On the Optimal Succinctness and Efficiency of Functional Encryption and Attribute-Based Encryption

Aayush Jain[1] Huijia Lin[2] Ji Luo[2]

[1] Carnegie Mellon University
aayushja@andrew.cmu.edu

[2] University of Washington
{rachel,luoji}@cs.washington.edu

October 2022

Abstract

In this work we investigate the asymptotically best-possible succinctness and efficiency of functional encryption (FE) and attribute-based encryption (ABE), focusing on simultaneously minimizing the sizes of secret keys and ciphertexts and the decryption time. To this end, we consider the notion of partially hiding functional encryption (PHFE) that captures both FE and ABE, and the most efficient computation model of random access machine (RAM). A PHFE secret key sk_f is tied to a function f, whereas a ciphertext $\mathsf{ct}_x(y)$ is tied to a public input x (e.g., ABE attribute) and encrypts a private input y. Decryption reveals $f(x, y)$ and nothing else about y.

We present the first PHFE scheme for RAMs based on the necessary assumption of FE for circuits. It achieves nearly optimal succinctness and efficiency:

- The secret keys sk_f are of (optimal) *constant size*, independent of the description size $|f|$ of the function tied to it.
- The ciphertexts $\mathsf{ct}_x(y)$ have (nearly optimal) *rate-2* dependency on the private input length $|y|$ and (optimal) independency of the public input length $|x|$.
- Decryption is efficient, running in time linear in the instance running time T of the RAM computation, in addition to the input and function sizes, i.e., $T_{\mathsf{Dec}} = (T + |f| + |x| + |y|)\,\mathrm{poly}(\lambda)$.

Our construction significantly improves upon the asymptotic efficiency of prior schemes. As a corollary, we obtain the first ABE scheme with both constant-size keys and constant-size ciphertexts, and the best-possible decryption time matching an existing lower bound.

We show barriers to further improvement on the asymptotic efficiency of (PH-)FE. We prove the first unconditional space-time trade-off for (PH-)FE. *No* secure (PH-)FE scheme can have both key size and decryption time sublinear in the function size $|f|$, and *no* secure PHFE scheme can have both ciphertext size and decryption time sublinear in the public input length $|x|$. These space-time trade-offs apply even in the simplest selective 1-key 1-ciphertext secret-key setting. Furthermore, we show a conditional barrier towards achieving the optimal decryption time $T_{\mathsf{Dec}} = T\,\mathrm{poly}(\lambda)$ — any such (PH-)FE scheme implies a primitive called secret-key doubly efficient private information retrieval (SK-DE-PIR), for which so far the only known candidates rely on new and non-standard hardness conjectures.

Contents

1 Introduction **1**
- 1.1 Technical Overview . . . 6
- 1.2 Related Works . . . 13

2 Preliminaries **15**
- 2.1 Multi-Tape Random Access Machine . . . 15
- 2.2 Laconic Garbled RAM . . . 17
- 2.3 Partially Hiding Functional Encryption and FE for Circuits . . . 19
- 2.4 Universal RAM and PHFE for RAM . . . 21
- 2.5 Indistinguishability Obfuscation . . . 22
- 2.6 Laconic Oblivious Transfer . . . 23
- 2.7 Garbled Circuits . . . 25
- 2.8 Puncturable Pseudorandom Function . . . 26
- 2.9 Secret-Key Encryption . . . 26
- 2.10 Oblivious RAM . . . 27

3 Efficiency Trade-Offs of PHFE for RAM **29**
- 3.1 Contention Between Storage Overhead and Decryption Time . . . 29
- 3.2 Barrier to Time Optimality . . . 33

4 Bounded LGRAM with Fixed-Memory Security **39**
- 4.1 Construction . . . 40
- 4.2 Security . . . 43

5 Transformations of LGRAM **53**
- 5.1 Fixed-Memory to Fixed-Address . . . 53
- 5.2 Fixed-Address to Full Security . . . 57
- 5.3 Bounded to Unbounded . . . 60

6 PHFE for RAM **63**
- 6.1 Bounded Private Input . . . 63
- 6.2 Full-Fledged PHFE for RAM . . . 72

References **75**

1 Introduction

Functional encryption (FE) [BSW11,O'N10] is a powerful enhancement of public-key encryption enabling creation of *functional secret keys* that allow for computing specific functions on encrypted data and reveal only the output. A closely related concept is attribute-based encryption (ABE) [SW05,GPSW06], which supports fine-grained access control. An ABE secret key is tied to a predicate and can only decrypt ciphertexts that are tied to attributes satisfying the predicate. Both FE and ABE have been extensively studied in the literature and have numerous applications.

In this work, we consider the most general notion of partially hiding functional encryption (PHFE) [GVW15,AJL+19,JLMS19] that captures both FE and ABE. A secret key sk_f is tied to a function f, and a ciphertext $\mathsf{ct}_x(y)$ is tied to a public input x in addition to encrypting a private input y, so that decryption produces the computation output $z = f(x, y)$. Collusion-resistant (indistinguishability-based) security ensures that given unboundedly (polynomially) many secret keys $\{\mathsf{sk}_{f_q}\}_q$ tied to different functions, ciphertexts $\mathsf{ct}_x(y_0)$ and $\mathsf{ct}_x(y_1)$ encrypting different private inputs w.r.t. the same public input remain indistinguishable so long as none of the secret keys separate the ciphertexts through outputs, i.e., $f_q(x, y_0) = f_q(x, y_1)$. Put simply, the only information revealed about the private input y is the outputs $\{f_q(x, y)\}_q$. Thanks to its generality and flexibility, advances in PHFE construction immediately imply those in FE, ABE, and their special cases such as IPFE, broadcast encryption, IBE, etc.

Over the past decade, significant progress has been made in establishing the feasibility of FE and PHFE, for various classes of computation, with different levels of efficiency, and from different assumptions. However, we still yet to understand the asymptotic optimality and theoretical limits of the efficiency of (PH-)FE. We ask

Can we construct PHFE with best-possible asymptotic efficiency?
Can we show matching lower bounds?

We make progress towards answering the above questions. For the lower bounds, we show inherent trade-offs between the sizes of keys/ciphertexts and the decryption time, and show barriers towards achieving asymptotically optimal decryption time. From the constructive side, we present the first collusion-resistant PHFE for RAM based on the necessary assumption of (polynomially secure) collusion-resistant FE for circuits, which in turn can be based on well-studied assumptions [JLS21,JLS22]. Our scheme has nearly minimally sized keys and ciphertexts, and the best-possible decryption time matching our lower bounds. As a corollary, we obtain the first ABE with both constant-size keys and constant-size ciphertexts, and the best-possible decryption time matching the recently discovered lower bound [Luo22]. We emphasize again that this work focuses on asymptotic efficiency rather than concrete efficiency.

Dream Efficiency. Before describing our results, we first picture the dream efficiency w.r.t. three important dimensions. Each dimension has been a consistent research theme across many primitives in cryptography.

- *Efficient Computation Model.* Functions should be represented by random access machines (RAM), the most efficient computation model subsuming both circuits and Turing machines.

RAM Computation. It would be flexible to represent a RAM computation by a constant-size RAM machine U[1] with access to multiple tapes, a function tape containing f, a public input tape for x, and a private input tape for y. Moreover, the computation may have *arbitrarily long outputs*.

This model flexibly captures many natural scenarios, e.g., binary search where the database could be a part of f, x, or y. It can emulate the evaluation of a circuit C on input (x, y) by including the circuit description in the function tape. In ciphertext-policy ABE, the ciphertext is tied to a predicate C, which can be captured by putting the description of C on the public input tape. These examples tell us that any of f, x, and y could have a long description and we want to optimize the efficiency dependency on their lengths.

- *Succinctness.* Having small communication- and storage- overhead means having small-sized master public key mpk, secret keys sk_f, and ciphertext $\mathsf{ct}_x(y)$. At the most basic level, the size of each component should be *polynomial* in the length of the information it encodes — $|\mathsf{mpk}| = \mathrm{O}(1)$, $|\mathsf{sk}_f| = \mathrm{poly}(|f|)$, and $|\mathsf{ct}| = \mathrm{poly}(|x|, |y|)$, where $|f|, |x|, |y|$ are the description lengths of f, x, y, respectively — referred to as *polynomial efficiency*.[2] (In this introduction, $\mathrm{O}(\cdot)$ hides $\mathrm{poly}(\lambda)$ factors.) However, there is much to be desired beyond this basic level of efficiency. For instance, linear efficiency means $|\mathsf{sk}_f| = \mathrm{O}(|f|)$ and $|\mathsf{ct}| = \mathrm{O}(|x| + |y|)$, and rate-1 efficiency means $|\mathsf{sk}_f| = |f| + \mathrm{O}(1)$ and $|\mathsf{ct}| = |x| + |y| + \mathrm{O}(1)$.

  In fact, even smaller parameters are possible. Since (PH-)FE does not guarantee the privacy of the function f, it is possible to give a description of f in the clear in the secret key, and the *non-trivial* part of the secret key may be shorter than f. In this case, the right measure of efficiency should be the size of the non-trivial part (i.e., the overhead), which we now view as *the* secret key. We can now aim for secret keys of size *independent* of that of the function — i.e., $|\mathsf{sk}_f| = \mathrm{O}(1)$ — referred to as *constant-size* keys. The same observation applies to the public input x tied to the ciphertext and we can hope for ciphertexts of size *independent* of $|x|$ while having optimal, rate-1 dependency on the length $|y|$ of the private input — i.e., $|\mathsf{ct}_x(y)| = |y| + \mathrm{O}(1)$. In summary,

  Optimal Succinctness: $\quad |\mathsf{mpk}| = \mathrm{O}(1), \quad |\mathsf{sk}_f| = \mathrm{O}(1), \quad |\mathsf{ct}_x(y)| = |y| + \mathrm{O}(1)$.

  Note that the ideal component sizes are completely independent of the running time or output length of the computation.

- *Decryption Time.* Decryption is also a RAM computation, $\mathsf{Dec}^{f,x,\mathsf{sk}_f,\mathsf{ct}_x(y)}(\mathsf{mpk})$, which on input mpk and with random access to $f, x, \mathsf{sk}_f, \mathsf{ct}_x(y)$, computes the output $U^{x,y,f}()$. We want decryption to be efficient, ideally taking time linear in the instance running time T of the RAM computation with (f, x, y), that is,

  Optimal Decryption Time: $\quad T_{\mathsf{Dec}} = \mathrm{O}(T)$.

[1] With loss of generality, one can think of U as the universal RAM machine.

[2] It may appear that polynomial efficiency is the bare minimal. However, it is possible to consider components whose sizes depend on an upper bound on the length of the information *not* encoded in them. Many schemes are as such, e.g., the FE scheme of [GGH+13] has $|\mathsf{mpk}| = \mathrm{poly}(\max|y|)$, and the sizes of master public keys and ciphertexts of the celebrated ABE scheme by [BGG+14] grow polynomially with the maximal depth of the computation. When a scheme requires knowing an upper bound on parameter Z (e.g., input length, depth, or size), it is referred to as Z-bounded FE/ABE.

Table 1. Comparison among some FE schemes. T is the instance running time of the Turing machine computation. 1-key sls FE stands for 1-key sublinearly succinct FE. Sub-exp iO stands for sub-exponentially secure iO.

citation	functionality	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{ct}\rvert$	T_{Dec}	adaptive	assuming
[GGH+13]	Circuit-FE	$\mathrm{poly}(\lvert C\rvert)$	$\mathrm{poly}(\lvert y\rvert)$	$\mathrm{poly}(\lvert C\rvert)$	no	iO
[KNTY19]	Circuit-FE	$\mathrm{poly}(\lvert C\rvert)$	$\mathrm{poly}(\lvert y\rvert)$	$\mathrm{poly}(\lvert C\rvert)$	yes	1-key sls FE
[GWZ22]	Circuit-FE	$\mathrm{poly}(\lvert C\rvert)$	$\lvert y\rvert + \mathrm{O}(1)$	$\mathrm{poly}(\lvert C\rvert)$	no	iO
[AS16]	TM-FE	$\mathrm{poly}(\lvert f\rvert)$	$\mathrm{poly}(\lvert y\rvert)$	$\mathrm{poly}(\lvert f\rvert, \lvert y\rvert)T$	yes	iO
[AJS17]	TM-FE	$c \cdot \lvert f\rvert + \mathrm{O}(1)$	$c \cdot \lvert y\rvert + \mathrm{O}(1)$	$\mathrm{poly}(\lvert f\rvert, \lvert y\rvert)T$	yes	sub-exp iO
[AM18]	TM-FE	$\mathrm{poly}(\lvert f\rvert)$	$\mathrm{O}(\lvert y\rvert)$	$\mathrm{poly}(\lvert f\rvert)T$	yes	Circuit-FE
[KNTY19]	TM-FE	$\mathrm{poly}(\lvert f\rvert)$	$\mathrm{poly}(\lvert y\rvert)$	$\mathrm{poly}(\lvert f\rvert, \lvert y\rvert)T$	no	1-key sls FE
This work	RAM-PHFE	$\mathrm{O}(1)$	$2\lvert y\rvert + \mathrm{O}(1)$	$\mathrm{O}(T + \lvert f\rvert + \lvert x\rvert + \lvert y\rvert)$	yes	Circuit-FE

Our Results. The question is whether the dream efficiency is simultaneously attainable in all above three dimensions. Towards understanding this, we present both new construction and lower bounds.

