An extended abstract [DGM23a] of this article appears in the proceedings of PKC 2023. The full version [DGM23b] of this article is available in the IACR Cryptology ePrint Archive.

Laconic Function Evaluation for Turing Machines

Nico Döttling[1] *, Phillip Gajland[2,3] **, and Giulio Malavolta[2] ***

[1] CISPA Helmoltz Center for Information Security
doettling@cispa.de
[2] Max Planck Institute for Security and Privacy
{phillip.gajland,giulio.malavolta}@mpi-sp.org
[3] Ruhr-University Bochum

Abstract Laconic function evaluation (LFE) allows Alice to compress a large circuit $\mathbf{C}$ into a small digest d. Given Alice's digest, Bob can encrypt some input x under d in a way that enables Alice to recover $\mathbf{C}(x)$, without learning anything beyond that. The scheme is said to be *laconic* if the size of d, the runtime of the encryption algorithm, and the size of the ciphertext are all sublinear in the size of $\mathbf{C}$.

Until now, all known LFE constructions have ciphertexts whose size depends on the *depth* of the circuit $\mathbf{C}$, akin to the limitation of *levelled* homomorphic encryption. In this work we close this gap and present the first LFE scheme (for Turing machines) with asymptotically optimal parameters. Our scheme assumes the existence of indistinguishability obfuscation and somewhere statistically binding hash functions. As further contributions, we show how our scheme enables a wide range of new applications, including two previously unknown constructions:

- Non-interactive zero-knowledge (NIZK) proofs with optimal prover complexity.
- Witness encryption and attribute-based encryption (ABE) for Turing machines from falsifiable assumptions.

1 Introduction

Laconic function evaluation (LFE) is a cryptographic primitive recently introduced by Quach, Wee, and Wichs [FOCS'18]. Using LFE, Alice can compress a large circuit $\mathbf{C}$ into a small digest d. Given Alice's digest, Bob can encrypt some input x under d in a way that enables Alice to recover $\mathbf{C}(x)$ without learning anything about Bob's input. The scheme is said to be *laconic* if the size of the digest d, the runtime of the encryption algorithm $\mathsf{LFE.Enc}$, and the size of the ciphertext c are all sublinear in the size of $\mathbf{C}$.

LFE is particularly interesting in the context of two-party and multi-party computation (2PC, MPC), since it enables the construction of protocols with novel properties. As an example, LFE enables a "Bob-optimised" two-round 2PC protocol in which Alice does all the work, while Bob's computation and communication are smaller than both the function being evaluated and Alice's input. However, for all known LFE constructions [QWW18, AR21, NRS21], the runtime of the encryption procedure and the size of Bob's ciphertext depend on the *depth* of the circuit being evaluated by Alice. This is a severe limitation which restricts the applicability of this primitive to "shallow" circuits. In some sense, this mirrors the efficiency gap between *levelled* and *fully* homomorphic encryption. This leaves us with the following open problem (also stated in [QWW18]):

Is it possible to construct LFE where Bob's work is independent of the circuit size?

* Funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

** Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and the European Union (ERC AdG REWORC - 101054911).

*** Funded by the German Federal Ministry of Education and Research BMBF (grant 16K15K042, project 6GEM) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038.

1.1 Our Results

We answer this question in the affirmative and our main result is the construction of an asymptotically optimal LFE scheme assuming indistinguishability obfuscation [BGI$^+$01] and somewhere statistically binding (SSB) hash functions [HW15]. Our construction enables the computation of any Turing machine M and, unlike all prior constructions [QWW18] [AR21] [NRS21], removes the dependency on the depth of the circuit (the runtime of the Turing machine in our case). In the standard simulation-security setting, we obtain the following result.

Theorem 1 (Informal). *Assuming indistinguishability obfuscation for circuits and somewhere statistically binding hash functions, there exists a simulation secure LFE scheme with the following parameters:*

– *The size of the digest* d *is* $\mathsf{poly}(\lambda)$.
– *The runtime of the encryption procedure is* $\mathcal{O}(|x| + |\mathsf{M}(x)|) \cdot \mathsf{poly}(\lambda)$.
– *The size of the ciphertext* c *is* $\mathcal{O}(|x| + |\mathsf{M}(x)|) \cdot \mathsf{poly}(\lambda)$.

If we relax the security to an indistinguishability-based notion, we can further improve the parameters by removing the dependency on the size of the output.

Theorem 2 (Informal). *Assuming indistinguishability obfuscation for circuits and somewhere statistically binding hash functions, there exists a LFE scheme with ciphertext indistinguishability and the following parameters:*

– *The size of the digest* d *is* $\mathsf{poly}(\lambda)$.
– *The runtime of the encryption procedure is* $\mathcal{O}(|x|) \cdot \mathsf{poly}(\lambda)$.
– *The size of the ciphertext* c *is* $\mathcal{O}(|x|) \cdot \mathsf{poly}(\lambda)$.

As for the underlying assumptions, SSB hash functions [OPWW15] can be constructed from a variety of standard assumptions (e.g. LWE or DDH), whereas indistinguishability obfuscation is a less understood primitive and currently the subject of a large body of research. Numerous recent works [BDGM20, GP20, JLS20, WW21, JLS22, BDGM22] show provably-secure constructions of indistinguishability obfuscation for circuits under simple assumptions, some of which are regarded as well-founded.

We briefly describe some additional implications which show how our construction enables a wide range of new results in cryptography.

(1) Witness Encryption for Turing Machines: We construct the first witness encryption where the size of the ciphertext depends only on the size of the witness and the security parameter (but not on the NP relation $\mathcal{R}$). Furthermore, the decryption runtime is only proportional to the runtime of the Turing machine computing $\mathcal{R}$, rather than its circuit representation. This implies the first ABE for Turing machines [GKP$^+$13] from falsifiable assumptions. Prior to our work, Goldwasser et al. [GKP$^+$13] constructed the same primitive from *extractable* witness encryption,[4] which is a considerably stronger and non-falsifiable assumption, whose validity has often been called into question [GGHW14, BP15, BSW16].

(2) NIZKs with Optimal Prover Complexity: By applying a known transformation [KNYY19], we construct the first *prover-optimal* NIZK proof system, where the prover's computational complexity depends only on the size of the witness and on the security parameter (and is otherwise independent of the size of the NP relation).

(3) MPC Compiler: By applying the transformation described in [QWW18] we obtain a compiler for multi-party computation (MPC) that reduces the communication complexity to be independent of the circuit size, *without introducing additional rounds of interaction.*

[4] We should also mention a recent work of Ananth et al. [AFS19], which constructs ABE for RAM programs from LWE, although it achieves only a weaker form of efficiency where the public parameters and the ciphertexts grow with the runtime of the RAM program.

1.2 Technical Overview

Following is a brief overview of the techniques developed in this work. Before delving into our approach, we briefly discuss why trivial solutions fall short in constructing LFE.

Why Trivial Solutions Fail. An astute reader may wonder why this is still a challenging problem, given iO for circuits. One plausible approach to constructing LFE via this route would be to place the hash of the circuit $\mathsf{d} := H(\mathbf{C})$ in the common reference string. Bob could then obfuscate and send Alice the following universal circuit

$$\mathcal{U}(\mathbf{C}') : \textbf{if } \mathsf{d} \stackrel{?}{=} H(\mathbf{C}') \ \textbf{ return } \mathbf{C}'(x).$$

Intuitively, Alice should only be able to run the obfuscated circuit on $\mathbf{C}$ unless she is able to find a collision for H. Unfortunately, this approach has two major flaws:

(1) Efficiency: The construction is *not laconic* since both the runtime of Bob and the size of the ciphertext depend on the size of $\mathbf{C}$. Even recent constructions of iO for Turing machines [AJS17] suffer from the drawback that the size of the obfuscated Turing machine depends on the maximum input size. An exception is the recent work of [BFK+19] which, however, requires a large shared random string or a random oracle. At present, constructing iO without input-size dependence remains an open problem.

(2) Provable Security: The above informal argument assumes the strong notion of virtual-blackbox obfuscation, which is known to be impossible [BGI+01]. Constructing a provably secure scheme requires a significant modification of the template in order to be able to leverage the weak *indistinguishability* security of iO.

Even if iO for Turing machines does not appear to be sufficient to construct LFE, it turns out that other techniques from the area [KLW15, CCHR15, CH16, CCC+16, ACC+16, GS18] will help us in building a provably-secure scheme, as we explain in the following.

Our Approach. Our construction builds on the techniques introduced in [GS18], and requires us to modify the construction in a non-blackbox manner, in order to constrain Alice to execute the Turing machine M on Bob's input while at the same time making Bob's runtime independent of it. To gain some intuition on the approach, we consider the simplified setting in which both parties know a public Turing machine M, where the transition function is denoted by $\mathbf{C}_{\mathsf{M}}$ and Bob holds an input x. Later in this overview, we show that this template can be lifted to the more generic setting where Alice evaluates a *private* Turing machine by letting M be a universal Turing machine with an additional input. To establish some notation, consider the insecure protocol where Bob sends his input x in plain: Alice can evaluate M by maintaining a database D that encodes x and the current state of the memory of M. Each operation of $\mathbf{C}_{\mathsf{M}}$ consists of reading the current state, one bit from Alice, and one from Bob.

Garbled Circuits. One possible way to secure this approach is to use Yao's garbled circuits [Yao82, Yao86], that allow for the secure computation of a circuit $\mathbf{C}$ by creating a *garbled* version $\widetilde{\mathbf{C}}$ and encoding the input $x = (x_1, \ldots, x_n)$ as a set of labels $(\mathsf{lbs}_1, \ldots, \mathsf{lbs}_n)$. Security is guaranteed as long as a *single* input encoding is revealed to the evaluator. If we were to garble the step circuit $\mathbf{C}_{\mathsf{M}}$, we immediately run into two problems: (1) From an efficiency perspective, Bob would need to garble one circuit for each step of the computation, which would be more expensive than just evaluating M locally. (2) With regards to functionality, the evaluator needs to receive the labels corresponding to an input encoding. This corresponds to a particular set of locations in D (depending on which bits $\mathbf{C}_{\mathsf{M}}$ needs to read). The difficulty here stems from the fact that the state of D evolves over the course of the computation, as it includes the memory tape of the Turing machines. Thus, we would need a way to *dynamically* select labels depending on the intermediate state of D. Fortunately, (1) can be solved using iO: Instead of garbling all step circuits explicitly, Bob sends an obfuscated circuit that, given an index i, returns the i^{th} garbled step circuit. The remainder of this overview is devoted to solving the second challenge (2).

Updatable Laconic Oblivious Transfer. Before explaining our solution, we recall the notion of updatable laconic oblivious transfer (ULOT) [CDG+17]. With an ULOT protocol, a large database D can be hashed to a small digest d offering the sender two operations.

Read: Given a pair of messages (m_0, m_1) and an index i, the sender can compute a ciphertext c such that the receiver (knowing D and d) can recover $m_{D[i]}$, where $D[i]$ is the value of the bit at the ith location of D.

Write: Given $|\mathsf{d}|$-many pairs of messages $\{m_{0,i}, m_{1,i}\}_{i\in[|\mathsf{d}|]}$, a bit b, and index i, the sender can compute a ciphertext c such that the receiver (knowing D and d) can recover $\left(m_{D'_1,1}, \ldots, m_{D'_{|\mathsf{d}|},|\mathsf{d}|}\right)$. Here, d' is the hash of D', the database D updated by writing b at index i.

Equipped with this functionality, we can now devise a mechanism to provide the evaluator with the appropriate input encodings. Bob compresses his input x, using the hashing procedure of the ULOT scheme and sends it to Alice, who will act as the evaluator. At each step of the computation, Alice is provided with the labels corresponding to the database locations needed by the current step circuit. She then uses these labels to evaluate the garbled step circuit, which performs the computation step and computes a ULOT ciphertext containing the pairs of labels for the next step of the computation. In the next step, Alice will be able to retrieve the set corresponding to the locations of the updated database, by running the receive algorithm of the ULOT. These include an encoding of the updated hash of D, as a result of the write operation of the step circuit.

Piecing it Together. Now only two problems remain. First, the state of D is given in clear to Alice, meaning the intermediate values of the computation are leaked. This is solved by adding a layer of symmetric encryption to the memory of the Turing machine. To ensure the correctness of the computation, we remove this layer before feeding the input into $\mathbf{C}_{\mathsf{M}}$. The output is then re-encrypted using a new key that is only available in the next step circuit. As this happens within the garbled circuit, security is preserved. We can now lift the construction to the setting where Alice's M is not known to Bob. This is done by including an additional ULOT digest of the description of the Turing machine, which allows the step circuit to read the description (via ULOT read) and determines the next operation of the computation. Given the above procedure, the database lookup algorithm can be naturally extended to the case of an additional tape, encoding the machine's instructions. To ensure that the random coins used in the garbled circuits are *consistent* across different computations steps, we use a (puncturable) PRF to sample the labels.

The Final Scheme. We provide some intuition for the encryption and decryption procedures in [Fig. 1]. For the encryption procedure, Bob starts by obfuscating the Garbling Step Circuit and computing the first set of labels that will be needed to evaluate the garbled circuit. These are then sent along with his encrypted input to Alice. For the decryption procedure, Alice evaluates the garbled circuit using the first set of labels sent by Bob. The output from the Step Circuit is then used for receiving the updatable laconic oblivious transfer. This is repeated for all steps of the computation until the final output is returned by the decryption procedure.

Security Proof. Next, we provide some intuition about the security argument. To prove the security of our construction we use a similar proof strategy to that of [GS18]. In particular, our proof proceeds via a hybrid argument. In each hybrid we change the way the obfuscated circuit computes the garbled circuits for each step of the computation. Each garbled step circuit can be computed in three modes. The first mode is real, where the computations are just as in the real protocol. The second mode is dummy, where the output of the garbled circuit is constant and hardwired, but the same as in the real execution. The third mode is sim, which is similar to real mode, with the difference being that the garbled circuit only outputs dummy values which are not the same as in the real execution. We cannot change directly from real mode to sim mode because at each step of the computation the labels from the previous step are visible to the adversary. Hence, we first need to change to dummy mode and then to sim mode. We show a set of rules that define a pebbling game, where the pebbles are represented by simulation slots. The aim of the game is to switch the pebbles from real (white pebbles) to sim (black pebbles), while minimizing the number of nodes in dummy (grey pebbles). Our objective is to minimize the number of grey pebbles at any point in time because the size of the obfuscated circuit grows with the number of simulation slots in dummy mode. Finally, with help of a pebbling strategy [GS18], we prove that our LFE construction is secure while having only a poly-logarithmic number of grey pebbles at any point in the simulation.

$\mathsf{LFE.Enc}(\mathsf{d}, x)$

Block-wise encrypt x
Obfuscate Garbling Step Circuit **GarbleSC**

$\mathsf{iO}(\mathbf{GarbleSC})$:

> Garbling Step Cicruit
>
> Computes labels for current and next Step Circuit
> Computes Step Circuit **SC**
>
> **SC** :
>
> > Step Circuit
> >
> > Decrypts secret inputs
> > Performs one step of the computation
> > Encrypts secret outputs
> > Writes outputs to D using $\mathsf{ULOT.Write}$
> > Sends next operation using $\mathsf{ULOT.Read}$

Compute initial labels for **GarbleSC**
return encrypted x, $\mathsf{iO}(\mathbf{GarbleSC})$, initial labels

$\mathsf{LFE.Dec}(\mathsf{d}, \mathsf{c})$

Compute **GarbleSC** at 1 using initial inputs
while M is not done
  Compute **GarbleSC** at i

  $\mathsf{iO}(\mathbf{GarbleSC})(i)$:

> Garbling Step Cicruit
>
> Compute labels for current and next Step Circuit
> Compute Step Circuit **SC** at i
>
> $\mathbf{SC}_i$:
>
> > Step Circuit
> >
> > Decrypt secret inputs
> > Perform one step of the computation
> > Encrypt secret outputs
> > Write outputs to D using $\mathsf{ULOT.Write}$
> > Send next operation using $\mathsf{ULOT.Read}$

  Run $\mathsf{ULOT.Receive}$ to obtain labels used as inputs for $\mathsf{iO}(\mathbf{GarbleSC})(i+1)$
return final output from Step Circuit

Figure 1. High level overview of the encryption and decryption procedures.

