How to Run Turing Machines on Encrypted Data

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

[⋆] MIT CSAIL [†] Microsoft Research [⋈] University of Toronto

Abstract. Algorithms for computing on encrypted data promise to be a fundamental building block of cryptography. The way one models such algorithms has a crucial effect on the efficiency and usefulness of the resulting cryptographic schemes. As of today, almost all known schemes for fully homomorphic encryption, functional encryption, and garbling schemes work by modeling algorithms as circuits rather than as Turing machines.

As a consequence of this modeling, evaluating an algorithm over encrypted data is as slow as the worst-case running time of that algorithm, a dire fact for many tasks. In addition, in settings where an evaluator needs a description of the algorithm itself in some "encoded" form, the cost of computing and communicating such encoding is as large as the worst-case running time of this algorithm.

In this work, we construct cryptographic schemes for computing Turing machines on encrypted data that avoid the worst-case problem. Specifically, we show:

- An attribute-based encryption scheme for any polynomial-time Turing machine and Random Access Machine (RAM).
- A (single-key and succinct) functional encryption scheme for any polynomial-time Turing machine.
- A reusable garbling scheme for any polynomial-time Turing machine.

  These three schemes have the property that the size of a key or of a garbling for a Turing machine is very short: it depends only on the description of the Turing machine and not on its running time.

  Previously, the only existing constructions of such schemes were for depth-d circuits, where all the parameters grow with d. Our constructions remove this depth d restriction, have short keys, and moreover, avoid the worst-case running time.
- A variant of fully homomorphic encryption scheme for Turing machines, where one can evaluate a Turing machine M on an encrypted input x in time that is dependent on the running time of M *on input* x as opposed to the worst-case runtime of M. 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 constructions remove both restrictions.

Our results are obtained via a reduction from SNARKs (Bitanski et al) and an "extractable" variant of witness encryption, a scheme introduced by Garg *et al.*. We prove that the new assumption is secure in the generic group model. We also point out the connection between (the variant of) witness encryption and the obfuscation of point filter functions as defined by Goldwasser and Kalai in 2005.

Keywords: Computing on encrypted data; Functional encryption; Fully homomorphic encryption; Turing machines; Input-specific running time.

1 Introduction

Almost all known cryptographic constructions for computing on encrypted data model algorithms as circuits instead of Turing machines.[1] The most notable such constructions are fully homomorphic encryption, attribute-based encryption, functional encryption, and garbling schemes. There are at least two unfortunate consequences of modeling algorithms as circuits.

The first consequence of using circuits is that evaluating an algorithm A on encrypted data is at least as slow as the *worst-case* running time of algorithm A on all inputs of a certain size. Ideally, the runtime of A on input x should be the time A takes to run on x. The reason for this slowdown is that all the known transformations from Turing machines to circuits essentially work by unrolling loops to their worst-case runtime, and by considering all branches of a computation. Even if the cryptographic overhead of these schemes were zero, such worst-case runtime can still make the computation prohibitively slow: for example, the simplex algorithm for linear programming runs in polynomial time on most instances one encounters in practice, but in exponential time on rare inputs.

Some schemes for computing on encrypted data (such as functional encryption and garbling schemes) require the evaluator to obtain a token for an algorithm A in order to run A on the encrypted data. The second consequence of modeling algorithms as circuits is that the size of the token is as large as the running time of the algorithm, instead of depending only on the description of the algorithm, which can be much shorter.

The earliest example of using circuits for computing on encrypted data is Yao's secure function evaluation protocol [Yao86] which takes as input any polynomial time computable function f – specified by a circuit – and outputs a "garbled circuit" with the same input-output functionality. Such worst-case runtime also affects known two-party and multi-party protocols for general secure function evaluation [Yao86,GMW87,BGW88,CCD88].