New PHFE for RAM with Almost Optimal Succinctness. Starting from polynomially secure FE for circuits with only polynomial efficiency (i.e., all of $\lvert\mathsf{mpk}\rvert, \lvert\mathsf{sk}_f\rvert, \lvert\mathsf{ct}(y)\rvert$ are just $\mathrm{poly}(\lvert f\rvert, \lvert y\rvert)$), which in turn can be constructed from three well-studied assumptions [JLS21,JLS22], we construct PHFE for RAMs with almost optimal succinctness.

Theorem 23 (informal)**.** *Assume polynomially hard selectively secure FE for circuits. There is an adaptively secure PHFE scheme for RAM with arbitrarily long outputs, satisfying*

$$\textbf{\textit{Efficiency of Our PHFE:}} \quad \lvert\mathsf{mpk}\rvert = \mathrm{O}(1), \quad \lvert\mathsf{sk}_f\rvert = \mathrm{O}(1), \quad \lvert\mathsf{ct}_x(y)\rvert = 2\lvert y\rvert + \mathrm{O}(1),$$
$$T_{\mathsf{Dec}} = \mathrm{O}(T + \lvert f\rvert + \lvert x\rvert + \lvert y\rvert).$$

Our construction gives the first collusion-resistant FE for RAM, and also the first FE for any model of computation (e.g., circuit or TMs) with almost optimal succinctness. The only gap to optimality is that the ciphertext has rate-2 dependency on $\lvert y\rvert$ instead of rate-1. Prior constructions work with either circuits [GGH+13,JLS21,JLS22] or Turing machines [AS16,AM18,KNTY19], and achieve only polynomial efficiency as summarized in Table 1 (more discussion on related works in Section 1.2).

As a corollary, we obtain the first ABE for RAM from falsifiable assumptions, and the first ABE for any model of computation with both constant-size keys and constant-size ciphertexts. The only prior construction of ABE for RAM by [GKP+13] relies on non-falsifiable assumptions like SNARK and differing input iO. In terms of efficiency, known schemes achieve either constant-size keys *or* constant-size ciphertexts [ALdP11,YAHK14, Tak14,Att16,ZGT+16,AT20,LL20,LLL22]. Achieving constant-size keys and ciphertexts simultaneously has been an important theoretical open question in the area of ABE (see discussion in [LLL22]). The state-of-the-art is summarized in Table 2.

Corollary (informal)**.** *Assume polynomially secure FE for circuits. There is an adaptively secure key-policy ABE scheme for RAM, and an adaptively secure ciphertext-policy ABE scheme*

Table 2. Comparison among some key-policy ABE schemes. d is maximal depth of the circuit. GGM stands for Generic Group Model, ABP for Arithmetic Branching Program, diO for differing input iO. In [GKP+13], the challenge attribute can be chosen after seeing mpk and keys for adaptively chosen TMs, but no keys can be published after seeing the challenge ciphertext.

citation	**functionality**	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{ct}\rvert$	T_{Dec}	**adaptive**	**assuming**
[BGG+14]	Circuit-ABE	$\mathrm{poly}(d)$	$\lvert x\rvert\,\mathrm{poly}(d)$	$\lvert C\rvert\,\mathrm{poly}(d)$	no	LWE
[LL20]	ABP-ABE	$\mathrm{O}(\lvert C\rvert\cdot\lvert x\rvert)$	$\mathrm{O}(1)$	$\mathrm{O}(\lvert C\rvert\cdot\lvert x\rvert)$	yes	k-lin
[LLL22]	Circuit-ABE	$\mathrm{O}(1)$	$\mathrm{poly}(d)$	$\lvert C\rvert\,\mathrm{poly}(d)$	yes	LWE & GGM
[GKP+13]	RAM-ABE	$\mathrm{O}(1)$	$\mathrm{poly}(\lvert x\rvert)$	$\mathrm{O}(T+\lvert f\rvert+\lvert x\rvert)$	no	SNARK & diO
This work	RAM-ABE	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(T+\lvert f\rvert+\lvert x\rvert)$	yes	Circuit-FE

for RAM, satisfying

Efficiency of Our KP-ABE: $\lvert\mathsf{mpk}\rvert=\mathrm{O}(1)$, $\lvert\mathsf{sk}_f\rvert=\mathrm{O}(1)$, $\lvert\mathsf{ct}(x,\perp)\rvert=\mathrm{O}(1)$, $T_{\mathsf{Dec}}=\mathrm{O}(T+\lvert f\rvert+\lvert x\rvert)$.

Efficiency of Our CP-ABE: $\lvert\mathsf{mpk}\rvert=\mathrm{O}(1)$, $\lvert\mathsf{sk}_x\rvert=\mathrm{O}(1)$, $\lvert\mathsf{ct}(f,\perp)\rvert=\mathrm{O}(1)$, $T_{\mathsf{Dec}}=\mathrm{O}(T+\lvert f\rvert+\lvert x\rvert)$.

It appears that the decryption time of our PHFE (or ABE) scheme is sub-optimal: Besides from the linear dependency on the RAM instance running time T, it additionally depends linearly on $|f|$, $|x|$, and $|y|$. It turns out that, *optimal succinctness and optimal decryption time are at conflict*: We show that conditioned on optimal succinctness, the linear dependency of decryption time on $|f|, |x|$ is inherent. We also show barriers towards removing the dependency on $|y|$, or $|f|$, or $|x|$, irrespective of succinctness. Hence, our PHFE scheme matches the lower bounds and barriers. For ABE, our lower bounds and barriers do not apply. Nevertheless, our ABE scheme matches an existing lower bound by [Luo22] showing that any ABE must satisfy $|\mathsf{ct}|T_{\mathsf{Dec}} = \Omega(|x|)$ and $|\mathsf{sk}|T_{\mathsf{Dec}} = \Omega(|f|)$[3]. Given that our ABE has constant size keys and constant-size ciphertexts, its decryption time matches the lower bound of [Luo22] and is optimal.

Trade-Offs Between Succinctness and Fast Decryption. We now describe our lower bounds in more detail. First, consider the efficiency dependency on the length of public information f and x. We show that unconditionally, it is impossible to simultaneously have key size sublinear in $|f|$, and decryption time with sublinear dependency on $|f|$. Similarly, it is impossible to simultaneously have ciphertext size sublinear in $|x|$, and decryption time sublinear in $|x|$. In fact, these trade-offs exist for the simplest, secret key, 1-key, 1-ciphertext, selectively secure PHFE, and the first trade-off w.r.t. $|f|$ applies to plain FE. More precisely,

Theorem 4 (informal)**.** *For a secret-key, 1-key, 1-ciphertext, selectively secure PHFE with the following efficiency, it must hold that $\alpha \geq 1$ or $\beta \geq 1$.*

$$|\mathsf{sk}| = \mathrm{O}(|f|^{\alpha}) \quad \text{and} \quad T_{\mathsf{Dec}} = \mathrm{O}(T + |f|^{\beta} + |y|)\,\mathrm{poly}(|x|)$$

The same holds for FE where $|x| = 0$.

[3] Theorem 14 in [Luo22] proved the first trade-off w.r.t. ciphertext size. Essentially the same proof also gives the second trade-off w.r.t. key size.

Theorem 5 (informal)**.** *For a secret-key, 1-key, 1-ciphertext, selectively secure PHFE with the following efficiency, it must hold that $\alpha \geq 1$ or $\beta \geq 1$.*

$$|\mathsf{ct}| = O(|x|^{\alpha}) \operatorname{poly}(|y|) \quad \text{and} \quad T_{\mathsf{Dec}} = O(T + |f| + |x|^{\beta}) \operatorname{poly}(|y|)$$

Our PHFE schemes achieve one profile of optimality, $\alpha = 0$ and $\beta = 1$. A natural question that our work leaves open is whether the other profile of optimality, $\alpha = 1$ and $\beta < 1$ or even $\beta = 0$, is attainable. Another question is whether decryption time must depend on the length of the private input y.

Barriers to Decryption Time Optimality. We illustrate barriers to positive answers to the above two questions. We show that a PHFE scheme with decryption time independent of $|y|$, or $|x|$, or $|f|$, implies a primitive called secret-key doubly efficient private information retrieval (SK-DE-PIR), which currently have no construction based on standard assumptions.

Theorem 9 (informal)**.** *If there is a secret-key,* 1-ciphertext*, many-key, selectively secure PHFE scheme with following decryption time, there is a secret-key doubly efficient PIR scheme.*

$$T_{\mathsf{Dec}} = O(T^{e_T} + |f|^{e_f} + |x|^{e_x} + |y|^{e_y}), \text{ where } e_x = 0 \text{ or } e_y = 0$$

If there is a secret-key, many-ciphertext, 1-key*, selectively secure PHFE scheme with the following decryption time, there is a secret-key doubly efficient PIR scheme.*

$$T_{\mathsf{Dec}} = O(T^{e_T} + |f|^{e_f} + |x|^{e_x} + |y|^{e_y}), \text{ where } e_f = 0$$

The same holds for FE where $|x| = 0$, w.r.t. e_y and e_f.

SK-DE-PIR, introduced by [BIPW17,CHR17], allows a client to privately encode a database D into $\tilde{D}$ while keeping a secret key k. Later, client can outsource the encoded database $\tilde{D}$ to a remote storage server, and *obliviously* query the database using k hiding the logical access pattern. Different from ORAM, the server never updates the encoded database nor keeps any additional state. Different from PIR, SK-DE-PIR allows the database to be privately encoded, in exchange for *double efficiency* — for each query, the complexity of both the client and server is independent of the database size $|D|$, whereas PIR necessarily have server complexity $\Omega(|D|)$. The double efficiency of SK-DE-PIR makes it highly desirable. However, so far, there are only candidates [BIPW17,CHR17] based on a new conjecture that permuted local-decoding queries (for a Reed-Muller code with suitable parameters) are computationally indistinguishable from uniformly random sets of points. More recently, a simple "toy conjecture" inspired by (but formally unrelated to) these SK-DE-PIR schemes has been broken [BHMW21]. Constructing SK-DE-PIR based on well-studied assumptions (including primitives like $i\mathcal{O}$/FE) has been left open. Our theorem shows that achieving optimal decryption time in PHFE based on standard assumptions entails resolving this open problem. In fact, even improving the decryption time to be sublinear in $|f|$, or $|y|$, or $|x|$ would imply SK-DE-PIR where the server complexity per query is sublinear in database size $|D|$. Moreover, having a ciphertext of size $c|y|$ implies a SK-DE-PIR where the encoded database size is $|\tilde{D}| = c|D|$ for any c, a even more challenging task.

Asymptotically Efficient Succinct Garbled RAM, and Constant-Overhead iO. The main tool in our construction of efficient RAM-FE is succinct Garbled RAM (GRAM). Initiated by [KLW15,BGL+15,CHJV15], a sequence of works have constructed succinct garbled RAM based on sub-exponentially hard circuit-FE [CH16,CCC+16,ACC+16,CCHR16] and succinct garbled Turing machines based on (polynomially-hard) circuit-FE [KLW15,AJS17,AL18, GS18b].

In this work, we formulate a new notion of succinct GRAM (informally described in the technical overview and formal definition in Section 2.2) geared for building efficient RAM-FE, and construct it based on circuit-FE. Our construction has two consequences: 1) we obtain the first succinct GRAM (under our or the standard notion) based on polynomial hardness, as opposed to sub-exponential hardness as in prior constructions, and 2) using iO for circuits, we obtain iO for RAM with constant-overhead – the size of the obfuscated program is $2|M| + \mathrm{poly}(\lambda, l)$, where M is the original RAM and l is the input length. Constant overhead iO was only known for Turing machines before [AJS17].

1.1 Technical Overview

We start with an overview of our negative results.

Unconditional Lower Bound. As introduced earlier, we show that it is impossible for a secure PHFE to enjoy sublinear dependency on $|f|$ (resp. $|x|$) simultaneously for $|\mathsf{sk}_f|$ (resp. $|\mathsf{ct}_x(y)|$) and T_{Dec} when T_{Dec} is linear in $T, |x|, |y|$ (resp. $T, |f|, |y|$). We illustrate our ideas of proving the contention between

$$|\mathsf{sk}_f| = \mathrm{O}(|f|^{\alpha}) \quad \text{and} \quad T_{\mathsf{Dec}} = \mathrm{O}(T + |f|^{\beta} + |x| + |y|) \qquad \text{for } \alpha < 1 \text{ and } \beta < 1$$

by exhibiting an efficient adversary breaking the security of PHFE for RAM (polynomial factors in the security parameter are ignored).

Adversarial Function and Strategy. The adversary will selectively request one secret key and one ciphertext. Let $n < N$ be determined later.

- The function f is described by a string $R \in \{0,1\}^N$.
- There is no public input, so $x = \perp$.
- The private input y is either $(I \subseteq [N], w \in \{0,1\}^n)$ or $z \in \{0,1\}^n$, where I is a set containing n indexes and w is a one-time pad.