Application: Witness Encryption for Turing Machines. We show how our newly constructed LFE scheme allows us to construct witness encryption for Turing machines. To encrypt a message m with respect to a relation $\mathcal{R}$, the witness encryption algorithm computes the crs of the LFE and hashes $\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathbf{M}_{\mathcal{R}})$, where the Turing machine is defined as

$$\mathsf{M}_{\mathcal{R}}(m, w) \coloneqq \begin{cases} \textbf{return } m & \textbf{if } \mathcal{R}(x, w) = 1 \\ \textbf{return } \perp & \textbf{else} \end{cases}.$$

Then it returns the obfuscation of a circuit $\mathsf{ob}\mathbf{C} \leftarrow \mathsf{iO}(\mathbf{C}_{x,m})$ where $\mathbf{C}_{x,m}$ is defined as

$$\mathbf{C}_{x,m}(w) \coloneqq \textbf{return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m, w)).$$

Given a witness w, one can recover m by querying the obfuscated circuit and evaluating the LFE decryption algorithm:

$$\begin{aligned} \mathsf{LFE.Dec}(\mathsf{crs}, \mathbf{M}_{\mathcal{R}}, \mathsf{ob}\mathbf{C}(w)) &= \mathsf{LFE.Dec}(\mathsf{crs}, \mathbf{M}_{\mathcal{R}}, \mathbf{C}_{x,m}(w)) \\ &= \mathsf{LFE.Dec}(\mathsf{crs}, \mathbf{M}_{\mathcal{R}}, \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m, w))) \\ &= \mathbf{M}_{\mathcal{R}}(m, w) \\ &= m. \end{aligned}$$

Note that the size of the ciphertext is only dependent on the size of the witness w, the size of the message m, and the security parameter. Furthermore, the runtime of the decryption algorithm only depends on the runtime of the Turing machine computing $\mathbf{M}_{\mathcal{R}}$. Security follows via a standard puncturing argument.

Application: ABE for Turing Machines. We also sketch how to turn the above witness encryption into an ABE for Turing machines. This is a standard transformation [GGSW13] and therefore we only include an outline of the construction. To delegate a decryption key for a Turing machine M, the authority computes a signature σ on the tuple $(\mathsf{crs}, \mathsf{d}_{\mathsf{M}})$, where $\mathsf{d}_{\mathsf{M}} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \tilde{\mathsf{M}})$ and $\tilde{\mathsf{M}}(x, m)$ returns m if and only if $\mathsf{M}(x) = 1$. Then encrypting a message m with respect to an attribute x can be done by obfuscating

$$\mathbf{C}_{x,m}(\mathsf{crs}, \mathsf{d}, \sigma, x) : \textbf{if } \mathsf{Verify}(\sigma, (\mathsf{crs}, \mathsf{d})) = 1; \textbf{return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m, x)).$$

Note that the runtime of the encryption algorithm (and consequently the size of the ciphertext) only depends on the size of the attribute x and the message m. Furthermore, the runtime of the decryption algorithm is only proportional to the runtime of the Turing machine M. We refer the reader to Appendix B for a more detailed discussion regarding applications and new implications.

1.3 Related Works

The notion of LFE was introduced in the work of Quach et al. [QWW18], in which they presented a construction for depth-bounded polynomial-size circuits from the learning with errors problem. Work by Pang, Chen, Fan, and Tang [PCFT20] extended the notion of (single-input) LFE to the multi-input settings, by additionally assuming the existence of indistinguishability obfuscation. Their protocol uses single-input LFE (and in particular the scheme from [QWW18]) generically. Thus, our scheme can be plugged into their work to obtain improved parameters. Recent work by Agrawal and Roşie [AR21] shows a new construction of LFE with adaptive security (based on the ring learning with errors assumption). However, the scheme is limited to the computation of NC^1 circuits. Another recent work by Naccache, Roşie, and Spignoli [NRS21] improves the concrete efficiency of LFE. In particular, the authors present a construction based on the LWE assumption with asymptotically smaller parameters than those used in [QWW18]. However, their construction is restricted to the class L/poly, i.e., the class of circuits that can be represented by branching programs of polynomial length.

A notion related to LFE is that of succinct randomized encodings (SRE) [BGL+15]: SRE allows one to encode an input x with respect to a public Turing machine M in such a way that nothing is revealed beyond $\mathsf{M}(x)$. The crucial difference is that the *runtime of the encoding algorithm and the size of the encoding* depend on the size of M, whereas in LFE Bob's ciphertext only depends on the size of his input (and the security parameter). Furthermore, SRE do not allow Alice to privately hash her circuit/Turing machine. In this sense, LFE can be though of as a stronger primitive than SRE.

Differences to [GS18]. Although our approach is intimately related to the work of [GS18], we explicitly mention that one cannot use their result "off the shelf" to build LFE for Turing machines, for similar reasons why the generic approach using iO for Turing machines does not work. Conceptually, one can think of our construction as a "double tape" variant of [GS18] where the obfuscated universal Turing machine has a tape in the clear (that encodes Bob's input) and the hash (which plays the role of a succinct commitment) of a tape encoding Alice's Turing machine. During the evaluation phase, Alice can run her Turing machine, provided that she supplies a valid opening for each chunk of the tape fed into the obfuscated Turing machine.

2 Definitions

Let $\lambda \in \mathbb{N}$ denote the security parameter. We say that a function $\mathsf{negl}(\cdot)$ is negligible if it vanishes faster than the inverse of any polynomial. Given a set S, we denote by $s \leftarrow\$ S$ the uniform sampling from S. We say that an algorithm is PPT if it can be implemented by a probabilistic Turing machine running in time $\mathsf{poly}(\lambda)$. Let X and Y denote two random variables and let $\{X\}_{\lambda \in \mathbb{N}}$ and $\{Y\}_{\lambda \in \mathbb{N}}$ be two distribution ensembles. We say that these distributions are computationally indistinguishable if for all PPT algorithms $\mathcal{A}$, $|\Pr_{x \leftarrow X_\lambda}[\mathcal{A}(x) = 1] - \Pr_{x \leftarrow Y_\lambda}[\mathcal{A}(x) = 1]| \leq \mathsf{negl}(\lambda)$. We denote this by $X_\lambda \overset{c}{\approx} Y_\lambda$. Let G_{par} denote a game, defined relative to a set of parameters par, where an adversary $\mathcal{A}$ interacts with a challenger that answers oracle queries issued by $\mathcal{A}$. We denote the output of the game G_{par}, between a challenger and an adversary $\mathcal{A}$, as $\mathsf{G}^{\mathcal{A}}_{par}$. $\mathcal{A}$ is said to *win* the game if $\mathsf{G}^{\mathcal{A}}_{par} = 1$. We define the advantage of $\mathcal{A}$ in G_{par} as $\mathsf{Adv}^{\mathsf{G}}_{par,\mathcal{A}} := \Pr[\mathsf{G}^{\mathcal{A}}_{par} = 1]$.

2.1 Laconic Function Evaluation for Turing Machines

Here, we adapt the definition of laconic function evaluation (LFE), a primitive recently introduced by Quach, Wichs, and Wee [QWW18], to that of LFE for Turing machines. The runtime of the Turing machine, denoted T, is publicly known and available to all parties. Without loss of generality we assume the Turing machine to be oblivious.

Definition 1 (Laconic Function Evaluation for Turing Machines). *A laconic function evaluation scheme* $\mathsf{LFE} := (\mathsf{LFE.Gen}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ *for Turing machines is defined as the following tuple of* PPT *algorithms.*

$\mathsf{crs} \leftarrow \mathsf{LFE.Gen}\left(1^\lambda, 1^N\right)$**:** *Given the security parameter* 1^λ *and the block size* 1^N *(encoded in unary), the generation algorithm returns a common reference string* crs.

$\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathsf{M})$**:** *Given the common reference string* crs *and the description of a Turing machine* M, *the compression algorithm returns a digest* d.

$\mathsf{c} \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, x)$**:** *Given the common reference string* crs, *a digest* d, *and a message* x, *the encoding algorithm returns a ciphertext* c.

$y \leftarrow \mathsf{LFE.Dec}(\mathsf{crs}, \mathsf{M}, \mathsf{c})$**:** *Given the common reference string* crs, *the description of a Turing machine* M, *and a ciphertext* c, *the decoding algorithm returns a message* y.

For correctness, we require the encoding of an input with respect to the digest of a Turing machine, when decoded, to return the same result as evaluating the machine on the input. A more formal definition follows.

Definition 2 (Correctness). *A laconic function evaluation scheme* $\mathsf{LFE} := (\mathsf{LFE.Gen}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ *for Turing machines is correct if for all* $\lambda \in \mathbb{N}$, $N \in \mathbb{N}$, *for all Turing machines* M, *and all messages* x *it holds that*

$$\Pr\left[\mathsf{M}(x) = y \left| \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{LFE.Gen}(1^\lambda, 1^N) \\ \mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathsf{M}) \\ \mathsf{c} \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, x) \\ y \leftarrow \mathsf{LFE.Dec}(\mathsf{crs}, \mathsf{M}, \mathsf{c}) \end{array}\right.\right] = 1,$$

where the probability is taken over the random coins of $\mathsf{LFE.Gen}$ *and* $\mathsf{LFE.Enc}$.

The security notion captures the requirement that the encryption of a message x with respect to a compressed Turing machine M reveals nothing beyond $\mathsf{M}(x)$.

Definition 3 (Security: Sender-Privacy Against Semi-Honest Receivers). *A laconic function evaluation scheme* $\mathsf{LFE} := (\mathsf{LFE.Gen}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ *for Turing machines is secure if there exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{LFE}}$ *such that for any stateful* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and* $N \in \mathbb{N}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\left| \begin{array}{l} \Pr\left[\mathcal{A}_2(c, \mathsf{st}) = 1 \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{LFE.Gen}(1^\lambda, 1^N) \\ (x, \mathsf{M}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ \mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathsf{M}) \\ c \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, x) \end{array} \right] \\ - \Pr\left[\mathcal{A}_2(c, \mathsf{st}) = 1 \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{LFE.Gen}(1^\lambda, 1^N) \\ (x, \mathsf{M}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ \mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathsf{M}) \\ c \leftarrow \mathsf{Sim}_{\mathsf{LFE}}(\mathsf{crs}, \mathsf{d}, \mathsf{M}, \mathsf{M}(x), T) \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of $\mathsf{LFE.Gen}$*,* $\mathcal{A}_1$*,* $\mathsf{LFE.Enc}$ *and* $\mathsf{Sim}_{\mathsf{LFE}}$*. Here,* T *denotes the runtime of* $\mathsf{M}(x)$ *and* st *the state of* $\mathcal{A}$*.*

An additional security property of an LFE scheme is that of *function hiding*, which captures the notion that the digest $\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathsf{M})$ should hide the description of the Turing machine M. We note that our scheme can be generically transformed to satisfy function-hiding using the transformation of [QWW18]. The transformation uses 2-round 2PC based on OT and garbled circuits, and maintains the same asymptotic efficiency.

3 Laconic Function Evaluation for Turing Machines

In this section we will construct a laconic function evaluation scheme [Fig. 4] with asymptotically optimal parameters.

Notation. We consider the case where the protocol computes a function $F(m_A, m_B)$, where m_A and m_B are the inputs of Alice and Bob, respectively. We assume that the function $F(m_A, m_B)$ is computed by a Turing machine M, where m_A and m_B are given to M on two different input tapes. We assume without loss of generality that the Turing machine M is publicly known.[5] More formally, M denotes the 4-tape Turing machine consisting of two read-only input tapes, a read/write work tape, and a read/write output tape. M is described by the tuple (Γ, Q, δ), where Γ denotes the finite alphabet of M containing a blank symbol $\square$ as well as a start symbol $\triangleright$, and the numbers 0 and 1; Q denotes a finite set of states containing a start state q_{start} and a halting state q_{halt}; and $\delta : Q \times \Gamma^4 \to Q \times \Gamma^2 \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^4$ denotes the transition function. We assume that the transition function δ of M is given by a circuit $\mathbf{C}_{\mathsf{M}}$. It is going to be convenient for us to load the input m_B onto the working tape of the Turing machine. For the remainder of this description, we consider the working tape and the input tape of m_B as a single tape. Furthermore, M is an oblivious Turing machine, meaning its head movements do not depend on the input but only on the input length. Note, that by a classical result of Pippinger and Fischer, Turing machines can be simulated by an oblivious (and deterministic) Turing machine with only a logarithmic slowdown [PF79]. For convenience, we denote by $\mathsf{HeadPos}(i)$ the function that outputs the state st', the write location on the working tape I_w, and the read locations I_r, J_r on the input tapes m_B and m_A respectively; all at step i of $\mathbf{C}_{\mathsf{M}}$'s computation.

Description. Our scheme assumes the existence of:

– A symmetric encryption scheme $\Pi := (\mathsf{Sym.Gen}, \mathsf{Sym.Enc}, \mathsf{Sym.Dec})$ that is IND-CPA secure.

[5] One can always make the function F private by including an encoding of F in the input of Alice and computing LFE of a universal Turing machine.

- An updatable laconic oblivious transfer $\mathsf{ULOT} := (\mathsf{ULOT.Gen}, \mathsf{ULOT.Hash}, \mathsf{ULOT.Send}, \mathsf{ULOT.Receive}, \mathsf{ULOT.SendWriteRead}, \mathsf{ULOT.ReceiveWriteRead})$ with sender privacy against semi-honest receivers.
- An indistinguishability obfuscator iO.
- A garbling scheme $\mathsf{GC} := (\mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Input})$ with selective security.
- A puncturable pseudorandom function $\mathsf{PPRF} := (\mathsf{PPRF.Gen}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Punc})$.

For convenience we make a few simplifying assumptions: (1) The Turing machine never writes to the same position twice (this does not affect its runtime, as we can just write to a new memory location every time) and (2) The input m_B is of length exactly N. Our scheme can be modified to handle the more general case but the description and the proof become somewhat more contrived.

The step circuit [Fig. 2] handles the tasks performed at each step of M's computation. Namely, decrypting the secret input into $\mathbf{C}_\mathsf{M}$, computing one step of $\mathbf{C}_\mathsf{M}$ and encrypting the output with a new key. Furthermore, after each step, additional outputs are used to specify a location in the database where the encrypted data is to be written using the updatable laconic oblivious transfer. The garbling step circuit [Fig. 3] garbles each step circuit and generates the relevant labels and keys so that the garbled circuit can be evaluated.

We define the step circuit $\mathbf{SC}_i$ as in Fig. 2. As inputs, $\mathbf{C}_\mathsf{M}$ takes the state $\mathsf{st} \in Q$ of the Turing machine M, as well as two input blocks $x_A \subseteq m_A$ and $x_B = m_B$ both of size N. After evaluating the circuit on its inputs, $\mathbf{C}_\mathsf{M}$ returns a new state $\mathsf{st}' \in Q$; a write location I_w on the working tape, at which the next block of symbols y_B is written; a read location I_r on the input tape m_B; a read location J_r on the input tape m_A; and $q = \perp$, unless the halting state q_halt has been reached, in which case q is the only output of the computation.

$\mathbf{SC}_i\left[\mathsf{crs}, \mathsf{k}_i, \mathsf{k}_{i+1}, \mathsf{lbs}_\mathsf{st}, \mathsf{lbs}_A, \mathsf{lbs}_B\right](\mathsf{st}, z_A, z_B)$:

1. Parse $(\mathsf{d}_A, x_A) := z_A$
2. Parse $(\mathsf{d}_B, x'_B) := z_B$
3. Parse $\left(\mathsf{lbs}_{B[0]}, \mathsf{lbs}_{B[1]}\right) := \mathsf{lbs}_B$
4. $x_B \leftarrow \mathsf{Sym.Dec}(\mathsf{k}_i, x'_B)$
5. $(\mathsf{st}', I_w, y_B, I_r, J_r, q) \leftarrow \mathbf{C}_\mathsf{M}(\mathsf{st}, x_A, x_B)$
6. $y'_B \leftarrow \mathsf{Sym.Enc}(\mathsf{k}_{i+1}, y_B)$
7. $e_A \leftarrow \mathsf{ULOT.Send}(\mathsf{crs}, \mathsf{d}_A, J_r, \mathsf{lbs}_A)$
8. $e_B \leftarrow \mathsf{ULOT.SendWriteRead}\left(\mathsf{crs}, \mathsf{d}_B, I_w, y'_B, \mathsf{lbs}_{B[0]}, I_r, \mathsf{lbs}_{B[1]}\right)$
9. $\widehat{\mathsf{st}} \leftarrow \mathsf{GC.Input}(\mathsf{st}', \mathsf{lbs}_\mathsf{st})$

return $\left(\widehat{\mathsf{st}}, I_w, y'_B, I_r, J_r, e_A, e_B, q\right)$

Figure 2. Step Circuit.