More recent constructions for computing on encrypted data also use circuits to model computation and thus suffer from the worst-case slowdown: fully homomorphic encryption schemes (FHE) [Gen09,BV11a,BV11b,BGV12,Bra12], attribute-based encryption (ABE) schemes [GVW13,GGH$^+$13b,GGH13a], and functional encryption (FE) schemes for general functions [SS10,GVW12,GKP$^+$13].

In this work, we construct cryptographic schemes for Turing machines, thus removing the two major limitations of circuits discussed above. We construct attribute-based encryption, single-key (succinct) functional encryption, reusable garbling schemes, and a version of FHE for polynomial-time Turing machines. For each of these schemes, we show that the time to evaluate a Turing machine M on an input x is *input specific*: it depends on the runtime of M on x and not on the worst-case runtime of M on all inputs of length n where $n = |x|$. Moreover, we show that the token an evaluator needs to run a Turing machine M on encrypted data is short: its size depends on the size of *the description of the Turing machine* M and not on M's runtime. Our schemes are for

[1] An exception is the garbling scheme of [LO12] for RAMs, but this scheme also suffers from worst-case running time problem (see Sec. 1.1).

both uniform and non-uniform Turing machines (so in particular, they can compute any circuits).

Since the evaluator can compute a Turing machine M in input-specific runtime, it means that the evaluator necessarily learns the runtime of M on specific inputs. In some of the schemes we consider, such as ABE, such runtime provides no new information to the evaluator. In other schemes, such as FE, the evaluator could learn additional information from the runtime. This is why we also provide a second functional encryption scheme that does not leak the runtime and runs in worst-case time, while still benefiting from the Turing machine model because it has short tokens. Depending on how one weighs leaking runtime versus worst-case performance, one can choose one scheme or the other.

Our schemes are based on extractable witness encryption, a variant of the witness encryption notion of Garg *et al.* [GGSW13]. We show how to obtain such an extractable witness encryption scheme using the construction of Garg *et al.* [GGSW13], by strengthening their assumption with a knowledge property. We prove the new assumption secure in the generic group model. Interestingly, we show that extractable witness encryption is closely related to (weakly) obfuscatable point-filter functions [GK05].

1.1 Our results

We now explain our results in detail.

Attribute-based encryption (ABE) for Turing machines and RAMs. Attribute-based encryption schemes, originally defined by Sahai and Waters [SW05], allow a user holding the master secret key msk to generate a function key sk_f for any predicate f of his choice, where sk_f does not hide f. Using the master public key mpk, anyone can encrypt a message m with respect to an "attribute" x: such a 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 if $f(x) = 1$. On the other hand, if $f(x) = 0$, ABE leaks no information about m and provides semantic security.

Attribute-based encryption is a powerful primitive and has thus received significant attention [GPSW06,LOS$^+$10,LW12,GVW13]. The state-of-the-art is the scheme of Gorbunov *et al.* [GVW13]: based on the LWE assumption, they construct an ABE for the class of all circuits of depth at most d, where the efficiency of the scheme (such as the size of the ciphertexts) decreases polynomially with d. In concurrent work, Garg *et al.* constructed ABE schemes with similar properties [GGH$^+$13b], and an ABE scheme with large ciphertexts [GGSW13], both from candidate multi-linear maps.

In this work, we construct an attribute-based encryption scheme for all circuits, with no restriction on the depth. More importantly, we model functions as Turing machines (with possibly non-uniform advice), as opposed to circuits as in previous work. Computing a function key sk_M, corresponding to a Turing machine M, takes roughly linear time in the *size of the description* 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 Sec. 3 for details). We then show that a modification of our construction provides ABE for RAMs.

Theorem 1 (Informal). *There exists an attribute-based encryption scheme (as defined in Defs. 3, 4) for (uniform or non-uniform) polynomial-time Turing machines and RAMs from the assumptions in Sec. 1.2.*

Interestingly, we show how to extend our ABE scheme beyond Turing machines and RAMs: for example, an evaluator can choose by himself which Turing machines to run on the ciphertexts, as long as they satisfy some property expressed in a function key.

