\
&< 1/2 + 1/q'^2(|y|) + p/2 \\
&< 1/2 + 1/q'^2(|y|) + 1/2q'(|y|) \\
&< 1/2 + 1/q'(|y|) \text{ for } |y| \text{ sufficiently large,}
\end{aligned}$$

thus reaching a contradiction. □

By the lemma's hypothesis, there exists a poly p_2 such that for all y, if $\Pr[X_A(y)] \geq 1/2 + 1/q'^2(|y|)$, then $\Pr[E(y)] \geq 1/p_2(|y|)$. Now, we are ready to compute a lower bound on $\Pr[y \leftarrow \mathcal{D} : X_E(y)]$:

$$\begin{aligned}
\Pr[y \leftarrow \mathcal{D} : X_E(y)] &\geq \Pr[y \leftarrow \mathcal{D} : y \in S] \Pr[y \leftarrow \mathcal{D} : X_E(y) \text{ given } y \in S] \\
&\geq 1/q'(|y|) 1/p_2(|y|).
\end{aligned}$$

Letting $p' := q' \cdot p_2$, we concluded the lemma's proof.

The following can be applied to the proof of Lemma D.2, after a change of variables. We need to define z, r, $\mathcal{Z}$, G, E_1 and E_2 from Lemma E.3:

- z is the same and $r := (\mathsf{crs}, y, \pi)$.
- $\mathcal{Z}_\kappa$ is the same as $\mathcal{D}_\kappa$ without the generation of crs.
- Define G as follows. On input z, run $\mathsf{crs} \leftarrow \mathsf{SNARK.Gen}(1^\kappa, B_{|x|})$ and $*y, \pi) \leftarrow P^*(\mathsf{crs}, z)$.
- $E_1(z, r)$ is the following experiment: $w \leftarrow E_{\mathsf{SNARK}}(\mathsf{crs}, z) : (y, w) \in R_L$, and $E_2(z, r)$ is the deterministic event $\mathsf{SNARK.Verify}(\mathsf{crs}, y, \pi)$.

□

Lemma E.3. *Let Z_κ be a set of strings of length $\mathrm{poly}(\kappa)$, let $\mathcal{Z}(\kappa)$ be any distribution over Z_κ, and let G be a p.p.t.. Given two strings z and r, let $E_1(z, r)$ and $E_2(z, r)$ be two random variables with values in $\{$true, false$\}$. Assume that the following holds:*

$$\forall\, z \in Z_\kappa, \Pr[r \leftarrow G(z); E_1(z, r) \text{ and } \overline{E_2(z, r)}] = \mathrm{negl}(\kappa)$$

and there exists a polynomial p_1 such that

$$\Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_2(z, r)] > 1/p_1(\kappa),$$

then there exists a polynomial p_2 such that

$$\Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r)] > 1/p_2(\kappa).$$

Proof. We know that for all $z \in Z_\kappa$

$$\Pr[r \leftarrow G(z); E_1(z,r) \text{ and } \overline{E_2(z,r)}] < 1/p_1^2(\kappa) \tag{8}$$

$$\begin{aligned}
&\Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r) \text{ and } E_2(z,r)] \\
&\quad = 1 - \Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); \overline{E_1(z,r)}] - \Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r) \text{ and } \overline{E_2(z,r)}] \\
&\quad = \Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r)] - \Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r) \text{ and } \overline{E_2(z,r)}] \\
&\quad > 1/p_1(\kappa) - 1/p_1^2(\kappa),
\end{aligned}$$

by the lemma's hypothesis and Eq. (8). Moreover, $\Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r)] \geq \Pr[z \leftarrow \mathcal{Z}_\kappa; r \leftarrow G(z); E_1(z,r) \text{ and } E_2(z,r)]$; thus, there exists a polynomial p_2 as desired. □

F Functional encryption for Turing machines

F.1 Construction for FE for Turing machines

The construction uses a *two-outcome* ABE scheme, denoted ABE_2, as opposed to a standard ABE scheme. In ABE_2, instead of encrypting a message m w.r.t. a public attribute x, one encrypts two messages L_0 and L_1 w.r.t. a public attribute x. Given a secret key sk_M and a ciphertext $\mathsf{Enc}(\mathsf{fmpk}, x, (L_0, L_1))$ one can compute L_b where $b = M(x)$ and L_{1-b} remains hidden. It is straightforward to construct ABE_2 from any secure ABE scheme (see [GKP+13] for details). Moreover, if the original ABE was for Turing machines then the ABE_2 is also for Turing machines. In what follows, we further assume that ABE_2 can encrypt an input of polynomial size rather than one bit (which can be achieved by repetition).