The functionality is simply *reading and XORing* or *outputting as-is,* i.e.,

$$f(x,y) = \begin{cases} R[I] \oplus w, & \text{if } y = (I, w); \\ z, & \text{if } y = z; \end{cases}$$

where $R[I]$ means the n bits of R at the indices in I. Clearly,

$$|f| = \mathrm{O}(N), \quad |\mathsf{sk}| = \mathrm{O}(N^{\alpha}), \qquad |x| = \mathrm{O}(1), \quad |y| = \mathrm{O}(n), \quad T = \mathrm{O}(n), \quad T_{\mathsf{Dec}} = \mathrm{O}(n + N^{\beta}).$$

More precisely, $|y| = \mathrm{O}(n \log n)$, but the $\log n$ factor is absorbed by the $\mathrm{poly}(\lambda)$ factor hidden in the big-O notation.

The adversary chooses

$$\begin{array}{lll} \text{key query} & f \text{ with } R \xleftarrow{\$} \{0,1\}^N, & \\ \text{challenge} & x \leftarrow \perp, & y_0 \xleftarrow{\$} \text{random } (I,w), \quad y_1 \leftarrow z = R[I] \oplus w. \end{array}$$

By our choice, $f(x, y_0) = R[I] \oplus w = z = f(x, y_1)$, so the challenge is well-formed. Upon receiving sk and ct, the adversary simply runs the decryption algorithm on $(\mathsf{sk}, \mathsf{ct})$ with random access to the function, i.e., R, in the clear. Let $L \subseteq [N]$ be the set of indexes in R that is read during decryption.

$$R[I] \oplus w \leftarrow \mathsf{Dec}^{R, x=\perp}(\mathsf{sk}, \mathsf{ct}), \text{ where } \mathsf{Dec} \text{ reads } R[L]$$

The adversary determines that

$$\text{ct is an encryption of } \begin{cases} y_0 = (I, w), & \text{if } |L \cap I| \text{ is large;} \\ y_1 = z, & \text{if } |L \cap I| \text{ is small;} \end{cases}$$

where the threshold for *large* and *small* is described below.

Intuition and Toy Proof. The intuition behind the adversary's strategy is simple. Let L_b be the index set L that decryption accessed when decrypting ct encrypting y_b.

- When ct encrypts y_0, and decryption must correctly recover $R[I]$ (as the adversary knows w). Decryption can only obtain information of $R[I]$ either from sk or via accesses to the tape R. Since $R[I]$ contains n bits of information and by setting $|\mathsf{sk}| = O(N^\alpha) << n$, decryption must read a *large* portion of information of $R[I]$ from the tape R, implying $|L \cap I|$ is large $\Omega(n)$.
- In contrast, when ct encrypts y_1, observe that the joint distribution of $(R, \mathsf{ct}, \mathsf{sk})$ is independent of I, as w is a one-time pad and completely hides I in $y_1 = R[I] \oplus w$. Thus the behavior of, Dec, is independent of I. Since Dec runs in a short time $O(n + N^\beta) \ll N$, without knowing I, where it reads in R cannot overlap with I for a large portion. Therefore, $|L \cap I|$ is likely to be *small*.

It remains to analyze how large and small $|L \cap I|$ is in the above two cases. Let us first consider a toy proof, under the hypothesis that sk contains no information about $R[I]$ at all. We will remove this hypothesis below. By this hypothesis, when ct encrypts y_0, the decryption algorithm must read the entire $R[I]$ from the tape R and hence $|L \cap I| = n$. When ct encrypts y_1, since the indexes L_1 that the decryption algorithm reads from R is independent of I, $|L_1 \cap I|$ follows a hypergeometric distribution, and hence

$$\mathbb{E}\big[\, |L_1 \cap I| \,\big] \leq \frac{T_{\mathsf{Dec}} \cdot n}{N} \ll n \tag{1}$$

This means the adversary can distinguish when ct encrypts y_0 or y_1 with good probability, and contradicts the security of PHFE.

Removing the Hypothesis. The hypothesis that sk contains no information about $R[I]$ at all may well be false. When it is removed, we can no longer argue that $I \subseteq L_0$, as the adversary may obtain some information of $R[I]$ from sk. Our intuition is that $|L_0 \cap I| \geq |I| - |\mathsf{sk}|$, but proving this formally is not trivial as sk may contain arbitrary information of $R[I]$, such as, its parity.

We employ a compression argument. The basic rationale behind a compression argument is that there is no pair of encoding and decoding algorithms $(\mathsf{Encode}, \mathsf{Decode})$, with arbitrarily long shared randomness s, is able to transmit a n-bit random string u (independent of s) from one end to the other, via an encoding v containing less than n bits. Informally,

$$\text{if } \Pr\left[\mathsf{Decode}(s, v) = u \;\middle|\; \begin{array}{c} s \leftarrow \mathcal{D}_s \\ u \leftarrow \{0,1\}^n \\ v \leftarrow \mathsf{Encode}(s, u) \end{array}\right] = 1, \text{ then } |v| \geq |u|$$

(Formal statement by [DTT10] is in Lemma 8.) We show that suppose $|L_0 \cap I| < |I| - |\mathsf{sk}|$ then we can design a pair of $(\mathsf{Encode}, \mathsf{Decode})$ violating the above information theoretic bound.

- the shared randomness s contains the PHFE randomness and $I, w, R[[N] \setminus I]$.
- To encode $u \in \{0,1\}^n$, Encode sets $R[I] = u$. Using s, it then generates a PHFE key sk for R and a ciphertext ct encrypting $y_0 = (I, w)$, runs Dec to obtain the locations L_0 in R that decryption reads. Finally, it sets the codeword to be $v = (\mathsf{sk}, R[L_0 \cap I])$.
- To decode, Decode regenerates ct using s, and runs Dec to obtain the output $z = R[I] \oplus w$ and recover $u = z \oplus w$. During decryption, Dec queries for locations L_0 in R. Every query is in either $R[[N] \setminus I]$ or $R[L_0 \cap I]$; the former can be answered by finding the right element in s and the latter in v.

Suppose $|L_0 \cap I| < |I| - |\mathsf{sk}|$, we have $|v|$ is less than $|I| = |u|$, which contradicts the incompressibility of u. (In the formal proof, we make v fixed-length and suffer from incorrect decoding, hence the statements are probabilistic. See Section 3.1 for more details.)

Stepping back, the compression argument shows that $|L_0 \cap I| \geq |I| - |\mathsf{sk}|$. In contrast, by Equation (1), $|L_1 \cap I| \leq n/2$ with high probability. To show that the adversary can distinguish ct encrypting y_0 or y_1, we can set, e.g., $n = N^{(\alpha+1)/2}$, so that $|\mathsf{sk}| = O(N^\alpha) \ll n$, and $|L_0 \cap I| \geq |I| - |\mathsf{sk}| \geq n/2$. (In the formal proof, N itself is a large $\mathrm{poly}(\lambda)$ to overwhelm $\mathrm{poly}(\lambda)$ factors, which is ignored in this overview.) In summary, any PHFE with both $|\mathsf{sk}|$ sublinear in $|f|$ and T_{Dec} sublinear in $|f|$ (and linear in $T, |x|, |y|$) is insecure.

Computational Implausibility for Fast Decryption. As described earlier, we show implausibility results against constructing a PHFE scheme with fast decryption. Consider a PHFE scheme where the decryption time is $O(T_{\varphi(f,x,y)}^{\beta_T} + |f|^{\beta_f} + |x|^{\beta_x} + |y|^{\beta_y})$ for some constants $\beta_T, \beta_f, \beta_x \beta_y$. We show that even if any of β_x, β_y or β_f is zero, then such a scheme implies SKDEPIR protocols (an informal description of SKDEPIR is in the introduction and formal definition in Section 3.2).

To illustrate our main idea, we start by describing this transformation for the case when the decryption is efficient in the length of the public input x, namely when $\beta_x = 0$. The ideas naturally extend to the other cases.

Since the decryption is efficient in $|x|$, as a first attempt, we set $\mathsf{DB} \in \{0,1\}^n$ as x, and y as empty. The client processes the database DB by first sampling $(\mathsf{mpk}, \mathsf{msk})$ for the PHFE scheme and then encodes the database into $\widetilde{\mathsf{DB}} = (x, \mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y)))$ and sends it over to the server. To look up a location DB_{i_j}, the client can compute a PHFE function key sk_{f_j} for the program f_j that looks up and outputs DB_{i_j} and sends the key as the query to the Server.

$$\widetilde{\mathsf{DB}} = (\mathsf{DB}, \mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (\mathsf{DB}, \perp))) \ , \quad \text{query} = \mathsf{sk}_{f_j} \text{ where } f_j^{\mathsf{DB}} = \mathsf{DB}_{i_j}$$

The server responds to the query by decrypting $\mathsf{ct}_{\mathsf{PHFE}}$ in $\widetilde{\mathsf{DB}}$ using the key sk_{f_j} and with random access to DB, and learns DB_{i_j}. Note that double efficiency requirement is already satisfied. Client only needs to compute a function key sk_{f_j} that can be computed in time polynomial in the description length of f_j, and hence polynomial in λ and $\log n$. On the other hand, due to the supposed efficiency of decryption, the decryption time is polynomial in $T_{f_j(x,y)}, |f_j|, |y|$, which are also polynomial in λ and $\log n$.

While this idea solves the core issue, we have completely missed one important aspect. The scheme reveals the indices $\{i_j\}$ to the Server as the function key $\{\mathsf{sk}_{f_j}\}$ is not guaranteed to hide the function description $\{f_j\}$. To resolve this issue, we observe that if we had a function hiding PHFE scheme, we would have been done. To enable this, we will use similar techniques as used to convert any functional encryption scheme to a function hiding functional encryption scheme [BS15]. Namely, we will compute a symmetric key encryption SKE of the index i (denoted as ctk_1). We will hardwire ctk_1 in the function secret key instead of the index i. The corresponding secret key $\mathsf{SKE.sk}_1$ will be put in the private component y, which will be used to decrypt ctk_1 to learn index i. While this might seem to be enough, we face yet another issue. Learning DB_{i_j} in the clear upon decryption can reveal information about the index i_j to the Server. To fix this, the decryption will output an encryption of DB_{i_j} under another secret key $\mathsf{SKE.sk}_2$ of the secret-key encryption scheme. We will put this key in the private input y along with a PRF key to derive randomness to compute the encryption.

$$\widetilde{\mathsf{DB}} = (\mathsf{DB}, \mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (\mathsf{DB}, (\mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2, \mathsf{PRF}.k)))) \ ,$$
$$\text{query} = \mathsf{sk}_{f_j} \text{ where } f_j[\mathsf{ckt}_1(i_j), \$]^{\mathsf{DB}, (\mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2, \mathsf{PRF}.k)} = \mathsf{SKE}(\mathsf{SKE.sk}_2, \mathsf{DB}_{i_j} \ ; \ \mathsf{PRF}(\mathsf{PRF}.k, \$))$$

Observe how double efficiency is still preserved. We have increased the complexity of f_j multiplicatively by a polynomial amount (in λ and $\log n$), similarly the size of y is also polynomial in λ to store secret keys of SKE and a PRF key. There are few more subtleties : to make the proof go through, we need to use the trojan method in the FE literature [DIJ+13] which requires another encryption key $\mathsf{SKE.sk}_3$ and additional programming.

Overview of Our FE for RAM. At a very high level, we use a *succinct garbled RAM (GRAM) scheme* to lift a FE for circuits to a FE for RAMs. This former can be viewed as a 1-key, 1-ciphertext, secret-key FE for RAMs, where succinctness implies that the running time of key generation and encryption is independent of the running time of the RAM computation. The (collusion resistant) FE for circuits is then used to lift the one-time security to many-time security. This high-level approach first appeared in [AS16] for building FE for TMs. In this work, towards nearly optimally efficient FE for RAMs, we first observe that existing definitions and constructions of succinct GRAM [BGL+15,

CHJV15,CH16,CCC+16,ACC+16,CCHR16] are insufficient. Therefore, we first formulate a new variant of succinct GRAM, termed *laconic GRAM*, and then construct it using polynomially hard FE for circuit. Along the way, we weaken the assumption underlying succinct GRAM schemes from iO, which requires sub-exponentially hard FE for circuit, to polynomially-hard FE for circuits.

Let us first review the syntax and security of standard GRAM schemes. They consist of the following algorithms. The encoding algorithm encodes a database D into $\hat{D}$ and outputs a private state τ. The garbling algorithms uses τ to garble a RAM M into $\hat{M}$ and outputs a collection of input labels $\{L_{i,b}\}_{i,b}$. The evaluation algorithm given the garbled RAM $\hat{M}$, a subset of labels L_k corresponding to an input k, and random access to $\hat{D}$, returns the output $M^D(k)$ of the RAM computation. Simulation based security ensures that $\hat{D}, \hat{M}, L_k = \{L_{i,k_i}\}_i$ can be simulated using only the output $M^D(k)$. The efficiency of different algorithms is is described below.

$$(\hat{D},\tau) \leftarrow \mathsf{Encode}(D)\ , \quad \hat{M}, \{L_{i,b}\}_{i,b} \leftarrow \mathsf{Garble}(M,\tau)\ , \quad M^D(k) = \mathsf{Eval}^{\hat{D}}(\hat{M}, L_k)$$
$$|\hat{D}| = |D|\,\mathrm{poly}(\lambda)\ , \quad |\hat{M}| = \mathrm{poly}(\lambda), \quad T_{\mathsf{Eval}} = T\,\mathrm{poly}(|M|,\lambda)$$

We now describe why the standard notion falls short for our purpose of building very efficient FE for RAMs and how to address these issues. An informal definition of our succinct GRAM is provided in Figure 1.