Functional encryption (FE) for Turing machines. Functional encryption, formalized by Boneh, Sahai and Waters [BSW11], is a generalization of attribute-based encryption. In functional encryption, a user holding the master secret key msk can generate a function key sk_f corresponding to a function f; then, anyone having a ciphertext $\mathsf{Enc}(x)$ and a function key sk_f can compute $f(x)$, but learns nothing else about the input x.

So far, the only many-keys FE schemes known (schemes in which the secret key owner can securely release an unbounded number of function keys) are for the inner-product predicates [KSW08,SSW09]. For general functions, Agrawal *et al.* [AGVW12] showed that there does not exist a many-keys FE scheme if one wants to achieve a natural simulation-based security definition[2], so the natural question was to construct a single-key functional encryption scheme for general functions. Sahai and Seyalioglu [SS10], Gorbunov *et al.* [GVW12], and Goldwasser *et al.* [GKP$^+$13] constructed such schemes for circuits. The work of Goldwasser *et al.* [GKP$^+$13] is the first to provide succinct ciphertexts: the ciphertext size is much smaller than the circuit size; they constructed a *succinct* single-key FE scheme for any depth d circuit, where the parameters of the scheme grow with d (but are independent of the circuit size).

In this work, we not only remove this depth-d restriction, but we model functions as (possibly non-uniform) Turing machines, as opposed to circuits as in prior work. Our schemes have short function keys: computing the function key of a Turing machine M depends only on the size of M and does not depend on the runtime of M. We note that in all previous schemes for general functions 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 leaking 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 by *input-specific runtime functional encryption*.

Theorem 2 (Informal). *There exists a single-key (succinct) functional encryption scheme and input-specific runtime functional encryption scheme for (uniform or non-uniform) polynomial-time Turing machines from the assumptions in Sec. 1.2.*

[2] Their lower bound does not apply to weaker security definitions.

Variant of FHE for Turing machines. We construct a variant of FHE where one can evaluate a Turing machine M on a ciphertext $\mathsf{Enc}(x)$ in time that depends on the runtime of P *on the specific input* x. We naturally call this scheme *input-specific FHE*. 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 only the runtime of M is not harmful, and it can provide significant efficiency gains.

Our construction is an improvement of Goldwasser *e*t al. [GKP+13] who showed how to construct input-specific runtime FHE from single-key functional encryption. As in Goldwasser *et al.* [GKP+13], we also encrypt a Turing machine M and x together into a token $\mathsf{tk}_{M,x}$. Producing such a token depends only 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. The reason we provide a token for M at all is 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 entirely. We combine M and x in $\mathsf{tk}_{M,x}$ for a technical reason stemming from the fact that the FE scheme we use in the construction is single-key – we elaborate in our full paper.

Comparing to [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 polynomial-time Turing machine.* Because the ciphertext size in [GKP+13] depended on the depth of the worst-case circuit representation of the class of Turing machines, [GKP+13] 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 restriction and thus applies to any class of Turing machines with runtime upper-bounded by a polynomial.

Theorem 3 (Informal). *There exists an input-specific-runtime fully homomorphic encryption scheme for (uniform or non-uniform) polynomial-time Turing machines based on the assumptions in Sec. 1.2.*

Reusable garbling scheme for Turing machines. Garbling schemes, introduced in the seminal work of Yao [Yao86], have found many applications in cryptography. In such schemes, a user can "garble" a function f and then encode an input x in 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. Some works also considered an authenticity property [BHR12,GVW13], on which we do not dwell. Traditional garbling schemes are one-time: they are secure only if an adversary gets a token for at most one input. A reusable garbling scheme is secure when the adversary gets an unbounded number of tokens.

In known garbling schemes (even non-reusable ones), the size of the garbling is as large as the worst-case runtime of f. Often, the reason is that 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*. The work of [LO12] is an exception from the circuit model: they model computation as RAM, but their scheme still has large garbling size, at least as large as the worst-case running time.