Let $\lambda = \lambda(\kappa)$ be the length of the ciphertexts in the FHE scheme; this refers to both the ciphertexts produced by the FHE encryption algorithm and to the ciphertexts produced by the FHE evaluation algorithm. The construction is mainly the one of [GKP+13], the main difference being in $\mathsf{FE.KeyGen}$ – generating a key for a Turing machine.

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

$$(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{ABE}_2.\mathsf{Setup}(1^\kappa).$$

Output $(\mathsf{fmpk}, \mathsf{fmsk})$ as the master public and secret keys.

Key Generation $\mathsf{FE.KeyGen}(\mathsf{MSK}, M, t)$**:**

1. Compute $M_{\mathsf{FHE}} \leftarrow \mathsf{Compile}_{\mathsf{FHE}}(M, t)$, the Turing machine that runs the homomorphic evaluation of M for time t. Let M^i_{FHE} be the Turing machine that runs M_{FHE} and outputs the i-th bit of M_{FHE}.
2. Run the key generation algorithm of ABE_2 for the TMs M^i_{FHE} to construct secret keys:

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

3. Output the tuple $\mathsf{fsk}_M := (\mathsf{fsk}_1, \ldots, \mathsf{fsk}_\lambda)$ as the secret key for the TM M.

Note that if ABE_2 is a scheme for Turing machines, then the secret keys $\mathsf{fsk}_1, \ldots, \mathsf{fsk}_\lambda$ (and thus the secret key fsk_M) depend only on the size of the (non-uniform) Turing machines M^i_{FHE}. This size does not depend on the runtime (or on the unrolled circuit size) of M^i_{FHE} or M.

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 FHE scheme. Encrypt each bit of x by computing $\psi_i \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, x_i)$. Let $\psi := (\psi_1, \ldots, \psi_n)$ be the encryption of the input x.

2. Run the Yao garbled circuit generation algorithm to produce a garbled circuit for the FHE decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$ together with 2λ labels L^b_i for $i \in [\lambda]$ and $b \in \{0,1\}$. Namely,
$$\left(\Gamma, \{L^0_i, L^1_i\}_{i=1}^\lambda\right) \leftarrow \mathsf{Gb.Garble}(1^\kappa, \mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)),$$
where Γ is the garbled circuit and the L^b_i are the input labels.

3. Produce encryptions $c_1, \ldots, c_\lambda$ using the ABE_2 scheme in the following way. Let
$$c_i \leftarrow \mathsf{ABE}_2.\mathsf{Enc}\left(\mathsf{fmpk},\ (\mathsf{hpk}, \psi), (L^0_i, L^1_i)\right)$$
where (hpk, ψ) comes from the first step, and the labels (L^0_i, L^1_i) come from the second step.

The output of the encryption algorithm is $c = (c_1, \ldots, c_\lambda, \Gamma)$.

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

1. First, 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
$$L^{d_i}_i \leftarrow \mathsf{ABE}_2.\mathsf{Dec}(\mathsf{fsk}_i, c_i) \text{ for } i \in [\lambda],$$
where d_i is a bit representing $M^i_{\mathsf{FHE}}(\mathsf{hpk}, \psi)$.

2. Now, armed with the garbled circuit Γ and the labels $L^{d_i}_i$, run the garbled circuit evaluation algorithm to compute
$$\mathsf{Gb.Eval}(\Gamma, L^{d_1}_1, \ldots, L^{d_\lambda}_\lambda) = \mathsf{FHE.Dec}(\mathsf{hsk}, d_1 d_2 \ldots d_\lambda) = \mathsf{FHE.Dec}(\mathsf{hsk}, M_{\mathsf{FHE}}(\mathsf{hpk}, \psi)) = M(x)$$

We note that even though the size of fsk_M is small (independent of the runtime of M), given fsk_M and $\mathsf{FE.Enc}(\mathsf{MPK}, x)$, the time it takes to compute $M(x)$ is proportional to the *worst-case* running time of M. This is inherent since the security definition ensures that the only information that is leaked about x is $M(x)$, and in particular, does not allow to leak the running time of M on input x.

The proof of Theorem 4.1 is almost identical to the proof of the corresponding theorem in [GKP+13] (where functions are modeled as circuits as opposed to Turing machines). We thus refer the reader to [GKP+13] for the proof.