Now we define the following circuit **GarbleSC**, which has the crs and a PRF seed s hardwired [Fig. 3]. It takes as input an index i and outputs a garbled circuit $\mathbf{GC}^{(i)}$. We are now ready to present our laconic function evaluation protocol [Fig. 4].

3.1 Correctness

The correctness of our LFE construction follows routinely from the correctness of its components, namely the indistinguishability obfuscator iO, the garbling scheme GC, the updatable laconic oblivious transfer protocol ULOT, the symmetric encryption scheme Π and the puncturable pseudorandom function PPRF.

Proposition 1 (Correctness). *The Laconic Function Evaluation protocol in Fig. 4 is correct.*

GarbleSC[crs, s, k](i):

1. $(\mathsf{lbs}_{\mathsf{st}} \,||\, \mathsf{lbs}_A \,||\, \mathsf{lbs}_B \,||\, R) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i)$
2. $(\mathsf{lbs}'_{\mathsf{st}} \,||\, \mathsf{lbs}'_A \,||\, \mathsf{lbs}'_B \,||\, \cdot) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i+1)$
3. $(\mathsf{st}, I_w, I_r, J_r) \leftarrow \mathsf{HeadPos}(i)$
4. $(\mathsf{st}', I'_w, I'_r, J'_r) \leftarrow \mathsf{HeadPos}(i+1)$
5. $\mathsf{k}_i \leftarrow \mathsf{PPRF.Eval}(k, I_w)$
6. $\mathsf{k}_{i+1} \leftarrow \mathsf{PPRF.Eval}(k, I'_w)$
7. $\mathbf{C}' \leftarrow \mathbf{SC}_i\left[\mathsf{crs}, \mathsf{k}_i, \mathsf{k}_{i+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B\right]$
8. $\mathbf{GC} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathbf{C}', (\mathsf{lbs}_{\mathsf{st}} \,\big|\big|\, \mathsf{lbs}_A \,\big|\big|\, \mathsf{lbs}_B; R)\right)$

   return GC

Figure 3. Garbling Step Circuit. The circuit is padded to the maximum size of $\mathsf{Sim}_{\mathbf{GarbleSC}}$ [See proof of Theorem 3].

Proof of Proposition 1. We prove the claim via an inductive argument. Let $c_B^{(i)}$ denote the contents of the databases at the beginning of the i^{th} iteration of the while loop in $\mathsf{LFE.Dec}$. Let tr'_i denote the transcript tr_i of M, except that we remove Alice's input tape m_A, and let T denote the runtime of M. We argue that $\forall i \in \{1, \ldots, T\}$, $c_B^{(i)}$ block-wise decrypts to the transcript tr'_i at step i of M's computation. We also show that at each i, the garbled input labels $\left(\widehat{\mathsf{st}}^{(i)} \,\Big|\Big|\, \widehat{z}_A^{(i)} \,\Big|\Big|\, \widehat{z}_B^{(i)}\right)$ are a valid encoding of the state of the Turing machine M, d_A the block of m_A, and d_B the block of c_B all in step circuit $\mathbf{SC}_i$.

The base case, when $i = 1$, follows trivially. Initially, the database $c_B^{(1)}$ contains a block-wise encryption of m_B [step 5 of $\mathsf{LFE.Enc}$]. In step 9 of $\mathsf{LFE.Enc}$ x'_B is set to $c_B^{(1)}$, i.e. x'_B contains m_B and the content of the (empty) worktape. Similarly, x_A is also initialised to 0^N in step 8 of $\mathsf{LFE.Enc}$. Hence, the transcript tr'_1 consists of the input tape m_B concatenated with an empty working tape and the state. Thus, $\mathsf{Sym.Dec}\left(\mathsf{k}_1, c_B^{(1)}\right) = \mathsf{tr}'_1$. The garbled input labels $\left(\widehat{\mathsf{st}}^{(1)} \,\Big|\Big|\, \widehat{z}_A^{(1)} \,\Big|\Big|\, \widehat{z}_B^{(1)}\right)$ are passed to $\mathsf{LFE.Dec}$ in the ciphertext.

By the inductive hypothesis we assume that the database $c_B^{(i-1)}$ block-wise decrypts to give tr'_{i-1}. We now show that $\mathsf{Sym.Dec}\left(\mathsf{k}_i, c_B^{(i)}\right) = \mathsf{tr}'_i$. In the i^{th} iteration of the while loop in $\mathsf{LFE.Dec}$, $\mathbf{SC}_i$ is evaluated by $\mathbf{GC}^{(i)}$. Due to the correctness of the indistinguishability obfuscator iO, the obfuscated garbling step circuit $\mathsf{ob}\mathbf{G}$ can be correctly evaluated on input i, and $\mathbf{GC}^{(i)}$ is given by

$$
\begin{aligned}
\mathbf{GC}^{(i)} &= \mathsf{ob}\mathbf{G}(i) \\
&= \mathsf{iO}\left(\mathbf{GarbleSC}[\mathsf{crs}, \mathsf{s}, k](i)\right) \\
&= \mathsf{GC.Garble}\left(1^\lambda, \mathbf{SC}_i\left[\mathsf{crs}, \mathsf{k}_i, \mathsf{k}_{i+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B\right](\cdot,\cdot,\cdot), \mathsf{lbs}_{\mathsf{st}} || \,\mathsf{lbs}_A || \,\mathsf{lbs}_B; R\right).
\end{aligned}
$$

By the induction hypothesis, the garbled input labels $\left(\widehat{\mathsf{st}}^{(i)} \,\Big|\Big|\, \widehat{z}_A^{(i)} \,\Big|\Big|\, \widehat{z}_B^{(i)}\right)$ are a valid encoding of the state of the Turing machine M, d_A and the block of m_A, and d_B and the block of c_B all in step circuit $\mathbf{SC}_i$. In $\mathbf{SC}_i$, the decryption of ${x'}_B^{(i)}$ gives $x_B^{(i)}$. After running $\mathbf{C}_{\mathsf{M}}$, $y_B^{(i)}$ is then written to the work tape at $I_w^{(i)}$, and encrypted to ${y'}_B^{(i)}$. By the correctness of updatable laconic oblivious transfer, $\mathsf{ULOT.SendWriteRead}$ specifies ${y'}_B^{(i)}$ to be written to a database and in step 5 of $\mathsf{LFE.Dec}$, ${y'}_B^{(i)}$ is written to c_B at position $I_w^{(i)}$. Therefore, $\mathsf{Sym.Dec}\left(\mathsf{k}_i, c_B^{(i)}\right) = \mathsf{tr}'_{i-1}$ with $y_B^{(i)}$ written on the work tape at $I_w^{(i)}$. I.e., $\mathsf{Sym.Dec}\left(\mathsf{k}_i, c_B^{(i)}\right) = \mathsf{tr}'_i$. Furthermore, the garbled input labels $\widehat{z}_A^{(i+1)}$ and $\widehat{z}_B^{(i+1)}$ are given by

$$
\begin{aligned}
\widehat{z}_A^{(i+1)} &= \mathsf{ULOT.Receive}^{m_A^{(i)}}\left(\mathsf{crs}, e_A^{(i)}, J_r^{(i)}\right) \\
&= \mathsf{ULOT.Receive}^{m_A^{(i)}}\left(\mathsf{crs}, \mathsf{ULOT.Send}\left(\mathsf{crs}, \mathsf{d}_A, J_r^{(i)}, \mathsf{lbs}_A\right), J_r^{(i)}\right),
\end{aligned}
$$

$\mathsf{LFE.Gen}\left(1^\lambda, 1^N\right)$:

1. Compute $\mathsf{crs} \leftarrow \mathsf{ULOT.Gen}\left(1^\lambda, 1^N\right)$
   return crs

$\mathsf{LFE.Hash}(\mathsf{crs}, m_A)$:

1. Compute $(\mathsf{d}_A, \widehat{m_A}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, m_A)$
   return $(\mathsf{d}_A, \widehat{m_A})$

$\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}_A, m_B)$:

1. Choose two uniformly random PRF seeds (s, k)
2. Compute $(\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B \mid\mid R) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, 1)$
3. Compute $\mathsf{k}_1 \leftarrow \mathsf{PPRF.Eval}(k, 1)$
4. Compute $\mathsf{ob}\mathbf{G} \leftarrow \mathsf{iO}\left(\mathbf{GarbleSC}[\mathsf{crs}, \mathsf{s}, k]\right)$
5. Block-wise encrypt $c_B \leftarrow \mathsf{Sym.Enc}(\mathsf{k}_1, m_B)$
6. Compute $(\mathsf{d}_B, \widehat{c_B}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, c_B)$
7. Set $\mathsf{st} \leftarrow 0^N$
8. Set $z_A \leftarrow \left(\mathsf{d}_A, 0^N\right)$
9. Set $z_B \leftarrow \left(\mathsf{d}_B, c_B\right)$
10. Compute $\widehat{\mathsf{st}} \leftarrow \mathsf{GC.Input}(\mathsf{st}, \mathsf{lbs}_{\mathsf{st}})$
11. Compute $\widehat{z}_A \leftarrow \mathsf{GC.Input}(z_A, \mathsf{lbs}_A)$
12. Compute $\widehat{z}_B \leftarrow \mathsf{GC.Input}(z_B, \mathsf{lbs}_B)$
13. Set $\mathsf{c} \leftarrow \left(\widehat{c_B}, \mathsf{ob}\mathbf{G}, \widehat{\mathsf{st}}, \widehat{z}_A, \widehat{z}_B\right)$
    return c

$\mathsf{LFE.Dec}(\mathsf{crs}, m_A, \mathsf{c})$:

1. Parse $\left(\widehat{c_B}, \mathsf{ob}\mathbf{G}, \widehat{\mathsf{st}}, \widehat{z}_A, \widehat{z}_B\right) := \mathsf{c}$
2. Set $m_B^{(1)} \leftarrow \widehat{c_B}$, $\widehat{\mathsf{st}}^{(1)} \leftarrow \widehat{\mathsf{st}}$, $\widehat{z}_A^{(1)} \leftarrow \widehat{z}_A$, $\widehat{z}_B^{(1)} \leftarrow \widehat{z}_B$
3. Set $i := 1$
4. $q := \perp$
5. **while** true **do**
   if $q \neq \perp$ **then**
   return q
   Compute $\mathbf{GC}^{(i)} \leftarrow \mathsf{ob}\mathbf{G}(i)$
   Compute $\left(\widehat{\mathsf{st}}^{(i+1)} \,\middle|\middle|\, I_w \,\middle|\middle|\, m_B^{(i+1)} \,\middle|\middle|\, I_r \,\middle|\middle|\, J_r \,\middle|\middle|\, e_A \,\middle|\middle|\, e_B \,\middle|\middle|\, q\right) \leftarrow \mathsf{GC.Eval}\left(\mathbf{GC}^{(i)}, \left(\widehat{\mathsf{st}}^{(i)} \,\middle|\middle|\, \widehat{z}_A^{(i)} \,\middle|\middle|\, \widehat{z}_B^{(i)}\right)\right)$
   Compute $\widehat{z}_A^{(i+1)} \leftarrow \mathsf{ULOT.Receive}^{m_A}\left(\mathsf{crs}, e_A, J_r\right)$
   Compute $\widehat{z}_B^{(i+1)} \leftarrow \mathsf{ULOT.ReceiveWriteRead}^{\widehat{c_B}}\left(\mathsf{crs}, I_w, m_B^{(i)}, e_B, I_r\right)$
   Set $i := i + 1$

Figure 4. Laconic Function Evaluation Protocol.

and

$$\begin{aligned}
\widehat{z}_B^{(i+1)} &= \mathsf{UL\ OT.ReceiveWriteRead}^{\widehat{c_B^{(i)}}}\left(\mathsf{crs}, I_w^{(i)}, m_B^{(i)}, e_B^{(i)}, I_r^{(i)}\right) \\
&= \mathsf{UL\ OT.ReceiveWriteRead}^{\widehat{c_B^{(i)}}}\Big(\mathsf{crs}, I_w^{(i)}, m_B^{(i)}, \\
&\qquad \mathsf{ULOT.SendWriteRead}\left(\mathsf{crs}, \mathsf{d}_B, I_w^{(i)}, {y'}_B^{(i)}, \mathsf{lbs}_{B[0]}, I_r, \mathsf{lbs}_{B[1]}\right), I_r^{(i)}\Big),
\end{aligned}$$

respectively. ■

3.2 Proof of Security

We will now establish sender simulation security for our protocol, and start by stating the main security theorem.

Theorem 3 (Security). *Assume that* iO *is an indistinguishability obfuscator,* $(\mathsf{GC.Garble}, \mathsf{GC.Input}, \mathsf{GC.Eval})$ *is simulation secure,* $(\mathsf{ULOT.Gen}, \mathsf{ULOT.Hash}, \mathsf{ULOT.Send}, \mathsf{ULOT.Receive}, \mathsf{ULOT.SendWriteRead}, \mathsf{ULOT.ReceiveWriteRead})$ *has sender privacy against semi-honest receivers,* $(\mathsf{Sym.Gen}, \mathsf{Sym.Enc}, \mathsf{Sym.Dec})$ *is IND-CPA secure, and that* $(\mathsf{PPRF.Gen}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Punc})$ *is a puncturable pseudorandom function. Then* $(\mathsf{LFE.Gen}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ *has sender privacy against semi-honest receivers.*

To prove the security of our construction we use a similar proof strategy to that of [GS18]. In particular, our proof will proceed via a hybrid argument. In each hybrid we change the way the circuit $\mathsf{ob}\mathbf{G}$ computes the garbled circuits $\mathbf{GC}^{(i)}$. Each garbled step circuit $\mathbf{GC}^{(i)}$ can be computed in three modes [Fig. 8]. The first mode is real, where the computations are just as in the real protocol. The second mode is dummy, where the output of the garbled circuit is constant and hardwired, but the same as in the real execution [Fig. 5-6]. The third mode is sim, which is similar to real mode, with the difference being that the garbled circuit only outputs dummy values which are not the same as in the real execution [Fig. 2].

Both garbled circuits in real and dummy mode will keep the intermediate states and memory consistent (recall that the memory is accessed via an updatable laconic OT). On the other hand, a garbled circuit in sim mode will only output the dummy state and perform dummy read and writes to memory. Garbled circuits in real and sim mode are computed on-the-fly by $\mathsf{ob}\mathbf{G}$, whereas circuits in dummy need to be hardwired into $\mathsf{ob}\mathbf{G}$. As a result, the size of $\mathsf{ob}\mathbf{G}$ depends on the maximum number of dummy circuits needed in any given hybrid.

We will briefly discuss the necessary conditions under which we can switch the mode of a garbled step circuit. The first garbled circuit in the in line $\mathbf{GC}^{(1)}$ can always be switched from real to dummy or vice versa, provided there is a free simulation slot available, i.e., the number of currently simulated garbled circuits is less than some maximum amount t. For any other garbled circuit $\mathbf{GC}^{(i)}$, we can switch its mode from real to dummy or vice versa, given that the circuit $\mathbf{GC}^{(i-1)}$ is in dummy mode and a simulation slot is available. To switch a node into sim mode, we require that its successor node is in sim mode and that its predecessor is in dummy mode. In the case of the first node we only have the requirement for its successor node and for the last node we only have the requirement for its predecessor.

These rules define a pebbling game, where we identify pebbles as simulation slots. The goal of the game is to switch the nodes from real (white pebbles) to sim (black pebbles), while minimizing the number of nodes in dummy (grey pebbles). To win the game, we can use the same pebbling strategy as in [GS18], where $\mathcal{O}(\log(T))$ pebbles suffice to set a pebble at the last node (with index T) in $\mathsf{poly}(T)$ steps. Consequently, with this strategy we only need to simulate $\mathcal{O}(\log(T)) = \mathcal{O}(\lambda)$ nodes in any given hybrid. We refer the reader to the works of [GPSZ17] and [GS18] for an optimal strategy for the pebbling game. For the sake of completeness we state the main Lemmas here.