As in our FHE and FE schemes, if one allows leaking the runtime of M on input x, we can additionally avoid worst-case evaluation time and obtain an input-specific reusable garbling scheme: given a garbling for a Turing machine M and a token tk_x, the time to compute $M(x)$ is polynomial in the runtime of M on the specific input x.

Goldwasser *e*t al. [GKP$^+$13] provide a reusable garbling scheme only for depth bounded circuits; our schemes remove the depth dependency, provide short garbling size, and can additionally avoid worst-case running time.

Theorem 4 (Informal). *There exists a reusable garbling scheme and an input-specific reusable garbling scheme for (uniform or non-uniform) polynomial-time Turing machines from the assumptions in Sec. 1.2.*

In summary, our work models computation on encrypted data as Turing machines and thus avoids the worst-case "curse" for a set of well-known cryptographic notions.

Remark 1. Interestingly, we can easily overcome the worst-case curse for interactive tasks such as two-party and multi-party protocols as follows. To securely evaluate a Turing machine M, we evaluate the Turing machines $M_1, \ldots, M_{\omega(\log n)}$ sequentially, where M_i runs the Turing machine M for 2^i steps and outputs M's answer if M halted in 2^i steps, otherwise $\perp$. To evaluate M_i, we simply use existing multi-party protocols. Note that the circuit size for M_i is $\mathrm{poly}(2^i)$, and since we halt the computation as soon as we get a non-$\perp$ answer, the protocol runs in input-specific time. The reason we can overcome the worst-case curse in this manner is that interaction is allowed. In this work, we focus on non-interactive tasks, which are more challenging.

1.2 Our Assumptions

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

Extractable Witness Encryption. The recent work of Garg *et al.* [GGSW13] constructs a new primitive called witness encryption (WE). Such a scheme is associated with some $\mathbf{NP}$ complete language L. Given an instance x and a message m, any user can encrypt m with respect to x; this is denoted by $\mathsf{Enc}_x(m)$. Given $\mathsf{Enc}_x(m)$ and a valid witness w of x, any user can decrypt x efficiently. On the other hand, if x is not in the language, the scheme provides semantic security.

In our work, we additionally assume that the [GGSW13] scheme is extractable: if an adversary can break semantic security for an instance x, an extractor can extract the witness for x. Such an extractable scheme can be constructed from an extractable version of the [GGSW13] assumption (called extractable DGE No-Exact-Cover assumption) so we strengthen their assumption. While we state our assumption in a decisional form for simplicity, the search version of the assumption suffices for our schemes because we can use hard-core predicates to mask the one bit we care to hide (m).

We validate our assumption in the generic group model: we prove that no polynomial-time adversary can break the assumption in the generic group model where adversaries can only use multilinear map operations as a black-box. We refer the reader to our full paper 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.

We show that, interestingly, extractable witness encryption is highly related to another task that was already well-known in the cryptographic literature: (weakly) obfuscating point-filter functions, 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:

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

It can be shown that 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 obfuscation for a wide range of natural tasks based on [GK05]. See our full paper for more details.

The existence of SNARKs (Succinct Non-interactive Arguments of Knowledge). Bitansky *et al.* [BCCT13] construct SNARKs in a generic way (via a reduction from weaker SNARKs). Their work is based on "knowledge of exponent assumptions", and the existence of collision resistant hash functions.

If we remove SNARKs from our constructions, we still obtain novel schemes over prior work because the sizes of the function keys and of the garbling remain short, linear in the size of the Turing machine. Without SNARKs, though, the loss is that the ciphertext size grows with the running time of the Turing machines.

Our FE, FHE, and reusable garbling schemes 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 Techniques overview

ABE for Turing machines. The main technical challenge in this work is constructing an ABE scheme for Turing machines.

Our construction starts with witness encryption and a signature scheme. The function key for a Turing machine M is simply a signature of M. The master secret and public keys generated during setup 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 can be interpreted as a "proof" that $M(x) = 1$.