F.2 Proof of Th. 4.2

Proof of Th. 4.2. We need to prove the correctness, efficiency and security of the scheme.

Correctness. The correctness of FE^* follows from the correctness of the underlying scheme FE. Let M be any Turing machine for which a key sk_M was generated and x be any input for which a ciphertext ct was generated. Since M is a polynomial time TM, M must halt on x before time $B_{|x|}$. Therefore, there exists at least one i such that $2^i \geq \mathsf{runtime}(M, x)$. Let i_0 be the smallest such i. For all $j < i_0$, we have $\mathsf{FE.Dec}(\mathsf{fsk}_{M_j}, \mathsf{ct}_j) = \perp$ so the loop in $\mathsf{FE}^*.\mathsf{Dec}$ continues. When $j = i_0$, we have $\mathsf{FE.Dec}(\mathsf{fsk}_{M_i}, \mathsf{ct}_i) = M(x) \neq \perp$, so $\mathsf{FE}^*.\mathsf{Dec}$ outputs the correct value.

Efficiency (as per Def. C.5) . Let M be any Turing machine for which a key sk_M was generated and x be any input for which a ciphertext ct was generated. Let i be the smallest index such that $\mathsf{runtime}(M, x) \leq 2^i$. 2^i is at most twice the value of $\mathsf{runtime}(M, x)$. At each step $j \leq i$, the runtime of $\mathsf{FE.Dec}(\mathsf{fsk}_{M_i}, \mathsf{ct}_i)$ is a polynomial in t_j and κ; t_j is proportional to $2^j \leq 2^i \leq 2 \cdot \mathsf{runtime}(M, x)$. Therefore, each step takes $\mathrm{poly}(\mathsf{runtime}(M, x), \kappa)$. There are at most $\log B_\kappa = \log^2 \kappa$ such steps, thus efficiency is achieved.

Security (as per Def. C.6). Let us construct a simulator S^* for FE^* as in Def. C.6 based on a simulator S for FE as in Def. C.4.

Algorithm 6 ($S^*(\mathsf{MPK}, \mathsf{fsk}_M, M, M(x), 1^{|x|}, t = \mathsf{runtime}(M, x))$)**.**

Recall that $\mathsf{MPK} = (\mathsf{mpk}_1 \dots \mathsf{mpk}_\tau)$ and $\mathsf{fsk}_M = (\mathsf{fsk}_{M_1} \dots \mathsf{fsk}_{M_\tau})$

1. For each $i = 1, \dots, \tau$, generate ct_i as follows:
   (a) If $t > 2^i$: $\mathsf{ct}_i \leftarrow S(\mathsf{mpk}_i, \mathsf{fsk}_{M_i}, M_i, \perp, 1^{|x|})$.
   (b) Else ($t \leq 2^i$): $\mathsf{ct}_i \leftarrow S(\mathsf{mpk}_i, \mathsf{fsk}_{M_i}, M_i, M(x), 1^{|x|})$.
2. Output $(\mathsf{ct}_1, \dots, \mathsf{ct}_\tau)$.

We can see that S^* is a correct simulation based on the security properties of S. □

G Fully homomorphic encryption for Turing machines

In this section, we provide a construction for FHE^*, a fully homomorphic encryption scheme that avoids worst-case running time. The idea is to use FE^* to enable an evaluator to determine the running time of a TM M on an input x and then to use a standard FHE scheme FHE to perform the FHE evaluation for that runtime.

Theorem G.1. *Assuming there exists an input-specific runtime functional encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines, and assuming the existence of a fully homomorphic encryption scheme (for circuits), there is an input-specific fully homomorphic encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines.*

Corollary G.2. *There exists an input-specific fully homomorphic encryption scheme for the class of all (uniform or non-uniform) polynomial time Turing machines under the extractable DGE No-Exact-Cover assumption (Sec. 2), "knowledge of exponent assumption" (Sec. 1.2), and the LWE assumption with circular security.*

As [GKP+13] already discussed, the FHE evaluator must no longer be able to compute TMs of its choice on the encrypted inputs (in time that is input specific) because the running time of those TMs can leak the input. Therefore, the evaluator must receive a token permitting it to run a certain TM M. Moreover, because FE^* is only secure for a single-key, we cannot generate a FE^* key for an unbounded number of Turing machines. Instead we generate a FE^* key for the universal Turing machine U, that on input a TM M and x

outputs the running time of M on x. We separately provide an FHE encryption of x to be used for a standard FHE evaluation.