- *many-tape v.s. single-tape*: To start with, we consider RAM computation with multiple tapes $U^{x,y,f}(1)$ instead of a single tape $M^D(k)$.

- *public-tape v.s. private-tape*: Some of the tapes we consider, such as x and f, are public. But standard GRAM only provides a mechanism for encoding private tape, and the encoding is necessarily at least as long as the tape itself. However, optimal succinctness requires the FE ciphertext- and key-size to be independent of $|x|$ and $|f|$. Hence we cannot afford to encode x, f as in standard GRAM. Instead, our new notion of succinct GRAM has a Compress algorithm that compresses the public tapes into hashes/digests h_x and h_f; the Garble algorithm "ties" these hashes to the garbled program $\hat{U}$; and finally Eval makes random access to x and f in the clear directly (just as the decryption algorithm of RAM-FE does).

- *rate-2 encoding v.s. rate-*poly(λ) *encoding of private tape:* Our setting also has private tape, namely y. But optimal succinctness requires concretely efficient, rate-1 or rate-2, encoding of y, whereas standard GRAM allows much worse rate poly(λ). To achieve concretely efficient encoding, we can only encrypt y using a rate-1 encryption scheme. To bind the encryption $\hat{y}$ with a garbled program, we simply treat $\hat{y}$ as yet another public tape just like x, f. In other words, we consider the modified RAM computation $\bar{U}^{x,\hat{y},f}(k) = U^{x,y,f}(1)$, where k is the secret key of the rate-1 encryption. As such, our succinct GRAM only need to handle public tapes.

  In our construction of FE, additional care needs to be taken to ensure that our GRAM security implies that y is hidden. To achieve this, We rely on existing techniques [NY15], which requires two (rate-1) encryption of y with independent keys so that different security hybrids can invoke the semantic security of different encryption. This is is why our FE has rate-2 dependency on $|y|$, instead of rate-1. We omit details in this overview.

- *reusable digests v.s. one-time encoding:* In standard GRAM, the database encoding $\hat{D}$ can only be used, *once*, by a single garbled program $\hat{M}$ generated using the right state τ. The technical cause of the one-time security of $\hat{D}$ is due to the use of ORAM in order to hide the access pattern of M to D; the same ORAM storage $\hat{D}$ cannot be used by multiple garbled programs[4]. The one-time security means that when using succinct GRAM to construct RAM-FE, the decryption of every ciphertext and key pair must generate fresh encoding $\hat{D}$ (and $\hat{M}$). Such fresh encoding can only be generated using the underlying FE for circuit, by encoding D in its key or ciphertext, which would lead to large polynomial dependency on $|D|$.

  Our notion of succinct GRAM compresses the public tapes into hashes $h_x, h_f, h_{\hat{y}}$. For the same reasons, we cannot afford to generate fresh hashes at decryption of every pair of ciphertext and key. Instead, our hashes are *reusable* – they can be tied to multiple garbled programs; this is ensured by the fact that our Garble algorithm does not take any private state from the Compress algorithm. A technical issue we must resolve is how to hide access pattern to the public tapes $x, \hat{y}, f$ since they are not encoded using ORAM, which we discuss later.

- *Difference in decryption time:* The reusability comes at a cost. In our new notion, evaluation time is $(T + |x| + |y| + |f|)\,\mathrm{poly}(\lambda)$ whereas standard GRAM has evaluation time $T\,\mathrm{poly}(|M|, \lambda)$ independent of tape size $|D|$. Nevertheless, our lower bound for RAM-FE implies that the dependency on $|x|, |y|, |f|$ is hard to get around (as our succinct GRAM implies RAM-FE with the same decryption time).

- *RAM with long outputs v.s. single-bit output:* Standard succinct GRAM handles RAM computation with a single-bit output. To handle RAM with m-bit output, it reduces to creating m instances of garbled RAM, one for each output bit. Under simulation security, the size of the garbled RAM necessarily grows linearly with the output length m.

  In our notion, we require garbling RAM with arbitrarily long outputs, without efficiency degradation in the output length. To do so, we switch to indistinguishability based security instead of simulation security.

Putting the above pieces together, we formulate succinct GRAM as in Figure 1.

Our Construction of Succinct GRAM. One approach towards constructing succinct GRAM for TMs or RAMs is starting from a non-succinct GRAM for TMs or RAMs where the size of the garble program scales with the worst-case time complexity of the TM/RAM U, into one that is succinct. First introduced in [BGL+15], this approach uses iO to obfuscate a circuit that on input an index t, outputs the t'th component in the non-succinct GRAM. If every component of the non-succinct GRAM can be locally generated using a small circuit of size $\mathrm{poly}(|U|, \lambda)$, then the obfuscated circuit also has size $\mathrm{poly}(|U|, \lambda)$ and can be viewed as the succinct garbled program. To prove security, [BGL+15] identified that the non-succinct garbling scheme must satisfy another property,

[4] This should be distinguished from the scenario of GRAM with persistent database where a sequence of garbled program $(\hat{M}_1, \hat{M}_2, \cdots)^{\hat{D}}$ are executed sequentially with $\hat{D}$. The difference lies in that in sequential execution, each garbled RAM can modify $\hat{D}$ and the changes are kept persistently to the next computation. Here, we are considering the scenario where the unmodified $\hat{D}$ is used by multiple garbled program, which breaks ORAM security.

Our Notion of Succinct Garbled RAM

- $\mathsf{Compress}(\tau, D_\tau)$ compresses the τ'th public tape D_τ into a short hash digest_τ of length $\mathrm{poly}(\lambda)$. It runs in time $O(|D_\tau|)$.
- $\mathsf{Garble}(U, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$ outputs a garbled program $\widehat{U}$ tied with hashes of the public tapes, and pairs of labels $\{L_{i,b}\}_{i,b}$. It runs in time $\mathrm{poly}(\lambda)$ (U has constant-size).
- $\mathsf{Eval}^{D_1,\ldots,D_\mathcal{T}}(U, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{U}, L_k)$ returns the (long) output of RAM computation $U^{D_1,\ldots,D_\mathcal{T}}(k)$ if $L_k = \{L_{i,k_i}\}_i$. It runs in time $(T + \sum_\tau |D_\tau|)\,\mathrm{poly}(\lambda)$.

  Security guarantees that for two computations $U^{D_1,\ldots,D_\mathcal{T}}(k_0)$ and $U^{D_1,\ldots,D_\mathcal{T}}(k_1)$ with different inputs but identical outputs and running time, the distributions of $(\widehat{U}, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, L_{k_0})$ and $(\widehat{U}, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, L_{k_1})$ are indistinguishable. This holds when the tapes $\{D_i\}$ are chosen adaptively dependent on the hashes of previously chosen tapes, before the program U and inputs k_0, k_1 are chosen.

Figure 1. Our notion of succinct GRAM.

articulated later by [AL18], called *local simulation*. Informally, it requires that the non-succinct garbled scheme is proven secure via a sequence of hybrids, where components of every hybrid garbled program can be locally generated using a small circuit, and in neighboring hybrids, only a few components changes. Beyond succinct garbling, local simulation has also found application in achieving adaptive security [BHR12] of garbling schemes. A sequence of works developed local simulation strategies for garbled circuit [HJO+16,GS18a], TM [GS18b,AL18], and RAM [GOS18]. Most notably, the work by Garg and Srinivasan [GS18a] introduced a beautiful pebbling technique for realizing local simulation.

Our construction of succinct GRAM proceeds in steps. First, we use the Garg–Srinivasan [GS18a] pebbling technique to obtain a non-succinct GRAM that has a local simulation proof for, however, weak indistinguishability security called fixed memory security. Indistinguishability only holds when the two RAM computations have not only identical outputs and running time, but also identical memory access pattern and content. Then by the same approach of [BGL+15,GS18b,AL18], we turn it into a succinct GRAM, still with only *fixed memory security*, relying on iO for polynomial-sized domain which is implied by polynomially-hard FE for circuits. (In the body, we merge these two steps together in Section 4.) Many details need to be ironed out in order to realize our new notion of succinct GRAM. For example, prior works [GS18a,GS18b,AL18] deal with single-bit output RAM and can build intermediate security hybrid where the suffix (i.e., the last certain number of steps) of a computation is simulated by hard-wiring the single-bit output. In contrast, we directly garble RAM with arbitrarily long outputs. Hard-wiring the long output would compromise the local simulation property (since the hybrid garbled program can no longer be locally generated by a small circuit). To avoid this, we build a hybrid GRAM that runs with one input k_0 in the prefix of the computation and with another input k_1 in the suffix (recall that these two inputs produce identical memory). This ensures that the output is always correctly computed, while keeping local simulation. Similar hybrids appeared in [GOS18] for different reasons.

Finally, we lift *fixed memory security* to full security using punctured PRF and ORAM to protect the memory content and access pattern. One issue is that in our succinct GRAM, the public input tapes $D_1, \cdots, D_{\mathcal{T}}$ are not encoded using ORAM, and evaluation is given random access to them in the clear. Yet, to ensure security, evaluation must access these tapes in an oblivious way, independent of the input k_0 or k_1. To solve this issue, we consider a modified RAM program U', which has random access to $D_1, \cdots, D_{\mathcal{T}}$ and additionally a work-tape that contains an ORAM storage that initially contains no elements. The program U' starts with linearly scanning every tape D_τ and inserting every element into the ORAM storage. Only after that, it emulates the execution of the original RAM program U; every time U read from/write to a location in tape D_τ, U' accesses the corresponding location on its work-tape through ORAM, which hides the access pattern of U. The intuition is that since the access pattern of U' is independent of the input, it suffices to garble it using GRAM with fixed memory security. Clearly, the running time of U' scales linearly with the total length of all tapes $\sum_\tau |D_\tau|$. This is why the evaluation time of our succinct GRAM is linear in $\sum_\tau |D_\tau|$. Nevertheless, our lower bound shows that this dependency is hard to circumvent. Lastly, we mention that to prove security, one must ensure that the use of ORAM does not hurt local simulation. Fortunately, the techniques by [CH16] provide a solution.

1.2 Related Works

Our new construction significantly improve upon the efficiency of prior FE and ABE schemes. The state-of-the-art is summarized in Table 1 and 2; below, we compare with prior works in more detail.

FE for Circuits. The first construction of collusion resistant FE for polynomial-sized circuits is by [GGH+13] and based on iO, which in turn relies on subexponential hardness. Later works [GS16,LM16,KNT18,KNTY19] improved the assumption from iO to 1-key FE with sublinearly compact ciphertext, $|\mathsf{ct}(y)| = \mathrm{poly}(|y|)|T_f|^{1-\varepsilon}$, where ε is a positive constant and T_f is the circuit complexity of f. The latter has been recently constructed by [JLS21,JLS22] from the polynomial hardness of three well-studied assumptions. However, these circuit-FE schemes have polynomial efficiency. The only exception is the recent construction by [GWZ22], which has rate-1 ciphertext $|\mathsf{ct}(y)| = |y| + O(1)$ but still large secret keys $|\mathsf{sk}_f| = \mathrm{poly}(T_f)$.

FE for Turing Machines. Several works constructed FE for Turning machines with arbitrary-length inputs, first from the assumption of iO [AS16], then from FE for circuits [AM18], and more recently from 1-key sublinearly compact FE [KNTY19] (which implies collusion resistant FE for circuits). The construction of [AS16] relies on a 1-key 1-ciphertext secret-key FE for TMs, which is essentially a succinct garbling scheme for TMs with indistinguishability security. They constructed it by modifying the succinct garbling for TM of [KLW15]. Later, the works of [AL18,GS18b] improved and simplified the construction of succinct garbling for TM. Following these works, [KNTY19] improved the assumption to the existence of 1-key sublinearly succinct FE, and showed that their garbling scheme can be combined with [AS16] to obtain FE for TMs. On the other hand, [AM18] presented an alternative direct approach to FE for TMs from FE for circuits, that does not use succinct garbling for TM. Prior constructions of FE for TMs [AS16, AM18,KNTY19] focus more on weakening the underlying assumptions, and only show

polynomial efficiency. Examining their schemes, we think they achieve efficiency listed in Table 1.

FE for Bounded-Input RAM. A line of research obtained *bounded-input* iO for Turning machines [KLW15,AJS17,GS18b] and for RAMs [BGL+15,CHJV15,CH16,CCC+16,ACC+16, CCHR16]. Plugging these iO schemes into the [GGH+13] construction yields *bounded-input* FE for TMs and RAMs. Unfortunately, these schemes are not full-fledged FE for TMs and RAMs for the following reason: existing iO only handles bounded-input TMs and RAMs in the sense that the obfuscator needs to know the maximal input length $\max(|y|)$ to the TM/RAM f being obfuscated. (Constructing iO for unbounded input TMs/RAMs remains an major open question.) Plugging this into the FE construction of [GGH+13] gives a scheme where the key generation algorithm needs to know the maximal input length $\max(|y|)$, despite that the TM/RAM f could process arbitrarily long inputs. Such FE is called bounded-input FE. In terms of efficiency, the secret key contains an obfuscated program of size $\mathrm{poly}(|f|, \max(|y|))$ when using RAM-iO of [CHJV15,CH16], and $\mathrm{poly}(|f|, \max(|y|), S)$ where S is the space complexity of f when using RAM-iO of [BGL+15].