Loosely speaking, the security proof proceeds as follows. Suppose there exists a successful adversary $\mathcal{A}$ for our ABE scheme. Then, given $\mathsf{Enc}_{x^*}(b)$, the ABE encryption of a random bit b, and several secret keys $\mathsf{sk}_{M_i} = \sigma_i$ such that $M_i(x) = 0$, $\mathcal{A}$ 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 of 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 unforgeability of the signature scheme.

Unfortunately, this idea does not quite give us the results we want. The reason is that the time to check a witness for an instance $x^* = (x, \mathsf{VK})$ is very long because it involves checking the tableau π of M on input x. In this case, the witness encryption of Garg *et al.* [GGSW13] is not "succinct": the size of the ciphertext $\mathsf{Enc}_{x^*}(b)$ grows with the time to check the witness. Thus, the approach above gives us a non-succinct ABE scheme, where the size of a ciphertext depends on the worst-case runtime of any (allowed) Turing machine.

To obtain succinctness, we use a SNARG scheme [BCCT13]. A SNARG has a common reference string crs, which is assumed to be securely generated. Any user can prove any $\mathbf{NP}$ statement by computing a proof π. The length of the crs, the length of the proofs, and the time to verify a proof are all *short*: depending only on the security parameter, and not on the time to verify the $\mathbf{NP}$ witness.

$\mathsf{Enc}_{x^*}(b)$ now proceeds as follows. It generates a crs corresponding the underlying SNARG scheme. To encrypt a bit b w.r.t. a public attribute x, it simply computes $\mathsf{Enc}_{x^*}(b)$, where x^* is now $(x, \mathsf{crs}, \mathsf{VK})$. 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* SNARG proof that $M(x) = 1$. The fact that π can be verified in a short time makes the WE ciphertext succinct, as desired.

This gives us an ABE for Turing machines. Because SNARKs are for NP, our resulting ABE scheme is for any class of Turing machines for which there exists a polynomial that upper bounds the runtime of all machines in the class.

There scheme still has a slight drawback: it is succinct only for uniform Turing machines. If the Turing machines have non-uniform advice as large as the runtime, the resulting ABE ciphertexts are non-succinct. We would like our ABE scheme to be a generalization of previous work on circuits, and in particular to be succinct for any non-uniform Turing machine. To this end, we replace the SNARG scheme with a SNARK scheme (succinct non-interactive argument of knowledge) scheme. SNARKs have the additional property that if an adversary $\mathcal{A}$ succeeds in proving that $x \in L$, an extractor can extract a corresponding witness w from $\mathcal{A}$.

The final ABE scheme is 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 Sec. 3 for more details on our ABE scheme and the security proof.

Functional encryption for Turing machines. We use the reduction of Goldwasser *et al.* [GKP+13] to construct a (single-key and succinct) FE scheme from FHE and ABE. Their reduction is for circuits so we need to adapt it to Turing machines. The main technical issue is that we need to perform the FHE evaluation of a Turing machine M. To achieve this goal, we construct a new Turing machine M_{FHE} that evaluates homomorphically the transition function of M for a t number of times. The problem is that M_{FHE} needs to know what inputs to read from M's tape to feed into the FHE evaluation, but the movement of the head in M is an output of the transition function, so it is encrypted with FHE and unavailable to M_{FHE}. To solve this issue, we transform M into an *oblivious* Turing machine using Pippenger-Fischer [PF79]: now the movement of the head follows a fixed and known pattern independent of the input to M.

If one allows the runtime of M on x to leak, we can provide a second FE scheme FE^* whose decryption algorithm runs in input-specific time. We construct FE^* as a reduction from our FE scheme above using the idea of [GKP$^+$13]: instead of generating a function key sk_M for a Turing machine M, we generate many function 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 either outputs the output of M or $\perp$ if M did not halt in 2^i steps; the parameter B_n is a global bound on the runtime of the Turing machines we consider. To generate $\log B_n$ function 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, since the underlying functional encryption scheme is for Turing machines, generating sk_{M_i} can be done very efficiently, in time polynomial in the *size* of M_i, independent on the runtime of M_i.