Key generation $\mathsf{FHE}^*.\mathsf{KeyGen}(1^\kappa)$**:**

1. $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{FE}^*.\mathsf{Setup}(1^\kappa)$.
2. $(\mathsf{hpk}, \mathsf{hsk}) \leftarrow \mathsf{FHE}.\mathsf{KeyGen}(1^\kappa)$.
3. $\mathsf{sk}_U \leftarrow \mathsf{FE}^*.\mathsf{KeyGen}(\mathsf{MSK}, U)$.
4. Output $\mathsf{PK} = (\mathsf{MPK}, \mathsf{sk}_U, \mathsf{hpk})$ and $\mathsf{SK} = \mathsf{hsk}$.

Encryption $\mathsf{FHE}^*.\mathsf{Enc}(\mathsf{PK}, x, M)$:

1. Produce FE^* encryption $\mathsf{ct}_{\mathsf{FE}^*} = \mathsf{FE}^*.\mathsf{Enc}(\mathsf{MPK}, (x, M))$.
2. Produce FHE encryption $\hat{x} \leftarrow \mathsf{FHE}.\mathsf{Enc}(\mathsf{hpk}, x)$.
3. Output $\mathsf{ct} = (\mathsf{ct}_{\mathsf{FE}^*}, \hat{x})$.

Evaluation $\mathsf{FHE}^*.\mathsf{Eval}(\mathsf{PK}, \mathsf{ct})$ with $\mathsf{PK} = (\mathsf{MPK}, \mathsf{sk}_U, \mathsf{hpk})$ and $\mathsf{ct} = (\mathsf{ct}_{\mathsf{FE}^*}, \hat{x})$:

1. Compute $t \leftarrow \mathsf{FE}^*.\mathsf{Dec}(\mathsf{sk}_U, \mathsf{ct}_{\mathsf{FE}^*})$.
2. Use the Pippenger-Fischer transformation [PF79] on M and time bound t to obtain a circuit C_M that runs M for t steps.
3. Compute and output $\mathsf{ct} = (\mathsf{ct}_{\mathsf{FHE}} := \mathsf{FHE}.\mathsf{Eval}(\mathsf{hpk}, C_M, \hat{x}))$.

Decryption $\mathsf{FHE}^*.\mathsf{Dec}(\mathsf{SK}, \mathsf{ct})$ with $\mathsf{SK} = \mathsf{hsk}$:

1. If $\mathsf{ct} = (\mathsf{ct}_{\mathsf{FE}^*}, \hat{x})$, output $\mathsf{FHE}.\mathsf{Dec}(\mathsf{hsk}, \hat{x})$.
2. If $\mathsf{ct} = (\mathsf{ct}_{\mathsf{FHE}} = \mathsf{FHE}.\mathsf{Eval}(\mathsf{hpk}, C_M, \hat{x})$, output $\mathsf{FHE}.\mathsf{Dec}(\mathsf{hsk}, \mathsf{ct}_{\mathsf{FHE}})$.

Remark G.3. *Note that the run time for* $\mathsf{FHE}^*.\mathsf{KeyGen}$ *is* $\mathrm{poly}(\kappa, |U|)$ *because all algorithms invoked are p.p.t.-s. Therefore, the preprocessing phase for* FHE^* *does not depend on the worst-case runtime of the TMs in* $\mathcal{T}$.

We do not elaborate on correctness and security because they follow from the correctness and security of the FE^* and FHE schemes. Let's prove that the running time is indeed input-specific:

Lemma G.4. $\mathsf{FHE}^*.\mathsf{Eval}$ *runs in time* $\mathrm{poly}(\kappa, |M|, t)$.

Proof. From the proof of Theorem 4.2, we know that the runtime of $\mathsf{FE}^*.\mathsf{Dec}$ is $\mathrm{poly}(\kappa, |U|, \mathsf{runtime}(U, (M, x)))$. Since the running time of U on (M, x) is proportional to the running time of M, the time for $\mathsf{FE}^*.\mathsf{Dec}$ is $\mathrm{poly}(\kappa, |M|, \mathsf{runtime}(M, x))$.

The time to compute the Pippenger Fischer transformation is pseudolinear in t and the time to perform FHE evaluation of a circuit of size pseudolinear in t is polynomial in t.

Therefore, overall, the runtime of $\mathsf{FHE}^*.\mathsf{Eval}$ is $\mathrm{poly}(\kappa, |M|, \mathsf{runtime}(M, x))$, as desired. □