Lemma 1 ([GPSZ17]). *For any* $p \in \mathbb{Z}$*, such that* $n + 1 \leq p \leq n + 2^k - 1$*, it is possible to make* $\mathcal{O}\left((p-n)^{\log_2 3}\right) \approx \mathcal{O}\left((p-n)^{1.585}\right)$ *moves and get a black pebble at position* p *using* k *gray pebbles.*

Lemma 2 ([GS18]). *For any* $T \in \mathbb{N}$*, there exists a strategy for pebbling the line graph* $\{1, \ldots, T\}$ *according to rules* $\mathfrak{A}$ *and* $\mathfrak{B}$ *by using at most* $\log(T)$ *grey pebbles and making* $\mathsf{poly}(\lambda)$ *moves.*

Thus, our proof strategy will proceed as follows. First we will use the above pebbling argument to switch the last node, i.e. the node with index T to sim mode. This will take $\mathsf{poly}(T)$ steps. Next, we will again use the same pebbling argument to switch node $T-1$ to sim mode. This will take $\mathsf{poly}(T-1) = \mathsf{poly}(T)$ steps. Consequently, we replace nodes $T-2, T-3, \ldots, 2, 1$ with sim nodes, in this order. In total, this will require $T \cdot \mathsf{poly}(T) = \mathsf{poly}(T)$ steps. In the very last hybrid, we will replace the encryption of that database m_B by an encryption of 0. Once all pebbles (step circuits) are in sim mode, and the encryption of m_B has been replaced with the encryption of 0, this corresponds to the simulator $\mathsf{Sim}_{\mathsf{LFE}}$, which takes as input the crs, d, the machine M, the output $\mathsf{M}(x)$ and the time bound T. The simulator then outputs the ciphertext c. As a result, the view of the adversary, in this last hybrid, is independent of the sender input m_B. Hence, we can use this hybrid to simulate the view of a semi-honest receiver by only using the receiver's output. The full proof of Theorem 3 follows from that of two lemmas [Lem. 3, Lem. 4].

$\mathbf{SC}^{\mathsf{dummy}}_{i^*}[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}_{\mathsf{st}}, \mathsf{lbs}_A, \mathsf{lbs}_B](\mathsf{st}, z_A, z_B)$:

1. Parse $(\mathsf{d}_A, x_A) := z_A$
2. Parse $(\mathsf{d}_B, x'_B) := z_B$
3. Parse $\left(\mathsf{lbs}_{B[0]}, \mathsf{lbs}_{B[1]}\right) := \mathsf{lbs}_B$
4. $x_B \leftarrow \mathsf{Sym.Dec}(\mathsf{k}_{i^*}, x'_B)$
5. $(\mathsf{st}', I_w, y_B, I_r, J_r, q) \leftarrow \mathbf{C}_{\mathsf{M}}(\mathsf{st}, x_A, x_B)$
6. $y'_B \leftarrow \mathsf{Sym.Enc}(\mathsf{k}_{i^*+1}, y_B)$
7. $\mathsf{d}^*_B \leftarrow \mathsf{ULOT.Hash}\left(\mathsf{crs}, m^*_B\right)$
8. $e_A \leftarrow \mathsf{Sim}_{\mathsf{ULOT.S}}\left(\mathsf{crs}, m_A, J_r, \mathsf{GC.Input}\left(\mathsf{lbs}_A, m_{A[J_r]}\right)\right)$
9. $e_B \leftarrow \mathsf{Sim}_{\mathsf{ULOT.WR}}\Big(\mathsf{crs}, m_B, I_w, y'_B, \mathsf{GC.Input}\left(\mathsf{lbs}_{B[0]}, \mathsf{d}^*_B\right), I_r,$
   $\mathsf{GC.Input}\left(\mathsf{lbs}_{B[1]}, m^*_{B[I_r]}\right)\Big)$
10. $\widehat{\mathsf{st}} \leftarrow \mathsf{GC.Input}(\mathsf{st}', \mathsf{lbs}_{\mathsf{st}})$
    return $\left(\widehat{\mathsf{st}}, I_w, y'_B, I_r, J_r, e_A, e_B, q\right)$

Figure 5. Step Circuit in dummy mode. Let m^*_B denote the database that is identical to m_B except that $m^*_B[I_w] = y'_B$.

$\mathbf{GC}^{\mathsf{dummy}}_{i^*}[\mathsf{crs}, \mathsf{s}, k]$:

1. $(\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B \mid\mid R) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i)$
2. $(\mathsf{lbs}'_{\mathsf{st}} \mid\mid \mathsf{lbs}'_A \mid\mid \mathsf{lbs}'_B \mid\mid \cdot) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i+1)$
3. $(\mathsf{st}, I_w, I_r, J_r) \leftarrow \mathsf{HeadPos}(i)$
4. $(\mathsf{st}', I'_w, I'_r, J'_r) \leftarrow \mathsf{HeadPos}(i+1)$
5. $\mathsf{k}_i \leftarrow \mathsf{PPRF.Eval}(k, I_w)$
6. $\mathsf{k}_{i+1} \leftarrow \mathsf{PPRF.Eval}(k, I'_w)$
7. $\mathsf{L}_{\mathsf{st}} \leftarrow \mathsf{GC.Input}\left(\mathsf{lbs}_{\mathsf{st}}, \mathsf{st}^{(i^*)}\right)$
8. $\mathsf{L}_A \leftarrow \mathsf{GC.Input}\left(\mathsf{lbs}_A, z_A^{(i^*)}\right)$
9. $\mathsf{L}_B \leftarrow \mathsf{GC.Input}\left(\mathsf{lbs}_B, z_B^{(i^*)}\right)$
10. $\mathsf{out} \leftarrow \mathbf{SC}^{\mathsf{dummy}}_{i^*}\left[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B\right]\left(\mathsf{st}^{(i^*+1)}, z_A^{(i^*+1)}, z_B^{(i^*+1)}\right)$
11. $\mathbf{GC} \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|\mathbf{SC}^{\mathsf{dummy}}_{i^*}|}, \mathsf{out}, (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right)$
    return GC

Figure 6. Garbling Step Circuit in dummy mode.

Circuit Configuration. A circuit configuration conf consists of a subset of garbling step circuits in dummy mode as well as an index $i^* \in \{1, \ldots, T\}$ denoting the garbling step circuit to be changed by the rule.

Rules of Indistinguishability. We define the rules of indistinguishability (which determine the configurations in the pebbling game) below.

Rule $\mathfrak{A}$: Rule $\mathfrak{A}$ dictates when a garbling step circuit can be indistinguishably changed from real mode to dummy mode. Let conf and conf′ be two valid configurations and i^* be an index of the garbling step circuit, such that:
- Index i^* is changed from real mode to dummy mode, and there are no indices in sim mode to the left of i^*.
- Index i^* is either the first or its predecessor is in dummy mode.

$\mathbf{SC}^{\mathsf{sim}}_{i^*}[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}_{\mathsf{st}}, \mathsf{lbs}_A, \mathsf{lbs}_B](\mathsf{st}, z_A, z_B)$:

1. Parse $(\mathsf{d}_A, x_A) := z_A$
2. Parse $(\mathsf{d}_B, x'_B) := z_B$
3. Parse $\left(\mathsf{lbs}_{B[0]}, \mathsf{lbs}_{B[1]}\right) := \mathsf{lbs}_B$
4. $(\mathsf{st}', I_w, I_r, J_r) \leftarrow \mathsf{HeadPos}(i^*)$
5. $y'_B \leftarrow \mathsf{Sym.Enc}(\mathsf{k}_{i^*+1}, 0)$
6. **if** $i^* = T$ **then**
   $q := \mathbf{C}(x)$
7. **else**
   $q := \perp$
8. $e_A \leftarrow \mathsf{ULOT.Send}(\mathsf{crs}, \mathsf{d}_A, J_r, \mathsf{lbs}_A)$
9. $e_B \leftarrow \mathsf{ULOT.SendWriteRead}\left(\mathsf{crs}, \mathsf{d}_B, I_w, y'_B, \mathsf{lbs}_{B[0]}, I_r, \mathsf{lbs}_{B[1]}\right)$
10. $\widehat{\mathsf{st}} \leftarrow \mathsf{GC.Input}(\mathsf{st}', \mathsf{lbs}_{\mathsf{st}})$
    return $\left(\widehat{\mathsf{st}}, I_w, y'_B, I_r, J_r, e_A, e_B, q\right)$

Figure 7. Step Circuit in sim mode.

$\mathsf{Sim}_{\mathbf{GarbleSC}}[\mathsf{crs}, \mathsf{s}](i^*)$:

1. **if** $i^* \in \mathsf{dummy}$ **then**
   return $\mathbf{GC}^{\mathsf{dummy}}_{i^*}[\mathsf{crs}, \mathsf{s}, k]$
2. **else**
   $(\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B \mid\mid R) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i)$
   $(\mathsf{lbs}'_{\mathsf{st}} \mid\mid \mathsf{lbs}'_A \mid\mid \mathsf{lbs}'_B \mid\mid \cdot) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i+1)$
   $(\mathsf{st}, I_w, I_r, J_r) \leftarrow \mathsf{HeadPos}(i)$
   $(\mathsf{st}', I'_w, I'_r, J'_r) \leftarrow \mathsf{HeadPos}(i+1)$
   $\mathsf{k}_i \leftarrow \mathsf{PPRF.Eval}(k, I_w)$
   $\mathsf{k}_{i+1} \leftarrow \mathsf{PPRF.Eval}(k, I'_w)$
3. **if** $i^* \in \mathsf{real}$ **then**
   Set $\mathbf{C}' \leftarrow \mathbf{SC}_{i^*}[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B]$
4. **if** $i^* \in \mathsf{sim}$ **then**
   Set $\mathbf{C}' \leftarrow \mathbf{SC}^{\mathsf{sim}}_{i^*}[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B]$
   $\mathbf{GC} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathbf{C}', (\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B; R)\right)$
   return GC

Figure 8. Garbling Step Circuit in real, dummy and sim mode.

– The garbling step circuits in sim mode remain unchanged.

In Lemma 3 we show that for two valid circuit configurations conf and conf', satisfying the above constraints, the two distributions $\mathcal{H}_{\mathsf{conf}}$ and $\mathcal{H}_{\mathsf{conf}'}$ are computationally indistinguishable.

Rule $\mathfrak{B}$: Rule $\mathfrak{B}$ dictates when a step circuit can be indistinguishably changed from dummy mode to sim mode. Let conf and conf' be two valid configurations and i^* be an index of the garbling step circuit, such that:

– Index i^* is changed from dummy mode to sim mode.
– Index i^* is either the last or its predecessor is in dummy mode.
– The garbling step circuits in real mode remain unchanged.

In Lemma 4 we show that for two valid circuit configurations conf and conf', satisfying the above constraints, the two distributions $\mathcal{H}_{\mathsf{conf}}$ and $\mathcal{H}_{\mathsf{conf}'}$ are computationally indistinguishable.

3.3 Proof of Indistinguishability for the Rules

Implementing Rule $\mathfrak{A}$

Lemma 3 (Rule $\mathfrak{A}$). *Let* conf *and* conf' *be two valid circuit configurations satisfying the constraints of rule* $\mathfrak{A}$. *Assume that* iO *is an indistinguishability obfuscator,* GC *is simulation secure,* ULOT *has sender privacy against semi-honest receivers, and that* PPRF *is a puncturable pseudorandom function. Then, for the two distribution ensembles* $\{\mathcal{H}_{\mathsf{conf}_\lambda}\}_{\lambda\in\mathbb{N}}$ *and* $\{\mathcal{H}_{\mathsf{conf}'_\lambda}\}_{\lambda\in\mathbb{N}}$ *it holds that*

$$\left| \Pr_{\mathsf{c}\leftarrow\mathcal{H}_{\mathsf{conf}_\lambda}}\left[\mathcal{A}\left(1^\lambda,\mathsf{c}\right)=1\right] - \Pr_{\mathsf{c}\leftarrow\mathcal{H}_{\mathsf{conf}'_\lambda}}\left[\mathcal{A}\left(1^\lambda,\mathsf{c}\right)=1\right] \right| \leq \mathsf{negl}(\lambda).$$

Proof of Lemma 3. We prove this with help of a hybrid argument.

$\mathcal{H}_{\mathsf{conf}_\lambda}$**:** The garbling step circuit is in real mode.

$\mathcal{H}_1$**:** Instead of hardwiring the PPRF key s into $\mathsf{Sim}_{\mathbf{GarbleSC}}$, we hardwire the key $\mathsf{s}\{i^*\} \leftarrow \mathsf{PPRF.Punc}(\mathsf{s}, i^*)$, that is punctured at i^*. Since we cannot evaluate $\mathsf{PPRF.Eval}(\mathsf{s}\{i^*\}, i^*)$, we additionally hardwire the labels and key that are output by $\mathsf{PPRF.Eval}(\mathsf{s}, i^*)$ into $\mathsf{Sim}_{\mathbf{GarbleSC}}$.

$$(\mathsf{lbs}_{\mathsf{st}} \,||\, \mathsf{lbs}_A \,||\, \mathsf{lbs}_B \,||\, R) \leftarrow \mathsf{PPRF.Eval}(\mathsf{s}, i^*)$$

To be able to use the security of iO, the size of **GarbleSC** is padded to be the same size as $\mathsf{Sim}_{\mathbf{GarbleSC}}$.

Claim ($\mathcal{H}_{\mathsf{conf}} \to \mathcal{H}_1$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_{\mathsf{conf}}$ and $\mathcal{H}_1$ is:

$$\mathsf{Adv}_{\mathcal{A}_1}^{\mathcal{H}_{\mathsf{conf}}\to\mathcal{H}_1} \leq \mathsf{Adv}_{\mathsf{iO},\mathcal{B}_1}^{\mathsf{iO\text{-}sec}}.$$

$\mathcal{H}_{\mathsf{conf}} \to \mathcal{H}_1$. The proof relies on the security of the indistinguishability obfuscator iO to be able to switch the PPRF key and hardwire the labels. The reduction $\mathcal{B}_1$ gets a bit b from the adversary $\mathcal{A}_1$, where $b = 0$ if the obfuscated circuit is as described in $\mathcal{H}_{\mathsf{conf}}$ and $b = 1$ if the obfuscated circuit is as described in $\mathcal{H}_1$. If $\mathcal{A}_1$ wins the game with advantage ϵ, $\mathcal{B}_1$ wins the $\mathsf{iO\text{-}sec}$ game with greater than ϵ probability. □

$\mathcal{H}_2$**:** As opposed to using the labels output by $\mathsf{PPRF.Eval}(\mathsf{s}, i^*)$, we sample a string u from the uniform distribution U_λ.

Claim ($\mathcal{H}_1 \to \mathcal{H}_2$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_1$ and $\mathcal{H}_2$ is:

$$\mathsf{Adv}_{\mathcal{A}_2}^{\mathcal{H}_1\to\mathcal{H}_2} \leq \mathsf{Adv}_{\mathsf{PPRF},\mathcal{B}_2}^{\mathsf{PPRF\text{-}rand}}.$$

$\mathcal{H}_1 \to \mathcal{H}_2$. The proof relies on the pseudorandomness property of PPRF, to be able to switch the output of $\mathsf{PPRF.Eval}(\mathsf{s}, i^*)$ with u. The reduction $\mathcal{B}_2$ gets a bit b from the adversary $\mathcal{A}_2$, where $b = 0$ if the output of $\mathsf{PPRF.Eval}(\mathsf{s}, i^*)$ is used, and $b = 1$ if the uniform string is used. If $\mathcal{A}_2$ wins the game with advantage ϵ, $\mathcal{B}_2$ wins the $\mathsf{PPRF\text{-}rand}$ game with greater than ϵ probability. □

$\mathcal{H}_3$**:** Since each label is computed twice, once in step $i^* - 1$ and once in step i^*, we now remove the following labels at step $i^* - 1$;

$$\mathsf{lbs}_{\mathsf{st}} \setminus \mathsf{GC.Input}\left(\mathsf{lbs}_{\mathsf{st}}, \mathsf{st}^{(i^*-1)}\right)$$
$$\mathsf{lbs}_A \setminus \mathsf{GC.Input}\left(\mathsf{lbs}_A, z_A^{(i^*-1)}\right)$$
$$\mathsf{lbs}_B \setminus \mathsf{GC.Input}\left(\mathsf{lbs}_B, z_B^{(i^*-1)}\right).$$

I.e., those used in steps 8-10 in $\mathbf{SC}_{i^*-1}^{\mathsf{dummy}}$. This is possible, since by the constraints of rule $\mathfrak{A}$, the previous step is known to be in dummy mode.

Claim ($\mathcal{H}_2 \to \mathcal{H}_3$). The distributions $\mathcal{H}_2$ and $\mathcal{H}_3$ are identical.