On input a ciphertext $\mathsf{Enc}(x)$ and a function key $(\mathsf{sk}_{M_1}, \ldots, \mathsf{sk}_{M_{\log B}})$ for the Turing machine M, the decryption algorithm 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.

Reusable garbling and a variant of FHE for Turing machines. In our full version, we show how to construct these schemes from our FE scheme using a similar reduction to [GKP$^+$13].

Other related work. We discuss other related work in the full version of our paper.

1.4 Paper Roadmap

The rest of this paper is organized as follows. We provide definitions for extractable witness encryption and ABE in Sec. 2, and refer the reader to our full paper for other relevant preliminaries. Next, Sec. 3 presents our ABE scheme for Turing machines, which we prove formally in our full paper. Finally, Sections 4 and 4.2 show how to construct functional encryption for Turing machines. Due to space constraints, in our full paper, we present the construction of extractable witness encryption and prove the new assumption in the generic group model, we show that extractable witness encryption implies (weakly) obfuscatable point filter functions and deduce implications to obfuscation, and we present the construction of FHE for Turing machines.

2 Preliminaries

In this section, we define extractable witness encryption and ABE for Turing machines, and refer the reader to our full paper for definitions of FE for Turing machines, SNARKs, and other relevant preliminaries.

2.1 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 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, $f(\kappa) = \text{negl}(\kappa)$.

2.2 Witness encryption (WE)

The syntax of WE is as defined by Garg et al. [GGSW13], but the security definition has an additional extractability property.

Definition 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 $w \in \{0,1\}^*$ and a ciphertext* ct *and outputs a bit b or the symbol $\perp$.*

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

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

Definition 2 (Extractable security). *A witness encryption scheme for a language $L \in NP$ is secure if for all p.p.t. adversaries A, and all poly q, there exists a p.p.t. extractor E and a poly p, such that for all auxiliary inputs z and for all $x \in \{0,1\}^*$, the following holds:*

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

2.3 Attribute-based encryption (ABE) for Turing machines

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

Definition 3 (ABE for Turing machines). *An attribute-based encryption scheme* ABE *for a class of Turing machines $\mathcal{T}$ is a tuple of four algorithms* ($\mathsf{ABE.Setup}$, $\mathsf{ABE.KeyGen}$, $\mathsf{ABE.Enc}$, $\mathsf{ABE.Dec}$)*, 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 function 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,*

$$\begin{aligned}&\Pr[(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\kappa); \mathsf{fsk}_f \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}, f);\\ &\quad c \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}, x) : \mathsf{ABE.Dec}(\mathsf{fsk}_f, 1^t, c) = f(x)]\\ &\quad = 1 - \mathrm{negl}(\kappa).\end{aligned}$$

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).

Our security definition is full (the adversary can choose the challenge attribute based on the public key) and non-adaptive (the adversary chooses the Turing machines before getting the challenge ciphertext).