In summary, our construction gives the first full-fledged (PH-)FE scheme for RAM computation (with arbitrarily long inputs and outputs), significantly improves the efficiency of prior FE schemes, and matches lower bounds.

ABE for Circuits and Turing Machines. Since FE implies ABE, the above mentioned FE schemes immediately imply ABE with the same level of efficiency. The literature on ABE focuses on constructing ABE from weaker assumptions, or achieving better efficiency, etc. The celebrated works of [GVW13,BGG+14] showed that ABE for *bounded-depth* circuits can be constructed from Learning With Errors (LWE). Parameters of these schemes however depends polynomially in the maximal depth d of the circuits supported, namely, $|\mathsf{mpk}| = \mathrm{poly}(d)$, $|\mathsf{sk}_f| = \mathrm{poly}(d)$, $|\mathsf{ct}(x, m)| = \mathrm{poly}(d)|x|$, where the encrypted message m is a bit, and decryption time is $T_{\mathsf{Dec}} = \mathrm{poly}(d)T$. A recent work [LLL22] improved this construction to obtain constant-size keys while keeping the sizes of master public key and ciphertext, but at the cost of additionally relying on the Generic Group Model (GGM). ABE for low-depth computation such as NC^1 or (arithmetic) Branching programs can be constructed using pairing groups, where several schemes have either constant-size keys *or* constant-size ciphertexts, but never both [ALdP11,YAHK14,Tak14, Att16,ZGT+16,AT20,LL20].

The work of [GKP+13] constructed ABE for Turing machines and RAMs with constant-size secret keys $|\mathsf{sk}_f| = O(1)$, but still large ciphertexts $|\mathsf{ct}(x, m)| = \mathrm{poly}(|x|)$. However, their scheme uses SNARK and differing-input iO, which cannot be based on falsifiable assumptions. Another work [AFS19] tries to construct ABE for RAMs from LWE, at the cost of making master public keys, secret keys, and ciphertexts all grow polynomially with the maximal running time of RAMs supported; in other words, this is an ABE for bounded-time RAMs.

In summary, we give the first full-fledged ABE for RAMs from falsifiable assumptions, which has simultaneously constant-size secret keys and constant-size ciphertexts, and best-possible decryption time matching known lower bound [Luo22].

2 Preliminaries

Throughout the paper, we denote the security parameter by λ. An ensemble X of objects is a sequence $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ of such objects indexed by λ. Except in definitions, we omit λ and (ab-)use the symbol X of an ensemble to represent the λ^{th} object X_λ. Let S be a set and n a natural number, then $S^{\leq n}$ is the set of *non-empty* strings over alphabet S of length at most n. For a string x, its i^{th} symbol is denoted by $x[i]$. As an example, the third *bit* in some sufficiently long $x \in (\{0,1\}^2)^{\leq 3}$ is $x[2][1]$. For two strings x, y, we write $x\|y$ for their concatenation. We denote by $C[x_1]$ the circuit C with x_1 hardwired as the first portion of input so that $C[x_1](x_2) = C(x_1, x_2)$.

2.1 Multi-Tape Random Access Machine

We use a model of *multi-tape* random access machines, with several read-only input tapes and one read/write working tape. In addition to the tapes, the machine also has a short input and maintains a (small) state.[5] The behavior of a RAM is described by its step circuit:

$$(\ell_1, \ldots, \ell_{\mathcal{T}}, w, \mathsf{oldst}, \mathsf{rdata}) \xmapsto{\mathsf{CPU}} (\mathsf{done}, \mathsf{newst}, \tau, i, \mathsf{wdata}, \mathsf{out}).$$

The step circuit takes as input *i*) the input tape lengths $\ell_1, \cdots, \ell_{\mathcal{T}}$ and a short input w, which stay the same throughout the computation, and *ii*) the previous state oldst and the last string rdata read from the tapes, which change from step to step during the computation. It outputs whether the machine should halt in the flag done. If the machine does not halt, it outputs the next state newst, the next tape τ and address i to read from. Additionally, if τ specifies the working tape, then the step circuit also outputs a string wdata, which gets written at address i of the working tape. Each step also optionally generates out, one bit of output.

The execution of a machine M with $D_1, \ldots, D_{\mathcal{T}}$ on the input tapes and w as the short input begins by zero-initializing[6] the working tape, the state, and the last-read bit. Throughout the process, the short input stays unchanged and the state is updated by the machine. For each step, the requested location (determined by the output of the previous step) is read, whose value is passed into the last-read bit of the current step, before that location is overwritten. For simplicity, we insist that overwriting happen if and only if the requested tape is the working tape (the input tapes are read-only). The whole sequence of out's, including the timing information of at which step each output bit is produced, is the output of the execution. This process is denoted by $M^{D_1,\ldots,D_{\mathcal{T}}}(w)$. We now give the formal definition.

Definition 1 (multi-tape RAM)**.** Let $\mathcal{T}$ be a natural number. A *$\mathcal{T}$-tape RAM* M is specified by its step circuit CPU,

$$\mathsf{CPU} : (\ell_1, \ldots, \ell_{\mathcal{T}}, w, \mathsf{oldst}, \mathsf{rdata}) \mapsto (\mathsf{done}, \mathsf{newst}, \tau, i, \mathsf{wdata}, \mathsf{out}),$$

which is subject to the constraints below. The set of $\mathcal{T}$-tape RAMs is denoted by $\mathsf{RAM}_{\mathcal{T}}$.

[5] The distinction between input and state is arbitrary, and many works formulate them as a single entity. We choose to separate them for clearer semantics.

[6] We do not consider persistent memory in this work.

Addresses and Tapes. An *(input-tape) address* is an ℓ_{addr}-bit string indexing a *cell*. Each input tape consists of at most $2^{\ell_{\text{addr}}}$ cells, and each cell is ℓ_{cell}-bit long, i.e., the content D of each input tape is in $(\{0,1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$. For technical convenience, we allow the working tape to have different address and cell lengths, denoted by ℓ_{ADDR} and ℓ_{CELL}, than the input tapes. Without loss of generality (efficiency- and security-wise), we assume $\ell_{\text{ADDR}} \geq \ell_{\text{addr}}$ and $\ell_{\text{CELL}} \geq \ell_{\text{cell}}$. Conceptually, the working tape has exactly $2^{\ell_{\text{ADDR}}}$ cells.

Inputs and Outputs of CPU. The inputs include:

- $\ell_\tau \in [2^{\ell_{\text{addr}}}]$ is the length (in cells) of the τ^{th} input tape.
- $w \in \{0,1\}^{\ell_{\text{in}}}$ is the short input of length ℓ_{in}.
- $\mathsf{oldst} \in \{0,1\}^{\ell_{\text{st}}}$ is the state of length ℓ_{st}.
- $\mathsf{rdata} \in \{0,1\}^{\ell_{\text{CELL}}}$ is the string that was read.

Among the outputs, done is a flag indicating whether the machine should halt. If done is set, all of newst, τ, i, wdata, out should be $\perp$. Otherwise:

- $\mathsf{newst} \in \{0,1\}^{\ell_{\text{st}}}$ is the new state.
- $\tau \in [\mathcal{T}] \cup \{\mathsf{work}\}$ is the tape to read from (and possibly write to).
  - If $\tau \in [\mathcal{T}]$, then $i \in [\ell_\tau]$ is the address to read from on the τ^{th} input tape, and $\mathsf{wdata} = \perp$.
  - If $\tau = \mathsf{work}$, then $i \in [2^{\ell_{\text{ADDR}}}]$ is the address to read from and write to on the working tape, and $\mathsf{wdata} \in \{0,1\}^{\ell_{\text{CELL}}}$ is the string to be written.
- $\mathsf{out} \in \{\perp, 0, 1\}$ is an optional output bit.

Executing RAM. We now formally present the steps involved in executing RAM. Let $D_1, \ldots, D_{\mathcal{T}} \in (\{0,1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$ be the input tape contents and $w \in \{0,1\}^{\ell_{\text{in}}}$ the short input. To execute $M^{D_1,\ldots,D_{\mathcal{T}}}(w)$:

1. Let $\mathsf{st}_0 \leftarrow 0^{\ell_{\text{st}}}$, $\mathsf{rdata}_0 \leftarrow 0^{\ell_{\text{CELL}}}$, $D_{\mathsf{work},0} \leftarrow 0^{\ell_{\text{CELL}} 2^{\ell_{\text{ADDR}}}}$, $t \leftarrow 1$.
2. Let $(\mathsf{done}_t, \mathsf{st}_t, \tau_t, i_t, \mathsf{wdata}_t, \mathsf{out}_t) \leftarrow \mathsf{CPU}(|D_1|, \ldots, |D_{\mathcal{T}}|, w, \mathsf{st}_{t-1}, \mathsf{rdata}_{t-1})$.
3. If done_t is set, the execution halts.
4. If $\tau_t \in [\mathcal{T}]$, let $\mathsf{rdata}_t \leftarrow D_\tau[i_t] \| 0^{\ell_{\text{CELL}} - \ell_{\text{cell}}}$ and $D_{\mathsf{work},t} \leftarrow D_{\mathsf{work},t-1}$.
5. Otherwise, $\tau_t = \mathsf{work}$, let $\mathsf{rdata}_t \leftarrow D_{\mathsf{work},t-1}[i_t]$ and let $D_{\mathsf{work},t}$ be $D_{\mathsf{work},t-1}$ with $D_{\mathsf{work},t}[i_t]$ replaced by wdata_t.
6. Let $t \leftarrow t+1$ and go back to step 2.

We assume, without loss of generality, that all constraints of RAM are respected during all executions.[7] For a halting execution,

[7] For example, the memory access should never be out of range. Given a step circuit, it is efficient to wrap it inside another circuit checking all the constraints and correcting any error (e.g., by halting immediately when a constraint is violated).

- its *running time* $T = \mathsf{time}(M, D_1, \ldots, D_{\mathcal{T}}, w)$ is the value of t when it halts;[8]
- its *running space* is $\mathsf{space}(\cdots) = \max_{t<T, \tau_t=\mathsf{work}} i_t$;[9]
- its *state sequence* is $\mathsf{stS}(\cdots) = (\mathsf{st}_1, \ldots, \mathsf{st}_{T-1})$;
- its *address sequence* is $\mathsf{addrS}(\cdots) = (\tau_1, i_1, \ldots, \tau_{T-1}, i_{T-1})$;
- its *write sequence* is $\mathsf{writeS}(\cdots) = (\mathsf{wdata}_1, \ldots, \mathsf{wdata}_{T-1})$;
- its *output sequence* is $\mathsf{outS}(\cdots) = (\mathsf{out}_1, \ldots, \mathsf{out}_{T-1})$.

It is worth noting that the machine does *not* necessarily produce non-$\perp$ outputs at the end or even consecutively. For most part of this work, we do not care whether the output sequence is in some particular format. When defining secure evaluation of RAM, we simply require that all information be hidden except the output sequence, which implies that the running time and the time steps when each non-$\perp$ output is produced are not supposed to be hidden as such information is incorporated in the output sequence.

2.2 Laconic Garbled RAM

Our notion of garbling RAM laconically involves two steps. First, a short digest is created for each input tape. The digest has length independent of that of the tape itself, and the digest must be computable in linear time. Second, the RAM and the short digests are put together to produce a garbled program and the labels. This procedure runs in time poly-logarithmic in the RAM running time. Given a garbled program and one set of labels (selected by the bits of the short input), the evaluation procedure computes the output sequence in time linear in the RAM running time.

We consider indistinguishability-based security for the short input. The input tape contents can be chosen adaptively, but the short input cannot depend on the garbled program (i.e., selectiveness). We also consider several weaker notions, where the running space is bounded and more information about the execution is allowed to be revealed.

Definition 2 (LGRAM)**.** Let $\mathcal{T}$ be a natural number. A *laconic garbling scheme* for $\mathcal{T}$-tape RAM consists of 3 algorithms:

- $\mathsf{Compress}(1^\lambda, 1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}, \tau, D_\tau)$ takes as input a cell length $1^{\ell_{\mathrm{cell}}}$, an address length $1^{\ell_{\mathrm{addr}}}$, an input tape index $\tau \in [\mathcal{T}]$, and its content $D_\tau \in (\{0,1\}^{\ell_{\mathrm{cell}}})^{\leq 2^{\ell_{\mathrm{addr}}}}$. It outputs a short digest digest_τ. The algorithm runs in time $|D_\tau| \operatorname{poly}(\lambda, \ell_{\mathrm{cell}}, \ell_{\mathrm{addr}})$, and its output length is $\operatorname{poly}(\lambda, \ell_{\mathrm{cell}}, \ell_{\mathrm{addr}})$.
- $\mathsf{Garble}(1^\lambda, T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$ takes as input a time bound $T_{\max} \in \mathbb{N}^*$, a $\mathcal{T}$-tape RAM M, and $\mathcal{T}$ input tape digests. It outputs a garbled program $\widehat{M}$ and ℓ_{in} pairs of labels $\{L_{i,b}\}_{i\in[\ell_{\mathrm{in}}], b\in\{0,1\}}$. The algorithm runs in polynomial time.
- $\mathsf{Eval}^{D_1,\ldots,D_{\mathcal{T}}}(1^\lambda, T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}, \{L_i\}_{i\in[\ell_{\mathrm{in}}]})$ takes as input the time bound $T_{\max}$, the $\mathcal{T}$-tape RAM M, the input tape digests, the garbled program $\widehat{M}$, and one

[8] If the execution does not halt, we say $\mathsf{time}(\cdots) = +\infty$.