$\mathcal{H}_2 \to \mathcal{H}_3$. We note that $\mathbf{SC}_{i^*}^{\mathsf{dummy}}$ is not executed in the obfuscated circuit, but rather computed locally by the simulator. The output is hardwired in the obfuscated circuit, and we are simply removing unused variables. $\square$

$\mathcal{H}_4$**:** We hardwire the output out of

$$\mathsf{GC.Garble}\left(1^\lambda, \mathbf{SC}_{i^*}\left[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}', \mathsf{lbs}'_A, \mathsf{lbs}'_B\right], (\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B; R)\right)$$

into $\mathsf{Sim}_{\mathbf{GarbleSC}}$. iO reduction.

Claim ($\mathcal{H}_3 \to \mathcal{H}_4$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_3$ and $\mathcal{H}_4$ is:

$$\mathsf{Adv}_{\mathcal{A}_4}^{\mathcal{H}_3 \to \mathcal{H}_4} \leq \mathsf{Adv}_{\mathsf{iO}, \mathcal{B}_4}^{\mathsf{iO\text{-}sec}}.$$

$\mathcal{H}_3 \to \mathcal{H}_4$. The proof relies on the security of the indistinguishability obfuscator iO to be able to hardwire the output of the garbling scheme. The reduction $\mathcal{B}_4$ gets a bit b from the adversary $\mathcal{A}_4$, where $b = 0$ if the obfuscated circuit is as described in $\mathcal{H}_3$ and $b = 1$ if the obfuscated circuit is as described in $\mathcal{H}_4$. If $\mathcal{A}_4$ wins the game with advantage ϵ, $\mathcal{B}_4$ wins the $\mathsf{iO\text{-}sec}$ game with greater than ϵ probability. $\square$

$\mathcal{H}_5$**:** We simulate the garbling step circuit, as

$$\mathbf{GC} \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|\mathbf{SC}_{i^*}|}, \mathsf{out}, (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right),$$

where $\mathsf{out} \leftarrow \mathbf{SC}_{i^*}\left[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B\right]\left(\mathsf{st}^{(i^*+1)}, z_A^{(i^*+1)}, z_B^{(i^*+1)}\right)$, and $(\mathsf{lbs}'_{\mathsf{st}} \mid\mid \mathsf{lbs}'_A \mid\mid \mathsf{lbs}'_B \mid\mid R') \leftarrow \mathsf{PPRF.Eval}\left(\mathsf{s}\{i^*\}, i^*+1\right)$. Recall that $\mathsf{st}^{(i^*+1)}, z_A^{(i^*+1)}$, and $z_B^{(i^*+1)}$ denote the state of the Turing machine M; the digest d_A and the input block x_A; as well as the digest d_B and the encrypted input block x_B, respectively, each at step i^*+1 of the computation.

Claim ($\mathcal{H}_4 \to \mathcal{H}_5$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_4$ and $\mathcal{H}_5$ is:

$$\mathsf{Adv}_{\mathcal{A}_5}^{\mathcal{H}_4 \to \mathcal{H}_5} \leq \mathsf{Adv}_{\mathsf{GC}, \mathcal{B}_5}^{\mathsf{GC\text{-}sec}}.$$

$\mathcal{H}_4 \to \mathcal{H}_5$. The proof relies on the selective simulation security of the garbling scheme GC to be able to simulate the garbling step circuit. The reduction $\mathcal{B}_5$ gets a bit b from the adversary $\mathcal{A}_5$, where $b = 0$ if $\mathcal{A}_5$ identified

$$\left\{\mathsf{GC.Garble}\left(1^\lambda, \mathbf{SC}_{i^*}\left[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B\right], (\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B; R)\right), (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right\}$$

and $b = 1$ if $\mathcal{A}_5$ identified

$$\left\{\mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|\mathbf{SC}_{i^*}|}, \mathsf{out}, (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right), (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right\}.$$

If $\mathcal{A}_5$ wins the game with advantage ϵ, $\mathcal{B}_5$ wins the $\mathsf{GC\text{-}sec}$ game with greater than ϵ probability. $\square$

$\mathcal{H}_6$**:** We simulate the $\mathsf{ULOT.Send}$ ciphertext as $e_A \leftarrow \mathsf{Sim}_{\mathsf{ULOT.S}}\left(\mathsf{crs}, m_A, J_r, \mathsf{GC.Input}\left(\mathsf{lbs}_A, m_{A[J_r]}\right)\right)$. Recall that $m_{A[J_r]}$ denotes M's input tape m_A at read location J_r, all at step i^*.

Claim ($\mathcal{H}_5 \to \mathcal{H}_6$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_5$ and $\mathcal{H}_6$ is:

$$\mathsf{Adv}_{\mathcal{A}_6}^{\mathcal{H}_5 \to \mathcal{H}_6} \leq \mathsf{Adv}_{\mathsf{ULOT}, \mathcal{B}_6}^{\mathsf{SenPriExpt}}.$$

$\mathcal{H}_5 \to \mathcal{H}_6$. The proof relies on the semi-honest sender privacy of ULOT to be able to simulate the ciphertext. The reduction $\mathcal{B}_6$ gets a bit b from the adversary $\mathcal{A}_6$, where $b = 0$ if $\mathcal{A}_6$ identified the ciphertext as

$$\{\mathsf{ULOT.Send}\left(\mathsf{crs}, \mathsf{d}_A, J_r, \mathsf{lbs}_A\right)\}$$

and $b = 1$ if $\mathcal{A}_6$ identified the ciphertext as

$$\left\{\mathsf{Sim}_{\mathsf{ULOT.S}}\left(\mathsf{crs}, m_A, J_r, \mathsf{GC.Input}\left(\mathsf{lbs}_A, m_{A[J_r]}\right)\right)\right\}.$$

If $\mathcal{A}_6$ wins the game with advantage ϵ, $\mathcal{B}_6$ wins the $\mathsf{SenPriExpt}$ game with greater than ϵ probability. □

$\mathcal{H}_7$**:** We simulate the $\mathsf{ULOT.SendWriteRead}$ ciphertext as $e_B \leftarrow \mathsf{Sim}_{\mathsf{ULOT.WR}}\Big(\mathsf{crs}, m_B, I_w, y'_B,$ $\mathsf{GC.Input}\left(\mathsf{lbs}_{B[0]}, \mathsf{d}^*_B\right), I_r, \mathsf{GC.Input}\left(\mathsf{lbs}_{B[1]}, m^*_{B[I_r]}\right)\Big)$. Here, m^*_B denotes the database that is identical to m_B except that $m^*_B[I_w] = y'_B$, and $\mathsf{d}^*_B \leftarrow \mathsf{ULOT.Hash}\left(\mathsf{crs}, m^*_B\right)$. Recall that I_w, I_r, and y'_B denote the write location on the working tape; the read location on the input tape m_B; and the encrypted block of symbols that are output by M, respectively, all step i^* of the computation.

Claim ($\mathcal{H}_6 \to \mathcal{H}_7$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_6$ and $\mathcal{H}_7$ is:

$$\mathsf{Adv}^{\mathcal{H}_6 \to \mathcal{H}_7}_{\mathcal{A}_7} \leq \mathsf{Adv}^{\mathsf{WriReaSenPriExpt}}_{\mathsf{ULOT}, \mathcal{B}_7}.$$

$\mathcal{H}_6 \to \mathcal{H}_7$. The proof relies on the semi-honest sender privacy for writes and reads of ULOT to be able to simulate the ciphertext. The reduction $\mathcal{B}_7$ gets a bit b from the adversary $\mathcal{A}_7$, where $b = 0$ if $\mathcal{A}_7$ identified the ciphertext as

$$\left\{\mathsf{ULOT.SendWriteRead}\left(\mathsf{crs}, \mathsf{d}_B, I_w, y'_B, \mathsf{lbs}_{B[0]}, I_r, \mathsf{lbs}_{B[1]}\right)\right\}$$

and $b = 1$ if $\mathcal{A}_7$ identified the ciphertext as

$$\left\{\mathsf{Sim}_{\mathsf{ULOT.WR}}\left(\mathsf{crs}, m_B, I_w, y'_B, \mathsf{GC.Input}\left(\mathsf{lbs}_{B[0]}, \mathsf{d}^*_B\right), I_r, \mathsf{GC.Input}\left(\mathsf{lbs}_{B[1]}, m^*_{B[I_r]}\right)\right)\right\}.$$

If $\mathcal{A}_7$ wins the game with advantage ϵ, $\mathcal{B}_7$ wins the $\mathsf{WriReaSenPriExpt}$ game with greater than ϵ probability. □

$\mathcal{H}_8 - \mathcal{H}_{10}$**:** Finally, we revert the changes made in $\mathcal{H}_1 - \mathcal{H}_3$. Here, the indistinguishability between $\mathcal{H}_8 - \mathcal{H}_{10}$ follows analogous to that of $\mathcal{H}_1 - \mathcal{H}_3$.

$\mathcal{H}_{\mathsf{conf'}_\lambda}$**:** The step circuit is in dummy mode.

This concludes the proof of Lemma 3. ■

Implementing Rule $\mathfrak{B}$

Lemma 4 (Rule $\mathfrak{B}$). *Let* conf *and* $\mathsf{conf'}$ *be two valid circuit configurations satisfying the constraints of rule* $\mathfrak{B}$. *Assume that* iO *is an indistinguishability obfuscator,* GC *is simulation secure,* ULOT *has sender privacy against semi-honest receivers, and that* PPRF *is a puncturable pseudorandom function. Then, for the two distribution ensembles* $\{\mathcal{H}_{\mathsf{conf}_\lambda}\}_{\lambda \in \mathbb{N}}$ *and* $\{\mathcal{H}_{\mathsf{conf'}_\lambda}\}_{\lambda \in \mathbb{N}}$ *it holds that*

$$\left| \Pr_{\mathsf{c} \leftarrow \mathcal{H}_{\mathsf{conf}_\lambda}}\left[\mathcal{A}\left(1^\lambda, \mathsf{c}\right) = 1\right] - \Pr_{\mathsf{c} \leftarrow \mathcal{H}_{\mathsf{conf'}_\lambda}}\left[\mathcal{A}\left(1^\lambda, \mathsf{c}\right) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

Proof of Lemma 4. We prove this with help of a hybrid argument. To keep the proof similar to that of Lemma 3, we start with hybrid $\mathcal{H}_{\mathsf{conf'}}$ and end with hybrid $\mathcal{H}_{\mathsf{conf}}$.

$\mathcal{H}_{\mathsf{conf'}}$**:** The garbling step circuit is in sim mode.

$\mathcal{H}_1$**:** Same as $\mathcal{H}_1$ in Lemma 3.
$\mathcal{H}_2$**:** Same as $\mathcal{H}_2$ in Lemma 3.
$\mathcal{H}_3$**:** Same as $\mathcal{H}_3$ in Lemma 3.
$\mathcal{H}_4$**:** Instead of hardwiring the PPRF key k into $\mathsf{Sim}_{\mathbf{GarbleSC}}$, we hardwire the key $k\{i^*\} \leftarrow \mathsf{PPRF.Punc}(\mathsf{s}, I_w)$, where I_w is the position of the writing head of the Turing Machine at step i^*. We additionally hardwire the labels and key that are output by $\mathsf{PPRF.Eval}(k, I_w)$ into $\mathsf{Sim}_{\mathbf{GarbleSC}}$.

$$k_{i^*} \leftarrow \mathsf{PPRF.Eval}(k, I_w)$$

To be able to use the security of iO, the size of **GarbleSC** is padded to be the same size as $\mathsf{Sim}_{\mathbf{GarbleSC}}$.

Claim ($\mathcal{H}_3 \to \mathcal{H}_4$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_3$ and $\mathcal{H}_4$ is:

$$\mathsf{Adv}_{\mathcal{A}_4}^{\mathcal{H}_3 \to \mathcal{H}_4} \leq \mathsf{Adv}_{\mathsf{iO}, \mathcal{B}_4}^{\mathsf{iO\text{-}sec}}.$$

$\mathcal{H}_3 \to \mathcal{H}_4$. The proof follows by a reduction to the security of the obfuscator, since the two circuits are functionally equivalent. □

$\mathcal{H}_5$**:** As opposed to using the key output by $\mathsf{PPRF.Eval}(k, I_w)$, we sample a string u from the uniform distribution U_λ.

Claim ($\mathcal{H}_4 \to \mathcal{H}_5$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_4$ and $\mathcal{H}_5$ is:

$$\mathsf{Adv}_{\mathcal{A}_5}^{\mathcal{H}_4 \to \mathcal{H}_5} \leq \mathsf{Adv}_{\mathsf{PPRF}, \mathcal{B}_5}^{\mathsf{PPRF\text{-}rand}}.$$

$\mathcal{H}_4 \to \mathcal{H}_5$. Follows by the pseudorandomness of the puncturable PRF. □

$\mathcal{H}_6$**:** We hardwire the output out of

$$\mathsf{GC.Garble}\left(1^\lambda, \mathbf{SC}_{i^*}^{\mathsf{sim}}\left[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}', \mathsf{lbs}'_A, \mathsf{lbs}'_B\right], (\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B; R)\right)$$

into $\mathsf{Sim}_{\mathbf{GarbleSC}}$.

Claim ($\mathcal{H}_5 \to \mathcal{H}_6$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_5$ and $\mathcal{H}_6$ is:

$$\mathsf{Adv}_{\mathcal{A}_6}^{\mathcal{H}_5 \to \mathcal{H}_6} \leq \mathsf{Adv}_{\mathsf{iO}, \mathcal{C}_6}^{\mathsf{iO\text{-}sec}}.$$

$\mathcal{H}_5 \to \mathcal{H}_6$. The proof relies on the security of the indistinguishability obfuscator iO to be able to hardwire the output of the garbling scheme. The reduction $\mathcal{C}_6$ gets a bit b from the adversary $\mathcal{A}_6$, where $b = 0$ if the obfuscated circuit is as described in $\mathcal{H}_5$ and $b = 1$ if the obfuscated circuit is as described in $\mathcal{H}_6$. If $\mathcal{A}_6$ wins the game with advantage ϵ, $\mathcal{B}_6$ wins the $\mathsf{iO\text{-}sec}$ game with greater than ϵ probability. □

$\mathcal{H}_7$**:** We simulate the garbling step circuit, as

$$\mathbf{GC} \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|\mathbf{SC}_{i^*}^{\mathsf{sim}}|}, \mathsf{out}, (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right),$$

where $\mathsf{out} \leftarrow \mathbf{SC}_{i^*}^{\mathsf{sim}}[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B]\left(\mathsf{st}^{(i^*+1)}, z_A^{(i^*+1)}, z_B^{(i^*+1)}\right)$, and $(\mathsf{lbs}'_{\mathsf{st}} \mid\mid \mathsf{lbs}'_A \mid\mid \mathsf{lbs}'_B; R') \leftarrow \mathsf{PPRF.Eval}\left(\mathsf{s}\{i^*\}, i^*+1\right)$. Recall that $\mathsf{st}^{(i^*+1)}$, $z_A^{(i^*+1)}$, and $z_B^{(i^*+1)}$ denote the state of the Turing machine M; the digest d_A and the input block x_A; as well as the digest d_B and the encrypted input block x_B, respectively, each at step i^*+1 of the computation.

Claim ($\mathcal{H}_6 \to \mathcal{H}_7$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_6$ and $\mathcal{H}_7$ is:

$$\mathsf{Adv}_{\mathcal{A}_7}^{\mathcal{H}_6 \to \mathcal{H}_7} \leq \mathsf{Adv}_{\mathsf{GC}, \mathcal{B}_7}^{\mathsf{GC\text{-}sec}}.$$

$\mathcal{H}_6 \to \mathcal{H}_7$. The proof relies on the selective simulation security of the garbling scheme GC to be able to simulate the garbling step circuit. The reduction $\mathcal{B}_7$ gets a bit b from the adversary $\mathcal{A}_7$, where $b = 0$ if $\mathcal{A}_7$ identified

$$\left\{\mathsf{GC.Garble}\left(1^\lambda, \mathbf{SC}^{\mathsf{sim}}_{i^*}\left[\mathsf{crs}, \mathsf{k}_{i^*}, \mathsf{k}_{i^*+1}, \mathsf{lbs}'_{\mathsf{st}}, \mathsf{lbs}'_A, \mathsf{lbs}'_B\right], (\mathsf{lbs}_{\mathsf{st}} \mid\mid \mathsf{lbs}_A \mid\mid \mathsf{lbs}_B; R)\right), (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right\}$$

and $b = 1$ if $\mathcal{A}_7$ identified

$$\left\{\mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|\mathbf{SC}^{\mathsf{sim}}_{i^*}|}, \mathsf{out}, (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right), (\mathsf{L}_{\mathsf{st}} \mid\mid \mathsf{L}_A \mid\mid \mathsf{L}_B; R)\right\}.$$

If $\mathcal{A}_7$ wins the game with advantage ϵ, $\mathcal{B}_7$ wins the $\mathsf{GC\text{-}sec}$ game with greater than ϵ probability. □

$\mathcal{H}_8$: Same as $\mathcal{H}_6$ in Lemma 3.
$\mathcal{H}_9$: Same as $\mathcal{H}_7$ in Lemma 3.
$\mathcal{H}_{10}$: Instead of computing the state st', the write location I_w, the read locations I_r and J_r using $\mathsf{HeadPos}(i)$, as well as computing y'_B as $\mathsf{Sym.Enc}\left(\mathsf{k}_{i^*+1}, 0\right)$; we compute the output of $\mathbf{C}_{\mathsf{M}}$, and y'_B as $\mathsf{Sym.Enc}\left(\mathsf{k}_{i^*+1}, y_B\right)$.