Definition 4 (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.*

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

```
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*:*

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

3 Attribute-based Encryption for Turing Machines and RAMs

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

1. an extractable witness encryption scheme WE = (WE.Enc, WE.Dec) based on the work of [GGSW13], on which we elaborate in Sec. 2.2,

2. a succinct argument of knowledge scheme, $\mathsf{SNARK} = (\mathsf{SNARK.Gen}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$, based on the work of [BCCT13],
3. an existentially unforgeable signature scheme secure against adaptive chosen message attacks $\mathsf{SIG} = (\mathsf{SIG.KeyGen}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ [GMR88].

Theorem 5. *Assuming the above three primitives, there exists a secure attribute-based encryption scheme (as per Def. 4) for any class of (uniform or non-uniform) Turing machines $\mathcal{T}$, for which there exists a polynomial p such that the runtime of every machine in $\mathcal{T}$ is upper-bounded by p.*

The p restriction comes from the fact that SNARKs are for $\mathbf{NP}$. From now on, for brevity, we will refer to such a class by "a class of Turing machines with runtime upper-bounded by some polynomial".

Corollary 1. *There exists a secure attribute-based encryption scheme for any class of (uniform or non-uniform) Turing machines whose runtime is upper-bounded by some polynomial under the extractable DGE No-Exact-Cover assumption, "knowledge of exponent assumption", and the existence of collision-resistant hash functions (Sec. 1.2).*

3.1 Construction preliminaries

We advise the reader to recall the intuition we provided in technique overview, Sec. 1.3.

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. Moreover, $t < p(|x|)$, where p is a polynomial upper-bound on the runtime of every Turing machine in the class of interest. Let $(\mathsf{SNARK.Gen}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$ be a SNARK system for L.

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

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

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

3.2 Construction of ABE for Turing machines

Our construction of $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ for Turing machines proceeds as follows. 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 $\mathsf{SNARK.Gen}$ 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 $\mathsf{SNARK.Prover}$ 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. We prove Th. 5 formally in our full version, and we only provide intuition here for 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; but A_{ABE} only asked for signatures of Turing machines M_i for which $M_i(x) = 0$. Therefore, (M, σ) are a new signature pair and thus we used 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 $\mathsf{ABE.KeyGen}$. See our full paper for more details. Let (M, D) be a RAM pair: a RAM machine M and 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 a machine, $\sigma_{(M,r)}$ is a signature on the description of the machine M and the nonce r, S is a set of integers that represent memory addresses (the memory accesses M makes to D), 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 function key size are polynomial in the description of the RAM and the size of $|D|$, but they do not depend on the runtime of the RAM. (As a remark, to obtain a slightly shorter key size, one can sign a Merkle tree over the entries in D.) 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.

3.4 Beyond ABE for Turing machines and RAMs

Interestingly, it turns out the expressivity of our ABE construction goes beyond that of Turing machines and RAMs. The ABE construction can be easily changed to allow the evaluator to provide *an additional input* α to the computation. That is, given a function key sk_M, a ciphertext $\mathsf{ct}_{x,m}$, an evaluator can choose an input α by himself; then if $M(x, \alpha) = 1$, $\mathsf{ABE.Dec}$ outputs m, otherwise, it outputs $\perp$. To construct such an ABE, one only has to change the SNARK language L such that an instance has the form (VK, x, t) and a witness is (M, σ, α) with $M(x, \alpha) = 1$ and σ verifies M.

This extra input α makes the scheme significantly more expressive. We illustrate on two examples. The first example allows the secret key owner to delegate the choice of Turing machines to another user, say Alice, by issuing a function key for Alice; then Alice can choose Turing machines of her choice to run on the ciphertexts, without contacting the secret key owner. To construct this example, the secret key owner generates $\mathsf{sk}_{U_{\mathsf{Alice}}}$ where U_{Alice} is a universal circuit containing Alice's public key. U_{Alice} takes as input $\alpha = (\mathsf{TM}, \sigma(\mathsf{TM}))$ and x: it first checks that $\sigma(\mathsf{TM})$ verifies with Alice's public key as being a signature of TM, and if so, it runs $\mathsf{TM}(x)$. Now Alice can choose any Turing machine TM she wishes, and as long as she signs it, she will be able to evaluate it on the ciphertext. In fact, the secret key owner can delegate the choice of Turing machines to any group of people, and he can even express complex policies, e.g. "allow any Turing machine that is signed by (Alice and Bob) or Chris".

The second example is to run any approved RAM on any approved database, where approved means that it was signed by the secret key owner. We do not elaborate further on this construction and its applications in this short paper version.

4 Functional encryption for Turing machines

In this section we construct a (single-key and succinct) functional encryption scheme for Turing machines. We refer the reader to our full paper for a definition of FE for Turing machines.