[9] To be complete, $\mathsf{space}(\cdots)$ is defined to be 1 if the working tape is never accessed.

set of labels. Given random access to the input tapes, it is supposed to compute the output sequence. The algorithm runs in time

$$\left(\min\{T_{\max}, \mathsf{time}(M, D_1, \ldots, D_{\mathcal{T}}, w)\} + \sum_{i=1}^{\mathcal{T}} |D_\tau|\right) \mathrm{poly}(\lambda, |M|, \log T_{\max}),$$

where w is the short input corresponding to the labels.

The scheme must be *correct,* i.e., for all $\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}}, \ell_{\mathsf{in}} \in \mathbb{N}$, $T_{\max} \in \mathbb{N}^*$, $D_1, \ldots, D_{\mathcal{T}} \in (\{0,1\}^{\ell_{\mathsf{cell}}})^{\leq 2^{\ell_{\mathsf{addr}}}}$, $w \in \{0,1\}^{\ell_{\mathsf{in}}}$, $\mathcal{T}$-tape RAM M, if $M^{D_1,\ldots,D_{\mathcal{T}}}(w)$ halts in time at most $T_{\max}$, then

$$\Pr\left[\begin{array}{r} \mathsf{digest}_\tau \xleftarrow{\$} \mathsf{Compress}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}, \tau, D_\tau) \qquad \forall \tau \in [\mathcal{T}] \\ (\widehat{M}, \{L_{i,b}\}_{i\in[\ell_{\mathsf{in}}], b\in\{0,1\}}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}) \\ \vdots \\ \mathsf{Eval}^{D_1,\ldots,D_{\mathcal{T}}}(1^\lambda, T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}, \{L_{i,w[i]}\}_{i\in[\ell_{\mathsf{in}}]}) \\ = \mathsf{outS}(M, D_1, \ldots, D_{\mathcal{T}}, w) \end{array}\right] = 1.$$

Definition 3 (LGRAM security)**.** An LGRAM scheme (Definition 2) is *(tape-adaptively, indistinguishability-based) secure* if $\mathsf{Exp}^0_{\mathrm{LGRAM}} \approx \mathsf{Exp}^1_{\mathrm{LGRAM}}$, where $\mathsf{Exp}^b_{\mathrm{LGRAM}}(1^\lambda)$ with adversary $\mathcal{A}$ proceeds as follows:

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive from it the cell length $1^{\ell_{\mathsf{cell}}}$ and the address length $1^{\ell_{\mathsf{addr}}}$.
- **Tape Choices.** Repeat this for $\mathcal{T}$ rounds. In each round, $\mathcal{A}$ chooses $\tau \in [\mathcal{T}]$ and $D_\tau \in (\{0,1\}^{\ell_{\mathsf{cell}}})^{\leq 2^{\ell_{\mathsf{addr}}}}$. Upon receiving such choice, run
$$\mathsf{digest}_\tau \xleftarrow{\$} \mathsf{Compress}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}, \tau, D_\tau)$$
and send digest_τ to $\mathcal{A}$.
- **Challenge.** $\mathcal{A}$ chooses the instance running time bound $1^{\overline{T}}$ (in unary), the time bound $T_{\max}$ (in binary), a $\mathcal{T}$-tape RAM M, and two inputs w_0, w_1. Run
$$(\widehat{M}, \{L_{i,b}\}_{i\in[\ell_{\mathsf{in}}], b\in\{0,1\}}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$$
and send $(\widehat{M}, \{L_{i,w_b[i]}\}_{i\in[\ell_{\mathsf{in}}]})$ to $\mathcal{A}$.
- **Guess.** $\mathcal{A}$ outputs a bit b'. The output of the experiment is b' if all of the following conditions hold:
  - The τ's in all rounds of **Tape Choices** are distinct.
  - Both $M^{D_1,\ldots,D_{\mathcal{T}}}(w_0)$ and $M^{D_1,\ldots,D_{\mathcal{T}}}(w_1)$ halt in time $T \leq \overline{T} \leq T_{\max}$ with identical *output sequences* $\mathsf{outS}(\cdots)$.

  Otherwise, the output is set to 0.

Weaker security notions are obtained by strengthening the second condition in **Guess**.

- *Fixed-address security.* "... with identical $\mathsf{outS}(\cdots)$ and $\mathsf{addrS}(\cdots)$".
- *Fixed-memory security.* "... with identical $\mathsf{outS}(\cdots)$, $\mathsf{addrS}(\cdots)$, and $\mathsf{writeS}(\cdots)$".

We emphasize that the scheme (Definition 2) must be efficient when $T_{\max}$ is written in binary. However, security only has to hold for polynomially large instance running time, which is captured by the requirement that the adversary must produce $1^{\overline{T}}$, an upper bound of the instance running time in unary.

Bounded LGRAM. As an intermediate primitive, we consider *bounded* LGRAM:

Definition 4 (bounded LGRAM and security)**.** The notion of a *bounded LGRAM scheme* is obtained by modifying Definition 2 as follows:

- The evaluation procedure runs in time

$$\left(T_{\max} + \sum_{i=1}^{\mathcal{T}} |D_\tau|\right) \mathrm{poly}(\lambda, |M|, \log T_{\max}).$$

- Correctness is required only if $M^{D_1,\ldots,D_{\mathcal{T}}}(w)$ halts using space at most $T_{\max}$ in time at most $T_{\max}$.

Its security notions are obtained by modifying Definition 3 as follows:

- In **Challenge**, the adversary chooses $1^{T_{\max}}$ (instead of $1^{\overline{T}}$ and $T_{\max}$).
- In **Guess**, the second condition is strengthened to "... halt using space $S \le T_{\max}$ in time $T \le T_{\max}$ with identical ..."

2.3 Partially Hiding Functional Encryption and FE for Circuits

We define partially hiding functional encryption (PHFE) with respect to functions of the form

$$\varphi : F \times X \times Y \to \{\bot\} \cup (\mathbb{N}_+ \times Z),$$

where F, X, Y, Z are the sets of function description, public input, private input, and output, respectively. Each key is associated with some $f \in F$, and each ciphertext encrypts some private input $y \in Y$ and is associated with some public input $x \in X$. The decryptor should be able to recover z if $\varphi(f, x, y) = (T, z)$, in which case T is the time to compute z from f, x, y in the clear and serves as a baseline for decryption efficiency. For security, we only consider f, x, y for which T is polynomially bounded. On the other hand, when $\varphi(f, x, y) = \bot$, we require neither correctness nor security. This can be used to exclude non-halting computation.

Definition 5 (PHFE)**.** Let $\Phi = \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}$ be a sequence of functionality families such that each $\varphi \in \Phi_\lambda$ is a function

$$\varphi : F_\varphi \times X_\varphi \times Y_\varphi \to \{\bot\} \cup (\mathbb{N}_+ \times Z_\varphi).$$

A *partially hiding functional encryption scheme* for Φ consists of 4 algorithms, with efficiency defined in Definition 6:

- $\mathsf{Setup}(1^\lambda, \varphi)$ takes a functionality $\varphi \in \Phi_\lambda$ as input, and outputs a pair of master keys $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{KeyGen}(1^\lambda, \mathsf{msk}, f)$ takes as input the master secret key msk and a function description $f \in F_\varphi$. It outputs a secret key sk_f.
- $\mathsf{Enc}(1^\lambda, \mathsf{mpk}, x, y)$ takes as input the master public key mpk, a public input $x \in X_\varphi$, and a private input $y \in Y_\varphi$. It outputs a ciphertext ct_x.

- $\mathsf{Dec}^{f,x,\mathsf{sk}_f,\mathsf{ct}_x}(1^\lambda, \mathsf{mpk})$ takes the master public key mpk as input and is given random access to the function description f, the public input x, the secret key sk_f, and the ciphertext ct_x. It is supposed to compute z in $\varphi(f,x,y) = (T,z)$ efficiently.

The scheme is *correct* if for all $\lambda \in \mathbb{N}$, $\varphi \in \Phi_\lambda$, $f \in F_\varphi$, $x \in X_\varphi$, $y \in Y_\varphi$ such that $\varphi(f,x,y) = (T,z) \neq \perp$, it holds that

$$\Pr\left[\begin{array}{l}(\mathsf{mpk},\mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\varphi)\\ \qquad \mathsf{sk}_f \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,\mathsf{msk},f) \ : \ \mathsf{Dec}^{f,x,\mathsf{sk}_f,\mathsf{ct}_x}(1^\lambda,\mathsf{mpk}) = z\\ \qquad \mathsf{ct}_x \xleftarrow{\$} \mathsf{Enc}(1^\lambda,\mathsf{mpk},x,y)\end{array}\right] = 1.$$

Definition 6 (PHFE efficiency)**.** The basic efficiency requirements of a PHFE scheme (Definition 5) are as follows:

- Setup, KeyGen, Enc are polynomial-time.
- Dec runs in time $\mathrm{poly}(\lambda, |\varphi|, |f|, |x|, |y|, T)$, where $\varphi(M,f,x,y) = (T,z) \neq \perp$.

The following time-efficiency properties are considered for a PHFE:

- It has *linear-time* KeyGen [resp. Enc] if KeyGen [resp. Enc] runs in time $|f|\,\mathrm{poly}(\lambda,|\varphi|)$ [resp. $(|x|+|y|)\,\mathrm{poly}(\lambda,|\varphi|)$];
- It has $(T^{e_T} + |f|^{e_f} + |x|^{e_x} + |y|^{e_y})$*-time* Dec (for constants e_T, e_f, e_x, e_y) if Dec runs in time

  $$(T^{e_T} + |f|^{e_f} + |x|^{e_x} + |y|^{e_y})\,\mathrm{poly}(\lambda,|\varphi|),$$

  where $\varphi(M,f,x,y) = (T,z) \neq \perp$. Furthermore, the scheme has f*-fast* [resp. x*-fast*, y*-fast*] Dec if it has $(T^{e_T} + |f|^{e_f} + |x|^{e_x} + |y|^{e_y})$-time Dec with $e_f = 0$ [resp. $e_x = 0$, $e_y = 0$].

The following size-efficiency properties are considered for PHFE:

- It is f*-succinct* if $|\mathsf{sk}_f| = \mathrm{poly}(\lambda,|\varphi|)$, independent of $|f|$.
- It is x*-succinct* if $|\mathsf{ct}_x| = \mathrm{poly}(\lambda,|\varphi|,|y|)$, independent of $|x|$. Furthermore, it has *rate-c ciphertext* for some constant c if $|\mathsf{ct}_x| = \mathrm{poly}(\lambda,|\varphi|) + c|y|$;

We consider adaptive IND-CPA for polynomially bounded T:

Definition 7 (PHFE security)**.** A PHFE scheme (Definition 5) is *(adaptively IND-CPA) secure* if $\mathsf{Exp}^0_{\mathrm{PHFE}} \approx \mathsf{Exp}^1_{\mathrm{PHFE}}$, where $\mathsf{Exp}^b_{\mathrm{PHFE}}(1^\lambda)$ with adversary $\mathcal{A}$ proceeds as follows:

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive from it a functionality $\varphi \in \Phi_\lambda$ and a time bound $1^{\overline{T}}$. Run

  $$(\mathsf{mpk},\mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\varphi)$$

  and send mpk to $\mathcal{A}$.
- **Query I.** Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. In each round, $\mathcal{A}$ submits a function description $f_q \in F_\varphi$. Upon receiving the query, run

  $$\mathsf{sk}_q \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,\mathsf{msk},f_q)$$

  and send sk_q to $\mathcal{A}$.

- **Challenge.** $\mathcal{A}$ submits $x \in X_\varphi, y_0, y_1 \in Y_\varphi$. Upon the challenge, run

$$\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \mathsf{mpk}, x, y_b)$$

  and send ct to $\mathcal{A}$.

- **Query II.** Same as Query I.

- **Guess.** $\mathcal{A}$ outputs a bit b'. The outcome of the experiment is b' if

$$\begin{aligned} &|y_0| = |y_1|, \\ &\text{and } \varphi(f_q, x, y_0) = \varphi(f_q, x, y_1) = (T_q, z_q) \neq \bot && \text{for all } q, \\ &\text{and } T_q \leq \overline{T} && \text{for all } q. \end{aligned}$$

  Otherwise, the outcome is set to 0.

We will use FE for circuits as a building block:

Definition 8 (FE for circuits)**.** An *FE scheme for circuits* is a PHFE scheme (Definition 5) for

$$\begin{aligned} \Phi &= \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}, \qquad \Phi_\lambda = \{\varphi_{\ell,s}\}_{\ell,s \in \mathbb{N}_+}, \\ \varphi_{\ell,s} &: F_{\ell,s} \times X \times Y_\ell \to \{\bot\} \cup (\mathbb{N}_+ \times Z), \\ F_{\ell,s} &= \{\text{ circuits of input length } \ell \text{ and size at most } s\}, \\ X &= \{\bot\}, \qquad Y_\ell = \{0,1\}^\ell, \qquad Z = \{0,1\}^*, \\ \varphi_{\ell,s}(f, \bot, y) &= \big(1, f(y)\big), \end{aligned}$$

where the functionality $\varphi_{\ell,s}$ is represented by $(1^\ell, 1^s)$.