Claim ($\mathcal{H}_9 \to \mathcal{H}_{10}$). The advantage of any PPT adversary in distinguishing between $\mathcal{H}_9$ and $\mathcal{H}_{10}$ is:

$$\mathsf{Adv}^{\mathcal{H}_9 \to \mathcal{H}_{10}}_{\mathcal{A}_{10}} \leq \mathsf{Adv}_{\Pi, \mathcal{B}_{10}}.$$

$\mathcal{H}_9 \to \mathcal{H}_{10}$. The proof relies on the chosen plaintext attack security of the symmetric encryption scheme Π to be able to switch from encrypting 0 to y_B. We can do this, since the constraints of rule $\mathfrak{B}$ ensure that the next circuit is in sim mode and therefore the key k_{i^*+1} is not present in the view of the distinguisher. The reduction $\mathcal{C}_{10}$ gets a bit b from the adversary $\mathcal{A}_{10}$, where $b = 0$ if the plaintext is 0, and $b = 1$ if the plaintext is y_B. If $\mathcal{A}_{10}$ wins the game with advantage ϵ, $\mathcal{B}_{10}$ wins the symmetric encryption game with greater than ϵ probability. □

$\mathcal{H}_{11} - \mathcal{H}_{13}$: Finally, we revert the changes made in $\mathcal{H}_1 - \mathcal{H}_3$. Here, the indistinguishability between $\mathcal{H}_{11} - \mathcal{H}_{13}$ follows analogously to that of $\mathcal{H}_1 - \mathcal{H}_3$.
$\mathcal{H}_{\mathsf{conf}}$: The garbling step circuit is in dummy mode.

This concludes the proof of Lemma 4. ■

Proof of Theorem 3. The sequence of hybrids shown in the proof of Lemma 3 and Lemma 4 are reversible, and imply an inverse of rule $\mathfrak{A}$ and rule $\mathfrak{B}$. Thus, the proof of Theorem 3 follows directly from the proofs of Lemma 3 and Lemma 4. ■

3.4 Removing the Output Dependency

We note that our whilst our construction [Fig. 4] outputs only one bit, a generic transformation can be used to output multiple bits. Depending on the security definition that we want to achieve, there are two generic ways to carry out such a transformation.

Simulation Security. If we insist on simulation security (which is the same definition achieved by the protocol in [Fig. 4]) we can simply hash the circuit Φ as $\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \Phi)$, where Φ takes as input an m_B and an index i and returns the i-th output bit of $\mathbf{C}(x)_i$. Then, for all output bits we let the sender compute

$$\mathsf{c} := \left(c_1 \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (x, 1)), \ldots, c_{|y|} \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (x, |y|))\right)$$

where $|y|$ denotes the output size. The reciever can then recover the output bit-by-bit. Security follows from a standard hybrid argument.

Indistinguishability. If we relax the requirements to indistinguishability-based security, then it becomes possible to remove the output dependency entirely. Specifically, we require that $\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, x)$ and $\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, \bar{x})$ are computationally indistinguishable, for pairs $(x, \bar{x})$ such that $\mathbf{C}(x) = \mathbf{C}(\bar{x})$.

Our scheme proceeds as described above except that the sender does not explicitly compute the ciphertexts $\big(c_1, \ldots, c_{|y|}\big)$, instead the sender obfuscates a circuit that given an index $i \in \{1, \ldots, |y|\}$ returns

$$\mathsf{LFE.Enc}\big(\mathsf{crs}, \mathsf{d}, (x, i); \mathsf{PPRF.Eval}(k, i)\big)$$

where k is the key of a puncturable PRF. To compute the output, the receiver evaluates the obfuscated circuit on all possible indices to recover $\big(c_1, \ldots, c_{|y|}\big)$, then she applies the $\mathsf{LFE.Dec}$ algorithm to recover the output bit-by-bit. Observe that now the size of the ciphertext depends on $|y|$ only logarithmically.

In terms of security, we can show indistinguishabilty by defining $(|y|+1)$-many intermediate distributions, where in the i^*-th distribution $\mathcal{H}_{i^*}$ we obfuscate the circuit that given an index $i \in \{1, \ldots, |y|\}$ returns

$$\begin{array}{ll} \mathsf{LFE.Enc}\big(\mathsf{crs}, \mathsf{d}, (\bar{x}, i); \mathsf{PPRF.Eval}(k, i)\big), & \text{if } i < i^* \\ \mathsf{LFE.Enc}\big(\mathsf{crs}, \mathsf{d}, (x, i); \mathsf{PPRF.Eval}(k, i)\big), & \text{otherwise.} \end{array}$$

Note that $\mathcal{H}_0$ is functionally equivalent to the original obfuscated circuit, whereas $\mathcal{H}_{|y|+1}$ is functionally equivalent to the encryption of $\bar{x}$. Thus, it suffices to show that $\mathcal{H}_{i^*}$ and $\mathcal{H}_{i^*+1}$ are computationally indistinguishable. This is done with help of a five-steps argument:

- First we puncture the PRF key at point i^*, and indistinguishability follows from the security of iO.
- We switch $\mathsf{PPRF.Eval}(k, i^*)$ with a uniform string u, which is indistinguishable by the security of the puncturable PRF.
- We hardwire the output of $\mathsf{c}^* \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (x, i^*); u)$ in the obfuscated circuit. Again, indistinguishability follows from the security of iO.
- We set $\mathsf{c}^* \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (\bar{x}, i^*); u)$. Indistinguishability follows from the security of LFE.
- We undo the modifications done by the first three steps.

Note that the first distribution is identical to $\mathcal{H}_{i^*}$, whereas the latter is identical to $\mathcal{H}_{i^*+1}$.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge New York, 2009. (Cited on page 24.)

[ACC+16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating RAM computations with adaptive soundness and privacy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 3–30, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. (Cited on page 3.)

[AFS19] Prabhanjan Ananth, Xiong Fan, and Elaine Shi. Towards attribute-based encryption for RAMs from LWE: Sub-linear decryption, and more. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 112–141, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany. (Cited on page 2.)

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th Annual Symposium on Foundations of Computer Science*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press. (Cited on page 28.)

[AJJM21] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Unbounded multi-party computation from learning with errors. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 754–781, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. (Cited on page 31.)

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 252–279, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. (Cited on page 3.)

[AR21] Shweta Agrawal and Razvan Rosie. Adaptively secure laconic function evaluation for NC1. E-prints/Working papers: ORBilu, 2021. https://orbilu.uni.lu/handle/10993/46493. (Cited on pages 1, 2, and 6.)

[BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 79–109, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. (Cited on page 2.)

[BDGM22] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: circular-secure LWE suffices. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on page 2.)

[BFK+19] Saikrishna Badrinarayanan, Rex Fernando, Venkata Koppula, Amit Sahai, and Brent Waters. Output compression, MPC, and iO for turing machines. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 342–370, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany. (Cited on page 3.)

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. (Cited on pages 2, 3, and 27.)

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press. (Cited on page 6.)

[BJKL21] Fabrice Benhamouda, Aayush Jain, Ilan Komargodski, and Huijia Lin. Multiparty reusable non-interactive secure computation from LWE. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 724–753, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. (Cited on page 31.)

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 349–378, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany. (Cited on page 31.)

[BP15] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 236–261, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany. (Cited on page 2.)

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 792–821, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. (Cited on page 2.)

[CCC+16] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel RAM from indistinguishability obfuscation. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 179–190, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery. (Cited on page 3.)

[CCHR15] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Succinct adaptive garbled RAM. Cryptology ePrint Archive, Report 2015/1074, 2015. https://eprint.iacr.org/2015/1074. (Cited on page 3.)

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. (Cited on pages 3, 25, and 27.)

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 169–178, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery. (Cited on page 3.)

[DGGM19] Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In David Zuckerman, editor, *60th Annual Symposium on Foundations of Computer Science*, pages 661–685, Baltimore, MD, USA, November 9–12, 2019. IEEE Computer Society Press. (Cited on page 31.)

[DGM23a] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for turing machines. In *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2023. (Cited on page 1.)

[DGM23b] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for turing machines. Cryptology ePrint Archive, Report, 2023. (Cited on page 1.)

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press. (Cited on page 27.)

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 518–535, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on page 2.)

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. (Cited on pages 6 and 29.)

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. (Cited on page 2.)

[GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 515–544, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. (Cited on page 25.)

[GP20] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. Cryptology ePrint Archive, Report 2020/1010, 2020. https://eprint.iacr.org/2020/1010. (Cited on page 2.)

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 156–181, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. (Cited on page 12.)

[GS18] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 425–454, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. (Cited on pages 3, 4, 7, 12, and 27.)

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery. (Cited on pages 2 and 27.)

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003, 2020. https://eprint.iacr.org/2020/1003. (Cited on page 2.)

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb {F}_p$, dlin, and prgs in nc^0. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 670–699. Springer, 2022. (Cited on page 2.)

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual*

ACM Symposium on Theory of Computing, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press. (Cited on pages 3 and 27.)

[KNYY19] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Exploring constructions of compact NIZKs from various assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 639–669, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. (Cited on pages 2 and 31.)

[NRS21] David Naccache, Razvan Rosie, and Lorenzo Spignoli. Post-quantum secure lfe for L/poly with smaller parameters. E-prints/Working papers: ORBilu, 2021. `https://hdl.handle.net/10993/46725`. (Cited on pages 1, 2, and 6.)

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 121–145, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany. (Cited on pages 2 and 27.)

[PCFT20] Bo Pang, Long Chen, Xiong Fan, and Qiang Tang. Multi-input laconic function evaluation. In Joseph K. Liu and Hui Cui, editors, *ACISP 20: 25th Australasian Conference on Information Security and Privacy*, volume 12248 of *Lecture Notes in Computer Science*, pages 369–388, Perth, WA, Australia, November 30 – December 2, 2020. Springer, Heidelberg, Germany. (Cited on page 6.)

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, April 1979. (Cited on page 8.)

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press. (Cited on pages 1, 2, 6, 7, 8, and 31.)

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press. (Cited on page 28.)

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 127–156, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. (Cited on page 2.)

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. (Cited on pages 3 and 28.)

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. (Cited on pages 3 and 28.)

A Cryptographic Preliminaries

A.1 Models of Computation

In this section we recall some computational models, namely the Turing machine and Boolean circuits. We start by recalling the definition of a Turing machine M which is described by a tuple (Γ, Q, δ). Note that for our LFE construction [Sec. 3], we will make use of a 4-tape Turing machine consisting of two input tapes, one work tape, and an output tape.

Definition 4 (Turing Machine [AB09]). *A k-tape Turing Machine (Γ, Q, δ) is defined as the following tuple.*

Γ: *A finite set of symbols that M's tapes can contain, where the tapes can be read only for input, read/write for working, and read/write for output. We assume that Γ contains a blank symbol $\square$ as well as a start symbol $\triangleright$, and the numbers 0 and 1.*

Q: *A finite set of states containing a start state q_{start} and a halting state q_{halt}.*

δ: *A transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^k$ describing the computations done by M.*

Without loss of generality, we can think of a Turing machine M as being deterministic. Next, we introduce the notion of a transcript tr that can be thought of as the *memory* of a machine. Let tr_i denote the input tapes concatenated with the content of M's worktape at the start of step i of M's computation. For a Turing machine M that takes T steps to halt, running M on tr_T gives the output of M.

Definition 5 (Boolean Circuits [AB09]). *$\forall n \in \mathbb{N}$, an n-input, single-output Boolean circuit $\mathbf{C} : \{0,1\}^n \to \{0,1\}$, is a directed acyclic graph with n sources and one sink. All non-source vertices are called gates and are labelled with one of $\wedge$, $\vee$, or $\neg$. The vertices labeled with $\wedge$ and $\vee$ have fan-in 2 and the vertices labeled with $\neg$ have fan-in 1. The size of $\mathbf{C}$, denoted $|\mathbf{C}|$, is the number of vertices in it.*

A.2 Symmetric Encryption

Definition 6 (Symmetric Encryption). *A Symmetric Encryption scheme $\Pi := (\mathsf{Sym.Gen}, \mathsf{Sym.Enc}, \mathsf{Sym.Dec})$ is defined as the following tuple of PPT algorithms.*

$\mathsf{k} \leftarrow \mathsf{Sym.Gen}(1^\lambda)$: *Given the security parameter 1^λ (encoded in unary) the generation algorithm returns a key $\mathsf{k} \in \mathcal{K}$.*

$c \leftarrow \mathsf{Sym.Enc}(\mathsf{k}, x)$: *Given a key k and a message x, the encryption algorithm returns a ciphertext c.*

$x \leftarrow \mathsf{Sym.Dec}(\mathsf{k}, c)$: *Given a key k and a ciphertext c, the decryption algorithm returns a message x.*

We require a symmetric encryption scheme to satisfy the following properties.

Correctness: *For any message $x \in \mathcal{M}$ and any key $\mathsf{k} \in \mathcal{K}$ it holds that,*

$$\Pr\left[\mathsf{Sym.Dec}(\mathsf{k}, c) = x \,\middle|\, \begin{array}{l} \mathsf{k} \leftarrow \mathsf{Sym.Gen}(1^\lambda) \\ c \leftarrow \mathsf{Sym.Enc}(\mathsf{k}, x) \end{array}\right] = 1,$$

where the probability is taken over the random coins of $\mathsf{Sym.Enc}$, and $\mathcal{M}$ and $\mathcal{K}$ denote the message space and key space respectively.

Security: *For any two message $x_0, x_1 \in \mathcal{M}$ and any key $\mathsf{k} \in \mathcal{K}$, it holds that,*

$$\left|\Pr\left[\mathcal{A}(c, x_0, x_1) = 1 \,\middle|\, c \leftarrow \mathsf{Sym.Enc}(\mathsf{k}, x_0)\right] - \Pr\left[\mathcal{A}(c, x_0, x_1) = 1 \,\middle|\, c \leftarrow \mathsf{Sym.Enc}(\mathsf{k}, x_1)\right]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of $\mathsf{Sym.Gen}, \mathsf{Sym.Enc}$, and $\mathcal{A}$, and $\mathcal{M}$ and $\mathcal{K}$ denote the message space and key space respectively.

A.3 Updatable Laconic Oblivious Transfer

We recall the definition of Updatable Laconic Oblivious Transfer (ULOT) [CDG+17]. Informally, ULOT allows a receiver to commit to a large input D via a short message. Then, a single short message by a sender enables the receiver to learn $m_{D[L]}$, where the messages m_0, m_1 and the location $L \in [|D|]$ are dynamically chosen by the sender. Similar to the work of [GOS18], we operate on blocks of data as opposed to bits. Furthermore, our definition combines the $\mathsf{ULOT.Send}$ and $\mathsf{ULOT.SendWrite}$ algorithms, as well as the $\mathsf{ULOT.Receive}$ and $\mathsf{ULOT.ReceiveWrite}$ algorithms. Intuitively, sender privacy does not reveal anything about the message being sent in the transfer. A more formal definition follows.

Definition 7 (Updatable Laconic Oblivious Transfer [CDG+17]). *An Updatable Laconic Oblivious Transfer is defined as the following tuple of* PPT *algorithms.* $\mathsf{ULOT} := (\mathsf{ULOT.Gen}, \mathsf{ULOT.Hash}, \mathsf{ULOT.Send}, \mathsf{ULOT.Receive}, , \mathsf{ULOT.SendWriteReadULOT.ReceiveWriteRead})$.