Theorem 6. *Assuming we have:*

- *an attribute-based encryption scheme for any class of (uniform or non-uniform) Turing machines with running time upper-bounded by a polynomial, and*
- *a fully homomorphic encryption scheme,*

there is a (single-key and succinct) functional encryption scheme for any class of (uniform or non-uniform) Turing machines with running time upper-bounded by a polynomial.

Theorem 7. *Assuming there exists a (single-key and succinct) functional encryption scheme for any class of (uniform or non-uniform) Turing machines with running time bounded by a polynomial, there is a (single-key and succinct) input-specific runtime functional encryption scheme for any class of (uniform or non-uniform) Turing machines with running time bounded by a polynomial.*

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

4.1 FE for Turing machines construction (FE)

Recall the construction overview provided in Sec. 1.3. We follow the reduction of Goldwasser *et al.* [GKP+13] who showed how to construct a (single-key and succinct) functional encryption scheme from any ABE and FHE scheme, where functions were modeled as circuits.

Our construction of $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ proceeds similarly to the [GKP+13] construction, with the main difference being that we work with Turing machines instead of circuits. There are two places in the reduction where the treatment of circuits is different from the treatment of Turing machines: in the use of the ABE and FHE schemes. To adapt the reduction to Turing machines, we first use our ABE for Turing machines scheme. Second, we need to construct a Turing machine M_{FHE} that performs the FHE evaluation of another Turing machine M. We only present here the construction of M_{FHE} and delegate the full FE construction to our full paper.

Based on the intuition provided in Sec. 1.3, 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 M_{FHE} 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. Use the Pippenger-Fischer transformation [PF79] for time bound t to transform M into an oblivious Turing machine M_O with head movement function next. next is a function that takes as input i, the current step in the computation, and outputs whether the head of M_O should move left or right on the tape. The Turing machine M_O has a transition function δ: δ takes as input a tape input bit b, a state state and outputs a new state state', and the new content b' for the new tape location which is indicated by next.
2. Based on (M_O, next), construct a new Turing machine M_{FHE} that takes as input an FHE public key hpk and an input encryption $\hat{x}$. M_{FHE} evaluates homomorphically the transition function δ of M_O for t steps. Each cell of the tape of M_O corresponds to the FHE encryption of the cell value for M_{FHE}. At step i, M_{FHE} maintains the FHE encryption of the state of M_O at time i: $\widehat{\mathsf{state}_i}$. At step i, M_{FHE} takes as input the encrypted bit from the input tape $\hat{b}$ that the head currently points at, the current encrypted state $\widehat{\mathsf{state}_i}$, and outputs an encrypted new state $\widehat{\mathsf{state}_{i+1}}$ and a new content $\widehat{b'}$. M_{FHE} updates the current cell with $\widehat{b'}$ and then computes $\mathsf{next}(i)$ to determine whether to move left or right.
3. Output the description of M_{FHE}.

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

4.2 Input-specific runtime functional encryption for Turing machines (FE^*)

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 a function key for M denoted 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 Sec. 1.3 for the construction overview and to our full paper for the definition of input-specific runtime functional encryption.

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.[3]
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 $i = 1 \ldots \tau$.

[3] 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.

2. Output $\mathsf{ct} := (\mathsf{ct}_1, \ldots, \mathsf{ct}_\tau)$.

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

```
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. 7 in our full paper.

Acknowledgments

We would like to thank the authors of the witness encryption paper [GGH13a] and Zvika Brakerski for useful discussions. This work was supported by an NSERC Discovery Grant, by DARPA awards FA8750-11-2-0225 and N66001-10-2-4089, by NSF awards CNS-1053143 and IIS-1065219, and by Google. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

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/.

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

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. Foundations of garbled circuits. In *ACM CCS*, pages 784–796, 2012.

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.

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

GGH13a. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. In *EUROCRYPT*, pages 1–17, 2013.

GGH$^+$13b. Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps, 2013.

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

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

GKP$^+$13. Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

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.

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.

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

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

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

LO12. Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In *Eurocrypt*, pages 719–734, 2012.

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.

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

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

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

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

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