We remark that the first output of $\varphi_{\ell,s}$ is just a dummy value and all efficiency parameters (Definition 6) are always allowed arbitrary polynomial dependency on λ, ℓ, s by our choice of representing $\varphi_{\ell,s}$ by $(1^\ell, 1^s)$. This is intended as we use FE for circuits as a building block and do not wish to start with too strong a scheme. In fact, thanks to many beautiful prior works in the literature, we can further weaken the assumption all the way down to an "obfuscation-minimum FE" with only polynomial security.

Lemma 1 ([KNTY19])**.** *Assuming the existence of "weakly selective secure, 1-key, sublinearly succinct FE for circuits" (per definitions in [KNTY19]), i.e., so-called "obfuscation-minimum FE with only polynomial security", there exists an FE scheme for circuits (Definition 8) with adaptive IND-CPA security (Definition 7).*

2.4 Universal RAM and PHFE for RAM

In this section, we define PHFE for RAM after explaining some rationales.

We will employ the standard transformation [QWW18] of using FE for circuits to compute LGRAM to obtain PHFE for RAM. However, in LGRAM (Definition 2), the encryption time, as well as the ciphertext size, could depend on the size of the machine M. This dependency is inherited by the key size of the resultant PHFE for RAM if we associate each key with a RAM. To make the scheme f-succinct, we fix some universal RAM U upon setting up the scheme, and associate with each key a piece of

code interpreted by U. The size of U is some fixed $\mathrm{poly}(\lambda)$,[10] and so is the LGRAM garbling time. This translates to f-succinctness in the constructed PHFE for RAM.

The other issue is that LGRAM puts an upper bound on the running time and its incorrectness in case of exceeding time limit is propagated to the PHFE scheme. We avoid it[11] by defining $\varphi = \bot$ if the running time exceeds some super-polynomial value prescribed upon set-up.

The above explains the intended usage of our PHFE for RAM, yet we define it for general RAM. Moreover, as a stepping stone, we will first consider PHFE for RAM with *bounded private input*, where the private input is simply the short input to the RAM:

Definition 9 (PHFE for RAM with bounded private input)**.** A *PHFE scheme for RAM with bounded private input* is a PHFE scheme (Definition 5) for

$$\begin{aligned}
\Phi = \{\Phi_\lambda\}_{\lambda\in\mathbb{N}}, \qquad \Phi_\lambda &= \{\varphi_{M,T_{\max}}\}_{M\in\mathsf{RAM}_2,\, T_{\max}\in\mathbb{N}^*},\\
\varphi_{M,T_{\max}} &: F_M \times X_M \times Y_M \to \{\bot\} \cup (\mathbb{N}_+ \times Z),\\
F_M = X_M &= (\{0,1\}^{\ell_{\mathrm{cell}}})^{\le 2^{\ell_{\mathrm{addr}}}}, \qquad Y_M = \{0,1\}^{\ell_{\mathrm{in}}}, \qquad Z = \{\bot,0,1\}^*,\\
\varphi_{M,T_{\max}}(f,x,y) &= \begin{cases} \big(T, \mathsf{outS}(M,f,x,y)\big), & \text{if } \mathsf{time}(M,f,x,y) = T \le T_{\max};\\ \bot, & \text{otherwise;}\end{cases}
\end{aligned}$$

where $\varphi_{M,T_{\max}}$ is represented by $(M, T_{\max})$.

In a full-fledged PHFE for RAM, the RAM no longer has a short input, and the private input is encoded on a tape:

Definition 10 (full-fledged PHFE for RAM)**.** A *full-fledged PHFE scheme for RAM* is a PHFE scheme (Definition 5) for

$$\begin{aligned}
\Phi = \{\Phi_\lambda\}_{\lambda\in\mathbb{N}}, \qquad \Phi_\lambda &= \{\varphi_{M,T_{\max}}\}_{M\in\mathsf{RAM}_2 \text{ (with } \ell_{\mathrm{in}}=0)\text{ and } T_{\max}\in\mathbb{N}^*},\\
\varphi_{M,T_{\max}} &: F_M \times X_M \times Y_M \to \{\bot\} \cup (\mathbb{N}_+ \times Z),\\
F_M = X_M = Y_M &= (\{0,1\}^{\ell_{\mathrm{cell}}})^{\le 2^{\ell_{\mathrm{addr}}}}, \qquad Z = \{\bot,0,1\}^*,\\
\varphi_{M,T_{\max}}(f,x,y) &= \begin{cases} \big(T, \mathsf{outS}(M,f,x\|y,\varepsilon)\big), & \\ \qquad \text{if } |x|+|y| \le 2^{\ell_{\mathrm{addr}}} \text{ and } \mathsf{time}(M,f,x\|y,\varepsilon) = T \le T_{\max};\\ \bot, \quad \text{otherwise;}\end{cases}
\end{aligned}$$

where ε is the empty string and $\varphi_{M,T_{\max}}$ is represented by $(M, T_{\max})$.

2.5 Indistinguishability Obfuscation

We will use indistinguishability obfuscation for circuits *with polynomial-sized domains* as a building block.

[10] U is not a fixed RAM — its input address length should be $\omega(\log\lambda)$ to accommodate all polynomially long input.

[11] An alternative solution is to blatantly reveal everything if the running time is too large so that correctness in that case can be implemented by executing the machine in the clear. Security is not affected because the adversary is not allowed to choose keys and ciphertexts with super-polynomial instance running time. However, non-halting computation still needs to be handled separately.

Definition 11 ($i\mathcal{O}$)**.** An *indistinguishability obfuscator* is an efficient algorithm $i\mathcal{O}(1^\lambda, C)$ taking a circuit C as input. It outputs an obfuscated circuit $\widetilde{C}$.

The obfuscator must be *correct,* i.e., for all $\lambda \in \mathbb{N}$, circuit $C : \{0,1\}^n \to \{0,1\}^m$, input $x \in \{0,1\}^n$, it holds that

$$\Pr\left[i\mathcal{O}(1^\lambda, C)(x) = C(x)\right] = 1.$$

The obfuscator is *secure for polynomial-sized domains* if for all polynomial-sized $(1^{2^n}, C_0, C_1)$ such that $C_0, C_1 : \{0,1\}^n \to \{0,1\}^m$ have identical truth tables and sizes, it holds that

$$\left\{1^\lambda, C_0, C_1, i\mathcal{O}(1^\lambda, C_0)\right\} \approx \left\{1^\lambda, C_0, C_1, i\mathcal{O}(1^\lambda, C_1)\right\}.$$

We remark that the above definition does not allow the obfuscator to work by simply outputting the truth table, as the constraint of having polynomial-sized domains only applies to security, not efficiency nor correctness. Such an obfuscator is implied by the existence of FE for circuits (with polynomial security loss), either via a tight security reduction of FE to $i\mathcal{O}$ transformation [LT17] or via decomposable obfuscation [LZ17]:

Lemma 2 ([LT17,LZ17])**.** *If there exists a secure FE for circuits (Definition 8), there exists an indistinguishability obfuscator for circuits that is secure for polynomial-sized domains.*

2.6 Laconic Oblivious Transfer

Definition 12 (LOT [CDG+17])**.** A *laconic oblivious transfer scheme* consists of 4 algorithms:

- $\mathsf{HashGen}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}})$ takes as input the cell length $1^{\ell_{\mathsf{cell}}}$ and the address length $1^{\ell_{\mathsf{addr}}}$, and outputs a hash key hk. It runs in polynomial time.
- $\mathsf{Hash}(1^\lambda, \mathsf{hk}, D)$ takes as input the hash key hk and a database $D \in (\{0,1\}^{\ell_{\mathsf{cell}}})^{\leq 2^{\ell_{\mathsf{addr}}}}$, and *deterministically* outputs a hash h and a processed database $\widehat{D}$. It runs in time $|D|\,\mathrm{poly}(\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}})$, and the hash length is $\ell_{\mathsf{hash}} = \mathrm{poly}(\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}})$.
- $\mathsf{SendRead}(1^\lambda, \mathsf{hk}, h, i, \{m_j^b\}_{j\in[\ell_{\mathsf{cell}}]}^{b\in\{0,1\}})$ takes as input the hash key hk, a hash h, an address $i \in [2^{\ell_{\mathsf{addr}}}]$, and ℓ_{cell} pairs of messages of identical length. It outputs a read ciphertext rct. The algorithm runs in polynomial time.
- $\mathsf{RecvRead}^{\widehat{D}}(1^\lambda, \mathsf{hk}, h, i, \mathsf{rct})$ takes as input the hash key hk, the hash h, the address i, and the read ciphertext rct. Given random access to the processed database $\widehat{D}$, it is supposed to recover $\{m_j^{D[i][j]}\}_{j\in[\ell_{\mathsf{cell}}]}$. The algorithm runs in time $\mathrm{poly}(\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}}, |\mathsf{rct}|)$.

The scheme must be *correct,* i.e., for all $\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}} \in \mathbb{N}$, $D \in (\{0,1\}^{\ell_{\mathsf{cell}}})^{\leq 2^{\ell_{\mathsf{addr}}}}$, $i \in [|D|]$, $\{m_j^b\}_{j\in[\ell_{\mathsf{cell}}]}^{b\in\{0,1\}}$ of identical length, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}) \\ (h, \widehat{D}) \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, D) \\ \mathsf{rct} \xleftarrow{\$} \mathsf{SendRead}(1^\lambda, \mathsf{hk}, h, i, \{m_j^b\}_{j\in[\ell_{\mathsf{cell}}]}^{b\in\{0,1\}}) \\ \qquad : \mathsf{RecvRead}^{\widehat{D}}(1^\lambda, \mathsf{hk}, h, i, \mathsf{rct}) = \{m_j^{D[i][j]}\}_{j\in[\ell_{\mathsf{cell}}]} \end{array}\right] = 1.$$

An *updatable* LOT has 2 additional algorithms:

- $\mathsf{SendWrite}(1^\lambda, \mathsf{hk}, h, i, \mathsf{wdata}, \{m_j^b\}_{j\in[\ell_{\mathsf{hash}}]}^{b\in\{0,1\}})$ takes as input the hash key hk, the hash h, the address i, an overwriting string $\mathsf{wdata} \in \{0,1\}^{\ell_{\mathsf{cell}}}$, and ℓ_{hash} pairs of messages of identical length. It outputs a write ciphertext wct. The algorithm runs in polynomial time.
- $\mathsf{RecvWrite}^{\widehat{D}}(1^\lambda, \mathsf{hk}, h, i, \mathsf{wdata}, \mathsf{wct})$ takes as input the hash key hk, the hash h, the address i, the overwriting string wdata, and the write ciphertext wct. Given random access to the processed database $\widehat{D}$, it is supposed to update $\widehat{D}$ to $\widehat{D}'$ and recover $\{m_j^{h'[j]}\}_{j\in[\ell_{\mathsf{hash}}]}$, where D' is D with $D'[i]$ replaced by wdata and h' is the hash of D'. The algorithm runs in time $\mathrm{poly}(\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}}, |\mathsf{wct}|)$.

It is required that for all $\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}} \in \mathbb{N}$, $D \in (\{0,1\}^{\ell_{\mathsf{cell}}})^{\le 2^{\ell_{\mathsf{addr}}}}$, $i \in [|D|]$, $\mathsf{wdata} \in \{0,1\}^{\ell_{\mathsf{cell}}}$, $\{m_j^b\}_{j\in[\ell_{\mathsf{hash}}]}^{b\in\{0,1\}}$ of identical length, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}) \\ (h, \widehat{D}) \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, D), \qquad (h', \widehat{D}') \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, D') \\ \quad \mathsf{wct} \xleftarrow{\$} \mathsf{SendWrite}(1^\lambda, \mathsf{hk}, h, i, \mathsf{wdata}, \{m_j^b\}_{j\in[\ell_{\mathsf{hash}}]}^{b\in\{0,1\}}) \\ \qquad : \mathsf{RecvWrite}^{\widehat{D}}(1^\lambda, \mathsf{hk}, h, i, \mathsf{wdata}, \mathsf{wct}) = \{m_j^{h'[j]}\}_{j\in[\ell_{\mathsf{hash}}]} \\ \qquad\quad \text{and } \widehat{D} \text{ is updated to } \widehat{D}' \text{ by } \mathsf{RecvWrite} \end{array}\right] = 1,$$

where D' is D with $D'[i]$ replaced by wdata.

Part of our LGRAM garbling procedure involves hashing the all-zero string (the initial working tape) under a newly generated LOT hash key. It must be done very efficiently:

Definition 13 (LOT fast initialization)**.** An LOT scheme (Definition 12) has *fast initialization* if there is an efficient algorithm $\mathsf{Hash0s}(1^\lambda, \mathsf{hk}, S)$ computing the hash of $0^{\ell_{\mathsf{cell}}S}$ given the hash key and the length (in binary). More precisely, for all $\lambda, \ell_{\mathsf{cell}}, \ell_{\mathsf{addr}} \in \mathbb{N}$, $S \in [2^{\ell_{\mathsf{addr}}}]$, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}) \\ (h, \widehat{D}) \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, 0^{\ell_{\mathsf{cell}}S}) \end{array} : \mathsf{Hash0s}(1^\lambda, \mathsf{hk}, S) = h\right] = 1.$$

The generic bootstrapping procedure [CDG+17,AL18,KNTY19] for LOT using Merkle tree always yields a scheme with fast initialization, because the Merkle hash of an all-zero string of length S can be computed in time $\mathrm{poly}(\lambda, \log S)$.