$\mathsf{crs} \leftarrow \mathsf{ULOT.Gen}\left(1^\lambda, 1^N\right)$**:** *Given the security parameter 1^λ and the block size 1^N, the generation algorithm returns a common reference string* crs.

$\left(\mathsf{d}, \widehat{D}\right) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D)$**:** *Given the common reference string* crs *and a database $D \in \left\{\{0,1\}^N\right\}^*$, the compression algorithm returns a digest* d *and a state $\widehat{D}$.*

$e \leftarrow \mathsf{ULOT.Send}\left(\mathsf{crs}, \mathsf{d}, L, \{m_{i,0}, m_{i,1}\}_{i\in[N]}\right)$**:** *Given the common reference string* crs*, a digest* d*, a read location $L \in \mathbb{N}$, and a set of messages $\{m_{i,0}, m_{i,1}\}_{i\in[N]}$ with $m_{i,0}, m_{i,1} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ for every $i \in [N]$, the send algorithm returns a ciphertext e.*

$(m_1, \ldots, m_N) \leftarrow \mathsf{ULOT.Receive}^{\widehat{D}}(\mathsf{crs}, e, L)$**:** *Given the common reference string* crs*, a ciphertext e, and a read location $L \in \mathbb{N}$, the receive algorithm (a RAM algorithm with random read access to $\widehat{D}$) returns a set of messages $(m_1, \ldots, m_N)$.*

$e \leftarrow \mathsf{ULOT.SendWriteRead}\Big(\mathsf{crs}, \mathsf{d}, L_w, \{b_i\}_{i\in[N]}, \{\mu_{j,0}, \mu_{j,1}\}_{j=1}^{|\mathsf{d}|}, L_r, \{m_{k,0}, m_{k,1}\}_{k\in[N]}\Big)$**:** *Given the common reference string* crs*, a digest* d*, a write location $L_w \in \mathbb{N}$, a set of bits $\{b_i\}_{i\in[N]}$ to be written with $b_i \in \{0,1\}$ for every $i \in [N]$, $|\mathsf{d}|$ many pairs of messages $\{\mu_{j,0}, \mu_{j,1}\}_{j=1}^{|\mathsf{d}|}$, where every $\mu_{j,0}$ and $\mu_{j,1}$ is of length* $\mathsf{poly}(\lambda)$*, a read location $L_r \in \mathbb{N}$, and a set of messages $\{m_{k,0}, m_{k,1}\}_{k\in[N]}$ with $m_{k,0}, m_{k,1} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ for every $k \in [N]$, the algorithm returns a ciphertext e. In other words, the algorithm runs* $\mathsf{ULOT.SendWrite}(\cdot)$ *followed by* $\mathsf{ULOT.Send}(\cdot)$ *from [GOS18].*

$\left(\{\mu_j\}_{j=1}^{|\mathsf{d}|}, (m_1, \ldots, m_N)\right) \leftarrow \mathsf{ULOT.ReceiveWriteRead}^{\widehat{D}}\left(\mathsf{crs}, L_w, \{b_i\}_{i\in[N]}, e, L_r\right)$**:** *Given the common reference string* crs*, a write location $L_w \in \mathbb{N}$, a set of bits $\{b_i\}_{i\in[N]}$ with $b_i \in \{0,1\}$ for every $i \in [N]$, a ciphertext e, and a read location $L_r \in \mathbb{N}$, the algorithm (a RAM algorithm with random read/write access to $\widehat{D}$) updates the database D, such that $D[L_w] = b_1 \ldots b_N$, and returns a set of messages $\{\mu_j\}_{j=1}^{|\mathsf{d}|}$, as well as a set of messages $(m_1, \ldots, m_N)$. In other words, the algorithm runs* $\mathsf{ULOT.ReceiveWrite}^{\widehat{D}}(\cdot)$ *followed by* $\mathsf{ULOT.Receive}^{\widehat{D}}(\cdot)$ *from [GOS18].*

We require an updatable laconic oblivious transfer to satisfy the following properties.

Correctness: *For any database D of size at most $M =$ $\mathsf{poly}(\lambda)$, any memory location $L \in [M]$, any set of messages $(m_{i,0}, m_{i,1}) \in \{0,1\}^{\mathsf{poly}(\lambda)}$ where $i \in [N]$ it holds that,*

$$\Pr\left[\begin{array}{l} \forall i \in [N], \\ \quad m_i = m_{i,D[L,i]} \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{ULOT.Gen}(1^\lambda, 1^N) \\ (\mathsf{d}, \widehat{D}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D) \\ e \leftarrow \mathsf{ULOT.Send}\left(\mathsf{crs}, \mathsf{d}, L, \{m_{i,0}, m_{i,1}\}_{i\in[N]}\right) \\ (m_1, \ldots, m_N) \leftarrow \mathsf{ULOT.Receive}^{\widehat{D}}(\mathsf{crs}, e, L) \end{array}\right] = 1,$$

where $D[L, i]$ denotes the i^{th} bit in the L^{th} block of D and the probability is taken over the random coins of $\mathsf{ULOT.Gen}, \mathsf{ULOT.Send}$ *and* $\mathsf{ULOT.Receive}$.

Sender Privacy: *There exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{ULOT.S}}$ *such that for any non-uniform* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that,*

$$\left|\Pr\left[\mathsf{SenPriExpt}^{\mathsf{real}}\left(1^\lambda, \mathcal{A}\right) = 1\right] - \Pr\left[\mathsf{SenPriExpt}^{\mathsf{sim}}\left(1^\lambda, \mathcal{A}\right) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{SenPriExpt}^{\mathsf{real}}$ *and* $\mathsf{SenPriExpt}^{\mathsf{sim}}$ *are described in Figure 9.*

$\mathsf{SenPriExpt}^{\mathsf{real}}\left(1^\lambda, \mathcal{A}\right)$	$\mathsf{SenPriExpt}^{\mathsf{sim}}\left(1^\lambda, \mathcal{A}\right)$
1 : $\mathsf{crs} \leftarrow \mathsf{ULOT.Gen}(1^\lambda, 1^N)$	1 : $\mathsf{crs} \leftarrow \mathsf{ULOT.Gen}(1^\lambda, 1^N)$
2 : $\left(D, L, \{m_{i,0}, m_{i,1}\}_{i\in[N]}, \mathsf{st}\right) \leftarrow \mathcal{A}_1(\mathsf{crs})$	2 : $\left(D, L, \{m_{i,0}, m_{i,1}\}_{i\in[N]}, \mathsf{st}\right) \leftarrow \mathcal{A}_1(\mathsf{crs})$
3 : $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D)$	3 : $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D)$
4 : $e \leftarrow \mathsf{ULOT.Send}\left(\mathsf{crs}, \mathsf{d}, L, \{m_{i,0}, m_{i,1}\}_{i\in[N]}\right)$	4 : $e \leftarrow \mathsf{Sim}_{\mathsf{ULOT.S}}\left(\mathsf{crs}, D, L, \{m_{i,D[L,i]}\}_{i\in[N]}\right)$
5 : **return** $\mathcal{A}_2(\mathsf{st}, e)$	5 : **return** $\mathcal{A}_2(\mathsf{st}, e)$

Figure 9. Sender Privacy Security Game.

Correctness of Writes & Reads: *For any database* D *of size at most* $M = \mathsf{poly}(\lambda)$*, and any memory location* $L_w \in [M]$*, let* D^* *denote the database that is identical to* D *except that* $D^*[L_w, i] = b_i$ *for all* $i \in [N]$ *and some sequence* $\{b_i\} \in \{0,1\}$*, any sequence of messages* $\{\mu_{j,0}, \mu_{j,1}\}_{j\in[|\mathsf{d}|]} \in \{0,1\}^{\mathsf{poly}(\lambda)}$*, any memory location* $L_r \in [M]$*, and any set of messages* $(m_{k,0}, m_{k,1}) \in \{0,1\}^{\mathsf{poly}(\lambda)}$ *where* $k \in [N]$ *it holds that,*

$$\Pr\left[\begin{array}{c|r} & \mathsf{crs} \leftarrow \mathsf{ULOT.Gen}(1^\lambda, 1^N) \\ & (\mathsf{d}, \widehat{D}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D) \\ \forall j \in [|\mathsf{d}|], & (\mathsf{d}^*, \widehat{D}^*) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D^*) \\ \mu'_j = \mu_{j,\mathsf{d}^*_j} & e \leftarrow \mathsf{ULOT.SWR}\Big(\mathsf{crs}, \mathsf{d},\, L_w, \{b_i\}_{i\in[N]}, \\ \bigwedge & \{\mu_{j,0}, \mu_{j,1}\}_{j=1}^{|\mathsf{d}|}\,, L_r, \{m_{k,0}, m_{k,1}\}_{k\in[N]}\Big) \\ \forall k \in [N], & \left(\{\mu'_j\}_{j=1}^{|\mathsf{d}|}, (m_1, \ldots, m_N)\right) \leftarrow \mathsf{ULOT.RWR}^{\widehat{D}}\Big(\mathsf{crs}, \\ m_k = m_{k,D^*[L_r,k]} & L_w, \{b_i\}_{i\in[N]}, e, L_r\Big) \end{array}\right] = 1,$$

where $D^*[L_r, k]$ *denotes the* k^{th} *bit in the* ${L_r}^{th}$ *block of* D^* *and the probability is taken over the random coins of* $\mathsf{ULOT.Gen}, \mathsf{ULOT.SWR}$ *and* $\mathsf{ULOT.RWR}$[6].

Sender Privacy for Writes & Reads: *There exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{ULOT.WR}}$ *such that for any non-uniform* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that,*

$$\left|\Pr\Big[\mathsf{WriReaSenPriExpt}^{\mathsf{real}}\left(1^\lambda, \mathcal{A}\right) = 1\Big] - \Pr\Big[\mathsf{WriReaSenPriExpt}^{\mathsf{sim}}\left(1^\lambda, \mathcal{A}\right) = 1\Big]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{WriReaSenPriExpt}^{\mathsf{real}}$ *and* $\mathsf{WriReaSenPriExpt}^{\mathsf{sim}}$ *are described in Figure 10.*

Efficiency: *The algorithm* $\mathsf{ULOT.Gen}$ *runs in* $\mathsf{poly}(\lambda)$ *time, and* $\mathsf{ULOT.Hash}$ *runs in* $|D|\mathsf{poly}(\log(|D|), \lambda)$*.* $\mathsf{ULOT.Send}$*,* $\mathsf{ULOT.Receive}$*,* $\mathsf{ULOT.SendWriteRead}$*,* $\mathsf{ULOT.ReceiveWriteRead}$ *all run in* $\mathsf{poly}(\log(|D|), \lambda)$ *time.*

[6] Here, for the sake of improved readability we abbreviate $\mathsf{ULOT.SendWriteRead}$ and $\mathsf{ULOT.ReceiveWriteRead}$ to $\mathsf{ULOT.SWR}$ and $\mathsf{ULOT.RWR}$.

$\mathsf{WriReaSenPriExpt}^{\mathsf{real}}(1^\lambda, \mathcal{A})$

1 : $\mathsf{crs} \leftarrow \mathsf{ULOT.Gen}(1^\lambda, 1^N)$

2 : $\Big(D, L_w, \{b_i\}_{i\in[N]}, \{\mu_{j,0}, \mu_{j,1}\}_{[\lambda]}, L_r, \{m_{k,0}, m_{k,1}\}_{k\in[N]}, \mathsf{st}\Big) \leftarrow \mathcal{A}_1(\mathsf{crs})$

3 : $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D)$

4 : $e \leftarrow \mathsf{ULOT.SendWriteRead}\Big(\mathsf{crs}, \mathsf{d}, L_w, \{b_i\}_{i\in[N]}, \{\mu_{j,0}, \mu_{j,1}\}_{j=1}^{|\mathsf{d}|}, L_r,$
$\{m_{k,0}, m_{k,1}\}_{k\in[N]}\Big)$

5 : **return** $\mathcal{A}_2(\mathsf{st}, e)$

$\mathsf{WriReaSenPriExpt}^{\mathsf{sim}}(1^\lambda, \mathcal{A})$

1 : $\mathsf{crs} \leftarrow \mathsf{ULOT.Gen}(1^\lambda, 1^N)$

2 : $\Big(D, L_w, \{b_i\}_{i\in[N]}, \{\mu_{j,0}, \mu_{j,1}\}_{[\lambda]}, L_r, \{m_{k,0}, m_{k,1}\}_{k\in[N]}, \mathsf{st}\Big) \leftarrow \mathcal{A}_1(\mathsf{crs})$

3 : $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D)$

4 : $(\mathsf{d}^*, \widehat{D}^*) \leftarrow \mathsf{ULOT.Hash}(\mathsf{crs}, D^*)$, where D^* denotes the database that is identical to D except that $D^*[L_w, i] = b_i$ for all $i \in [N]$.

5 : $e \leftarrow \mathsf{Sim}_{\mathsf{ULOT.WR}}\Big(\mathsf{crs}, D, L_w, \{b_i\}_{i\in[N]}, \{\mu_{j,\mathsf{d}^*_j}\}_{j\in[\lambda]}, L_r, \{m_{k,D^*[L_r,k]}\}_{k\in[N]}\Big)$

return $\mathcal{A}_2(\mathsf{st}, e)$

Figure 10. Sender Privacy Security Game for Writes & Reads.

Theorem 4 ([CDG+17]). *Assuming* iO *for circuits and somewhere statistically binding hash functions [HW15, KLW15, OPWW15], there exists a construction of an updatable laconic oblivious transfer.*

In our LFE construction we will need to write to an index that exceeds the current size of the database, i.e., the database should stretch dynamically. However, the construction of Updatable Laconic Oblivious Transfer presented in [CDG+17] requires the contents of the entire database to be specified in advance. We observe that there exist methods for allowing *out of bound* writes to the database, and we refer the reader to the works of [GS18] and [OPWW15] for details.

A.4 Indistinguishability Obfuscation

Next we recall the definition of Indistinguishability Obfuscation (iO) [BGI+01, GGH+13]. Informally, two circuits are said to be functionally equivalent if they return the same result when evaluated on the same input. Given two functionally equivalent circuits, their obfuscations are computationally indistinguishable.

Definition 8 (Indistinguishability Obfuscation). *An indistinguishability obfuscator* (iO) *for a family of circuits* $\{\mathcal{C}_\lambda\}_\lambda$ *is defined as the following* PPT *algorithm.*

$\mathsf{ob}\mathbf{C} \leftarrow \mathsf{iO}(\mathbf{C})$**:** *Given a circuit* $\mathbf{C} \in \mathcal{C}_\lambda$*, the obfuscation algorithm returns an obfuscated circuit* $\mathsf{ob}\mathbf{C}$*.*

We require an indistinguishability obfuscator to satisfy the following properties.

Correctness: *For all* $\lambda \in \mathbb{N}$ *and for all* $\mathbf{C} \in \mathcal{C}_\lambda$ *and for all* x*,*

$$\Pr[\mathsf{iO}(\mathbf{C})(x) = \mathbf{C}(x)] = 1,$$

where the probability is taken over the random coins of iO*.*

Security (iO-sec): *For all* $\mathbf{C}_0, \mathbf{C}_1 \in \mathcal{C}_\lambda$ *such that for all* x, $\mathbf{C}_0(x) = \mathbf{C}_1(x) \wedge |\mathbf{C}_0| = |\mathbf{C}_1|$ *and for all polynomial sized adversaries* $\mathcal{A}$,

$$|\Pr[\mathcal{A}(\mathsf{iO}(\mathbf{C}_0)) = 1] - \Pr[\mathcal{A}(\mathsf{iO}(\mathbf{C}_1)) = 1]| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of iO.

A.5 Garbled Circuits

Informally, Garbled Circuits allow two parties to jointly evaluate a function, without introducing the need for a trusted third party or revealing each other's inputs. We recall the definition of Garbled Circuits [Yao82, Yao86, AIK04].

Definition 9 (Garbled Circuits). *A garbling scheme* $\mathsf{GC} := (\mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Input})$ *for circuits is defined as the following tuple of* PPT *algorithms.*

$\widetilde{\mathsf{C}} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathbf{C}, \{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}\right)$**:** *Given the security parameter* 1^λ, *a circuit* $\mathbf{C}$, *and input labels* $\mathsf{lbs}_{w,b}$ *(where* $w \in N$ *and* N *is the set of input wires to* $\mathbf{C}$ *and* $b \in \{0,1\}$*), the garbling algorithm returns a garbled circuit* $\widetilde{\mathsf{C}}$. *We assume that for each* w *and* b *the corresponding label* $\mathsf{lbs}_{w,b}$ *is selected uniformly at random from* $\{0,1\}^\lambda$.

$\{\mathsf{lbs}_{w,x_w}\}_{w \in N} \leftarrow \mathsf{GC.Input}\left(\{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}, x\right)$**:** *Given a set of pairs of labels* $\{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}$ *and an input* $x \in \{0,1\}^N$, *the algorithm selects the label for the* i^{th} *pair based on the value of* x_i *for all* $i \in |x|$, *and returns the input labels* $\{\mathsf{lbs}_{w,x_w}\}_{w \in N}$.

$y \leftarrow \mathsf{GC.Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{lbs}_{w,x_w}\}_{w \in N}\right)$**:** *Given a garbled circuit* $\widetilde{\mathsf{C}}$ *and garbled input labels* $\{\mathsf{lbs}_{w,x_w}\}_{w \in N}$, $\mathsf{GC.Eval}$ *returns a string* y.

Definition 10 (Correctness). *A garbling scheme* $\mathsf{GC} := (\mathsf{GC.Garble}, \mathsf{GC.Input}, \mathsf{GC.Eval})$ *is correct if for all* $\mathbf{C} \in \mathbf{C}_\lambda$, *for all inputs* $x \in \{0,1\}^{|N|}$, *and input labels* $\{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}$,

$$\Pr\left[\mathbf{C}(x) = \mathsf{GC.Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{lbs}_{w,x_w}\}_{w \in N}\right) \middle| \begin{array}{l} \widetilde{\mathsf{C}} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathbf{C}, \{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}\right) \\ \{\mathsf{lbs}_{w,x_w}\}_{w \in N} \leftarrow \mathsf{GC.Input}\left(\{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}, x\right) \end{array}\right] = 1,$$

where the probability is taken over the random coins of $\mathsf{GC.Garble}$.

Definition 11 (Selective Security (GC-sec)). *There exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{GC}}$ *such that for any polynomial time adversary* $\mathcal{A}$, *any circuit* $\mathbf{C} \in \mathbf{C}_\lambda$ *and input* $x \in \{0,1\}^{|N|}$,

$$\left| \Pr\left[\mathcal{A}\left(\widetilde{\mathsf{C}}, \{\mathsf{lbs}_{w,x_w}\}_{w \in N}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|\mathbf{C}|}, \mathbf{C}(x), \{\mathsf{lbs}_{w,x_w}\}_{w \in N}\right), \{\mathsf{lbs}_{w,x_w}\}_{w \in N}\right) = 1\right] \right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of $\mathsf{GC.Garble}$, $\widetilde{\mathsf{C}} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathbf{C}, \{\mathsf{lbs}_{w,b}\}_{w \in N, b \in \{0,1\}}\right)$, *and for each* $w \in N$ *and* $b \in \{0,1\}$ *the label* $\mathsf{lbs}_{w,b}$ *is selected uniformly at random from* $\{0,1\}^\lambda$.

A.6 Puncturable Pseudorandom Functions

Informally, a Puncturable Pseudorandom Function (PPRF) [SW14] must be efficiently computable and maintain its functionality after being punctured. It also remains pseudorandom at the punctured point.

Definition 12 (Puncturable Pseudorandom Function). *A puncturable pseudorandom function* $\mathsf{PPRF} := (\mathsf{PPRF.Gen}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Punc})$ *is defined as the following tuple of* PPT *algorithms.*

$\mathsf{PPRF.Gen}(1^\lambda)$**:** *Given the security parameter* 1^λ, *the generation algorithm returns a key* k.

$\mathsf{PPRF.Eval}(\mathsf{k}, x)$**:** *Given a key* $\mathsf{k} \in \{0,1\}^\lambda$ *and given an input* $x \in \{0,1\}^\lambda$*, the pseudo random function returns a pseudo random output* $R \in \{0,1\}^\lambda$*.*[7]

$\mathsf{PPRF.Punc}(\mathsf{k}, y)$**:** *Given a key* $\mathsf{k} \in \{0,1\}^\lambda$ *and a point* $y \in \{0,1\}^\lambda$*, the puncturing algorithm returns the punctured key at point* y*, denoted* $\mathsf{k}\{y\}$*.*

We require a puncturable pseudorandom function to satisfy the following properties.

Functionality: *For all* $\lambda \in \mathbb{N}$ *and for all* $y \in \{0,1\}^\lambda$*, and for all* $x \neq y$*,*

$$\Pr\left[\mathsf{PPRF.Eval}(\mathsf{k}\{y\}, x) = \mathsf{PPRF.Eval}(\mathsf{k}, x) \,\middle|\, \begin{array}{c} \mathsf{k} \leftarrow \mathsf{PPRF.Gen}(1^\lambda) \\ \mathsf{k}\{y\} \leftarrow \mathsf{PPRF.Punc}(\mathsf{k}, y) \end{array}\right] = 1,$$

where the probability is taken over the random coins of $\mathsf{PPRF.Gen}$ *and* $\mathsf{PPRF.Punc}$*.*

Pseudorandomness ($\mathsf{PPRF\text{-}rand}$)**:** *For all* $\lambda \in \mathbb{N}$*, for all* $y \in \{0,1\}^\lambda$*, and for all polynomial sized adversaries* $\mathcal{A}$*,*

$$\left| \Pr\left[\mathcal{A}(\mathsf{PPRF.Eval}(\mathsf{k}, y), \mathsf{k}\{y\}) = 1 \,\middle|\, \begin{array}{c} \mathsf{k} \leftarrow \mathsf{PPRF.Gen}(1^\lambda) \\ \mathsf{k}\{y\} \leftarrow \mathsf{PPRF.Punc}(\mathsf{k}, y) \end{array}\right] \right.$$
$$\left. - \Pr\left[\mathcal{A}(u, \mathsf{k}\{y\}) = 1 \,\middle|\, \begin{array}{c} \mathsf{k} \leftarrow \mathsf{PPRF.Gen}(1^\lambda) \\ u \leftarrow\!\!\$\, U_\lambda \\ \mathsf{k}\{y\} \leftarrow \mathsf{PPRF.Punc}(\mathsf{k}, y) \end{array}\right] \right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of $\mathcal{A}$*,* $\mathsf{PPRF.Gen}$*, and* $\mathsf{PPRF.Punc}$*; and the random choices of* u*, where* U_λ *denotes the uniform distribution over* $\{0,1\}^\lambda$*.*

B Applications

In the following section we present some new implications of our LFE construction.

B.1 Witness Encryption for Turing Machines

We show how to obtain a witness encryption scheme where the ciphertext size is independent of the size of (the circuit representation of) the underlying NP relation $\mathcal{R}$ and the decryption runtime is only proportional to the runtime of the Turing machine computing $\mathcal{R}$.

Definition. We recall the definition of witness encryption [GGSW13] and establish some notation.

Definition 13 (Witness Encryption). *A witness encryption scheme* $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$ *for an* NP *language* $\mathcal{L}$ *with relation* $\mathcal{R}$*, is defined as the following tuple of* PPT *algorithms.*

$c \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m)$**:** *Given the security parameter* 1^λ*, a statement* x*, and a message* m*, the encryption algorithm returns a ciphertext* c*.*

$m \leftarrow \mathsf{WE.Dec}(w, c)$**:** *Given a witness* w *and a ciphertext* c*, the decryption algorithm returns a message* m*.*

We require a witness encryption scheme to satisfy the following properties.

Correctness: *For any message* m*, any statement* $x \in \mathcal{L}$*, and any witness* $w \in \mathcal{R}(x, w)$ *it holds that,*

$$\mathsf{WE.Dec}(w, \mathsf{WE.Enc}(1^\lambda, x, m)) = m.$$

Security: *For any two message* (m_0, m_1) *and any* $x \notin \mathcal{L}$ *it holds that,*

$$\left| \Pr\left[\mathcal{A}(c) = 1 \,\middle|\, c \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m_0)\right] - \Pr\left[\mathcal{A}(c) = 1 \,\middle|\, c \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m_1)\right] \right| \leq \mathsf{negl}(\lambda).$$

[7] Note that k can also be punctured at at point y (denoted $\mathsf{k}\{y\}$), in which case $\mathsf{PPRF.Eval}(\mathsf{k}\{y\}, x)$ is called.

Construction. Given an LFE scheme $\mathsf{LFE} := (\mathsf{LFE.Gen}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$, an indistinguishability obfuscator iO, and a puncturable PRF $\mathsf{PPRF} := (\mathsf{PPRF.Gen}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Punc})$ (all with sub-exponential security) our scheme is defined as follows. The encryption algorithm ($\mathsf{WE.Enc}$) samples a key $\mathsf{k} \leftarrow \mathsf{PPRF.Gen}(1^\lambda)$ a $\mathsf{crs} \leftarrow \mathsf{LFE.Gen}\left(1^\lambda, 1^N\right)$ and computes the hash of $\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathbf{M}_\mathcal{R})$ where

$$\mathbf{M}_\mathcal{R}(m, w) : \textbf{if } \mathcal{R}(x, w) = 1 \textbf{ return } m \textbf{ else return } \bot.$$

The algorithm further defines the circuit $\mathbf{C}_{x,m}$ as

$$\mathbf{C}_{x,m}(w) : \textbf{return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m, w); \mathsf{PPRF.Eval}(\mathsf{k}, w))$$

and returns $(\mathsf{crs}, \mathsf{ob}\mathbf{C} \leftarrow \mathsf{iO}(\mathbf{C}_{x,m}))$. Given a witness w, the decryption algorithm ($\mathsf{WE.Dec}$) recovers the message by computing

$$\begin{aligned}
\mathsf{LFE.Dec}(\mathsf{crs}, \mathbf{M}_\mathcal{R}, \mathsf{ob}\mathbf{C}(w)) &= \mathsf{LFE.Dec}(\mathsf{crs}, \mathbf{M}_\mathcal{R}, \mathbf{C}_{x,m}(w)) \\
&= \mathsf{LFE.Dec}(\mathsf{crs}, \mathbf{M}_\mathcal{R}, \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m, w))) \\
&= \mathbf{M}_\mathcal{R}(m, w) \\
&= m.
\end{aligned}$$

Note that the size of the ciphertext is only dependent on the size of the witness $\omega = |w|$, the size of the message $|m|$, and the security parameter. Furthermore, the runtime of the decryption algorithm only depends on the runtime of the Turing machine computing $\mathbf{M}_\mathcal{R}$.

Security. The security of the scheme follows by a standard hybrid argument over all possible witnesses $w \in \{0,1\}^\omega$. For all $i = 0 \dots 2^\omega$, the ciphertext in hybrid $\mathcal{H}_i$ is computed as the obfuscation of the following circuit:

$$\begin{aligned}
\mathbf{C}_i(w) : &\textbf{if } w < i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_1, w); \mathsf{PPRF.Eval}(\mathsf{k}, w)) \\
&\textbf{else return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}, w)).
\end{aligned}$$

Note that in hybrid $\mathcal{H}_0$ the circuit $\mathbf{C}_0$ is functionally identical to an encryption of m_0, whereas in $\mathcal{H}_{2^\omega}$, the obfuscated circuit $\mathbf{C}_{2^\omega}$ is functionally equivalent to an encryption of m_1. By the sub-exponential security of iO, it follows that the resulting ciphertexts are computationally indistinguishable. To complete the analysis, it suffices to show that, for all $i = 0 \dots 2^\omega - 1$, hybrids $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ are computationally indistinguishable. This is done via a puncturing argument that we detail below.

$\mathcal{H}_0$**:** This is identical to $\mathcal{H}_i$.

$\mathcal{H}_1$**:** Here we redefine the obfuscated circuit as

$$\begin{aligned}
\mathbf{C}_i(w) : &\textbf{if } w < i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_1, w); \mathsf{PPRF.Eval}(\mathsf{k}, w)) \\
&\textbf{elseif } w = i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}, w)) \\
&\textbf{else return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}, w)).
\end{aligned}$$

Indistinguishability follows from the sub-exponential security of iO, since the circuits are functionally equivalent.

$\mathcal{H}_2$**:** Here we compute $\mathsf{k}\{i\} \leftarrow \mathsf{PPRF.Punc}(\mathsf{k}, i)$ and we define the obfuscated circuit as

$$\begin{aligned}
\mathbf{C}_i(w) : &\textbf{if } w < i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_1, w); \mathsf{PPRF.Eval}(\mathsf{k}\{i\}, w)) \\
&\textbf{elseif } w = i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}, w)) \\
&\textbf{else return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}\{i\}, w)).
\end{aligned}$$

Indistinguishability follows from the sub-exponential security of iO, since the circuits are functionally equivalent.

$\mathcal{H}_3$**:** Here we redefine the obfuscated circuit as

$$\begin{aligned}\mathbf{C}_i(w) : &\textbf{if } w < i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_1, w); \mathsf{PPRF.Eval}(\mathsf{k}\{i\}, w))\\ &\textbf{elseif } w = i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); r_i)\\ &\textbf{else return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}\{i\}, w))\end{aligned}$$

where $r_i \leftarrow \{0,1\}^\lambda$ is uniformly sampled. Indistinguishability follows from the sub-exponential security of the puncturable PRF.

$\mathcal{H}_4$**:** Here we redefine the obfuscated circuit as

$$\begin{aligned}\mathbf{C}_i(w) : &\textbf{if } w < i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_1, w); \mathsf{PPRF.Eval}(\mathsf{k}\{i\}, w))\\ &\textbf{elseif } w = i \textbf{ return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_1, w); r_i)\\ &\textbf{else return } \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, (m_0, w); \mathsf{PPRF.Eval}(\mathsf{k}\{i\}, w))\end{aligned}$$

and security follows from the sub-exponential sender privacy of the LFE scheme.

$\mathcal{H}_5 - \mathcal{H}_7$**:** We undo the modifications done in $\mathcal{H}_3 \dots \mathcal{H}_1$.

The proof is concluded by observing that the last hybrid is identical to $\mathcal{H}_{i+1}$.

B.2 Corollaries for Prior Work

The following new implications follow by plugging our new LFE construction into existing generic compilers. Due to the essentially optimal parameters, we obtain a large spectrum of new feasibility results.

Computation Complexity of 2PC/MPC. Using the compiler presented in [QWW18], we can use our LFE scheme to bring the communication complexity and the online computation complexity of 2PC/MPC protocols to be independent of the circuit size, without increasing the round complexity of the protocol. In a nutshell, the parties use the underlying 2PC/MPC protocol to compute $\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, x_1 \| \dots \| x_n)$, where $(x_1, \dots, x_n)$ are the inputs of the n parties and the digest is the hash of the circuit $\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathbf{C})$. This is particularly interesting for the case of multi-party reusable non-interactive secure computation (mrNISC) [BL20], for which a protocol with optimal communication complexity was previously unknown.

Corollary 1 ([BL20, BJKL21, AJJM21]). *Assuming indistinguishability obfuscation and* $\{SXDH, LWE\}$ *there exists a mrNISC protocol with communication complexity* $\mathsf{poly}(\lambda, |x_1|, \dots, |x_n|)$.

Reverse Delegation. A recent work [DGGM19] shows how to add malicious security to LFE from standard cryptographic assumptions. As a main application, they obtain a "reverse delegation" scheme where a client can delegate the computation of a large circuit $\mathbf{C}$ to a server on some private input x, where the server is in the end receiving the output $\mathbf{C}(x)$. Security also holds in the presence of malicious parties. Using our scheme, we overcome the client's depth dependency of previous schemes, which significantly restricted the applicability of the protocol.

Corollary 2 ([DGGM19]). *Assuming indistinguishability obfuscation and* $\{DDH, LWE\}$ *there exists a reverse delegation protocol with communication complexity and client's computational complexity of* $\mathsf{poly}(\lambda, |x|)$.

NIZK with Optimal Prover Complexity. A recent work [KNYY19] shows that LFE gives rise to a non-interactive zero-knowledge proof system which optimises the prover's work. Loosely speaking, this is done by letting the prover compute $\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{d}, w)$, where w is the witness and d is the hash of the circuit that computes the NIZK proof algorithm $\mathsf{d} \leftarrow \mathsf{LFE.Hash}(\mathsf{crs}, \mathsf{NIZK.Prove})$. Using our scheme, we can obtain a NIZK with optimal prover complexity.

Corollary 3 ([KNYY19]). *Assuming indistinguishability obfuscation, somewhere statistically binding hash functions, and a (standard) NIZK scheme for* NP *there exists a NIZK scheme for* NP *where the prover's computational complexity is* $\mathsf{poly}(\lambda, |w|)$.