Security. Following [KNTY19], we consider database-selective security for LOT.

Definition 14 (LOT read security [CDG+17])**.** An LOT scheme (Definition 12) is *read-secure* if there exists an efficient simulator $\mathsf{SimRead}$ such that for all polynomial-sized $1^{\ell_{\mathsf{cell}}}$, $1^{\ell_{\mathsf{addr}}}$, $D \in (\{0,1\}^{\ell_{\mathsf{cell}}})^{\le 2^{\ell_{\mathsf{addr}}}}$, $i \in [|D|]$, $\{m_j^b\}_{j\in[\ell_{\mathsf{cell}}]}^{b\in\{0,1\}}$ of identical length, it holds that

$$\begin{aligned} &\{1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}, \mathsf{hk}, D, i, \{m_j^b\}_{j\in[\ell_{\mathsf{cell}}]}^{b\in\{0,1\}}, \mathsf{rct}\} \approx \{\cdots, \widetilde{\mathsf{rct}}\}, \text{ where} \\ &\quad \mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}(1^\lambda, 1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}), \quad (h, \widehat{D}) \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, D), \\ &\quad \mathsf{rct} \xleftarrow{\$} \mathsf{SendRead}(1^\lambda, \mathsf{hk}, h, i, \{m_j^b\}_{j\in[\ell_{\mathsf{cell}}]}^{b\in\{0,1\}}), \\ &\quad \widetilde{\mathsf{rct}} \xleftarrow{\$} \mathsf{SimRead}(1^\lambda, \mathsf{hk}, D, i, \{m_j^{D[i][j]}\}_{j\in[\ell_{\mathsf{cell}}]}). \end{aligned}$$

Definition 15 (LOT write security [CDG+17])**.** An updatable LOT scheme (Definition 12) is *write-secure* if there exists an efficient simulator SimWrite such that for all polynomial-sized $1^{\ell_{\mathrm{cell}}}$, $1^{\ell_{\mathrm{addr}}}$, $D \in (\{0,1\}^{\ell_{\mathrm{cell}}})^{\leq 2^{\ell_{\mathrm{addr}}}}$, $i \in [|D|]$, $\mathsf{wdata} \in \{0,1\}^{\ell_{\mathrm{cell}}}$, $\{m_j^b\}_{j\in[\ell_{\mathrm{hash}}]}^{b\in\{0,1\}}$ of identical length, letting D' be D with $D'[i]$ replaced by wdata, it holds that

$$\begin{aligned}
\{1^\lambda, 1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}, \mathsf{hk}, D, i, \mathsf{wdata}, \{m_j^b\}_{j\in[\ell_{\mathrm{hash}}]}^{b\in\{0,1\}}, \mathsf{wct}\} \approx \{\cdots, \widetilde{\mathsf{wct}}\}, \text{ where}\\
\mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}(1^\lambda, 1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}),\\
(h, \widehat{D}) \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, D), \quad (h', \widehat{D}') \leftarrow \mathsf{Hash}(1^\lambda, \mathsf{hk}, D'),\\
\mathsf{wct} \xleftarrow{\$} \mathsf{SendWrite}\big(1^\lambda, \mathsf{hk}, h, i, \mathsf{wdata}, \{m_j^b\}_{j\in[\ell_{\mathrm{hash}}]}^{b\in\{0,1\}}\big),\\
\widetilde{\mathsf{wct}} \xleftarrow{\$} \mathsf{SimWrite}\big(1^\lambda, \mathsf{hk}, D, i, \mathsf{wdata}, \{m_j^{h'[j]}\}_{j\in[\ell_{\mathrm{hash}}]}\big).
\end{aligned}$$

We remark that the above definitions is also index-selective and message-selective. Without loss of generality, we may assume that security holds even if the adversary is allowed to choose i and m_j^b's adaptively (depending on hk). Adaptive security with respect to i can be obtained by a standard guessing argument as noted in [GOS18], and that with respect to m_j^b's by using the scheme as a key encapsulation mechanism (or by encrypting bit by bit and following a standard hybrid argument).

Lemma 3 ([LZ17,CDG+17,AL18,KNTY19])**.** *Assuming the existence of FE for circuits (Definition 8), there exists an updatable LOT with fast initialization that is both read-secure and write-secure.*

2.7 Garbled Circuits

Following the formulation in [GS18a], we make the garbling procedure to take the labels as input instead of letting it generate them. However, their formulation cannot be perfectly correct, which we fix by incorporating the point-and-permute [BMR90] technique:

Definition 16 (garbled circuits [Yao82])**.** A *circuit garbling scheme* (with label length $\ell_L(\lambda)$) consists of 2 efficient algorithms:

- $\mathsf{Garble}(1^\lambda, C, \pi, \{L_{i,b}\}_{i\in[n],b\in\{0,1\}})$ takes as input a circuit C of input length n, a point-and-permute string $\pi \in \{0,1\}^n$, and n pairs of labels, each of length $\ell_L(\lambda)$. It outputs a garbled circuit $\widehat{C}$.
- $\mathsf{Eval}(1^\lambda, \widehat{C}, x \oplus \pi, \{L_i\}_{i\in[n]})$ takes as input the garbled circuit $\widehat{C}$, the permuted input, and n labels. It is supposed to compute $C(x)$.

The scheme must be *correct*, i.e., for all $\lambda, n \in \mathbb{N}$, circuit C of input length n, input $x \in \{0,1\}^n$, it holds that

$$\Pr\left[\begin{array}{l}\pi \xleftarrow{\$} \{0,1\}^n\\ L_{i,b} \xleftarrow{\$} \{0,1\}^{\ell_L(\lambda)} \text{ for } i \in [n], b \in \{0,1\}\\ \widehat{C} \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C, \pi, \{L_{i,b}\}_{i\in[n],b\in\{0,1\}})\\ y \leftarrow \mathsf{Eval}(1^\lambda, \widehat{C}, x \oplus \pi, \{L_{i,x[i]}\}_{i\in[n]})\end{array} : y = C(x)\right] = 1.$$

The scheme is *secure* if there exists a simulator $\widetilde{\mathsf{Garble}}$ such that for all polynomial-sized C, x, the following distributions are indistinguishable:

$$\begin{aligned}&\big\{1^\lambda, C, x, \mathsf{Garble}(1^\lambda, C, \quad\quad\quad\quad \pi, \{L_{i,b} \quad\}_{i\in[n],b\in\{0,1\}}), x\oplus\pi, \{L_{i,x[i]}\}_{i\in[n]}\big\}\\ \approx\ &\big\{1^\lambda, C, x, \widetilde{\mathsf{Garble}}(1^\lambda, 1^{|C|}, C(x), x\oplus\pi, \{L_{i,x[i]}\}_{i\in[n]} \quad\quad), x\oplus\pi, \{L_{i,x[i]}\}_{i\in[n]}\big\},\end{aligned}$$

where $L_{i,b} \xleftarrow{\$} \{0,1\}^{\ell_L(\lambda)}$ for all $i \in [n], b \in \{0,1\}$ and $\pi \xleftarrow{\$} \{0,1\}^n$.

2.8 Puncturable Pseudorandom Function

We need puncturable pseudorandom functions as a building block.[12]

Definition 17 (puncturable PRF [BW13])**.** A *puncturable pseudorandom function* with key (resp. input, output) length $\ell_{\text{key}}(\lambda)$ (resp. $\ell_{\text{in}}(\lambda)$, $\ell_{\text{out}}(\lambda)$; all polynomial in λ) consists of 2 efficient algorithms:

- $\mathsf{Eval}(1^\lambda, k, x)$ takes as input a (punctured or not) key k, an input $x \in \{0,1\}^{\ell_{\text{in}}(\lambda)}$, and *deterministically* outputs a bit-string of length $\ell_{\text{out}}(\lambda)$.
- $\mathsf{Puncture}(1^\lambda, k, S)$ takes as input a non-punctured key $k \in \{0,1\}^{\ell_{\text{key}}(\lambda)}$ and a set $S \subseteq \{0,1\}^{\ell_{\text{in}}(\lambda)}$. It outputs a punctured key $\mathring{k}_S$.

The scheme must be *correct*, i.e., for all $\lambda \in \mathbb{N}$, $k \in \{0,1\}^{\ell_{\text{key}}(\lambda)}$, $S \subseteq \{0,1\}^{\ell_{\text{in}}(\lambda)}$, $x \in \{0,1\}^{\ell_{\text{in}}(\lambda)} \setminus S$, it holds that

$$\Pr\big[\mathsf{Eval}(1^\lambda, \mathsf{Puncture}(1^\lambda, k, S), x) = \mathsf{Eval}(1^\lambda, k, x)\big] = 1.$$

The scheme is *secure* if for all polynomial-sized $S \subseteq \{0,1\}^{\ell_{\text{in}}(\lambda)}$, it holds that

$$\begin{aligned}&\big\{1^\lambda, S, \mathring{k}_S, \{\mathsf{Eval}(1^\lambda, k, s)\}_{s\in S}\big\} \approx \big\{1^\lambda, S, \mathring{k}_S, \{r_s\}_{s\in S}\big\}, \text{ where}\\ &\qquad k \xleftarrow{\$} \{0,1\}^{\ell_{\text{key}}(\lambda)}, \quad \mathring{k}_S \xleftarrow{\$} \mathsf{Puncture}(1^\lambda, k, S),\\ &\qquad r_s \xleftarrow{\$} \{0,1\}^{\ell_{\text{out}}(\lambda)} \text{ for all } s \in S.\end{aligned}$$

2.9 Secret-Key Encryption

We need secret-key encryption as a building block.

Definition 18 (SKE)**.** A *secret-key encryption scheme* consists of 3 efficient algorithms

- $\mathsf{Gen}(1^\lambda)$ outputs a key k.
- $\mathsf{Enc}(1^\lambda, k, m)$ takes as input a key k and a message m of arbitrary length, and outputs a ciphertext c.
- $\mathsf{Dec}(1^\lambda, k, c)$ takes as input a key k and a ciphertext c. It is supposed to recover the message.

[12] We also use selectively secure usual PRF, yet omit it here as it is implied by PPRF.

The scheme must be *correct,* i.e., for all $\lambda \in \mathbb{N}$ and $m \in \{0,1\}^*$, it holds that

$$\Pr\left[\, k \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \;:\; \mathsf{Dec}\big(1^\lambda, \mathsf{Enc}(1^\lambda, k, m)\big) = m \,\right] = 1.$$

It has *pseudorandom ciphertexts* if the ciphertext length is a function of λ and the message length (independent of key generation and encryption randomness) and for all $\lambda \in \mathbb{N}$ and polynomially long sequence of messages $\{m_q\}_{q\in[Q]}$, it holds that

$$\big\{1^\lambda, 1^Q, \{m_q, c_q\}_{q\in[Q]}\big\} \approx \big\{1^\lambda, 1^Q, \{m_q, r_q\}_{q\in[Q]}\big\}, \text{ where}$$
$$k \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \quad c_q \xleftarrow{\$} \mathsf{Enc}(1^\lambda, k, m_q), \quad r_q \xleftarrow{\$} \{0,1\}^{|c_q|} \qquad \text{for all } q \in [Q].$$

2.10 Oblivious RAM

We define ORAM as a mechanism to translate logical accesses into physical accesses. It separates the logic of protecting memory access from the computation performed. Compared to defining it as an algorithm transforming an underlying RAM to an ORAM'ed RAM, we avoid having to incorporate randomness[13] into the definition of RAM. We also add error (overflow) checking into our definition so that our LGRAM and PHFE can be made perfectly correct.

Definition 19 (ORAM)**.** An *oblivious RAM scheme* is an efficient deterministic algorithm

$$\mathsf{MakeORAM}(1^\lambda, 1^{\ell_{\mathrm{CELL}}}, 1^{\ell_{\mathrm{ADDR}}}, T_{\max})$$

taking as input the logical cell length $1^{\ell_{\mathrm{CELL}}}$, the logical address length $1^{\ell_{\mathrm{ADDR}}}$, and a time bound $T_{\max} \in \mathbb{N}$. It outputs a physical space bound $S'_{\max}$, a physical cell length $1^{\ell'_{\mathrm{CELL}}}$, an ORAM randomness length $1^{\ell_{\mathrm{r}}}$, an ORAM state length $1^{\ell_{\mathrm{ost}}}$, and a sequence of circuits $\{\mathsf{oRW}_{t_0}\}_{t_0\in[T_0]}$ satisfying the following conditions:

- $\ell'_{\mathrm{CELL}} \le \mathrm{poly}(\lambda, \ell_{\mathrm{CELL}}, \ell_{\mathrm{ADDR}}, \log T_{\max})$.
- $S'_{\max} \le T_{\max}\,\mathrm{poly}(\lambda, \ell_{\mathrm{CELL}}, \ell_{\mathrm{ADDR}}, \log T_{\max})$.
- $T_0 \le \mathrm{poly}(\lambda, \ell_{\mathrm{CELL}}, \ell_{\mathrm{ADDR}}, \log T_{\max})$ is the number of physical steps to perform one logical access.
- The circuits have the following input/output syntax:

$$\begin{aligned}
&\mathsf{oRW}_1 : && (i, \mathsf{wdata}, r_1) \mapsto (\mathsf{ost}_1, i'_1, \mathsf{wdata}'_1) && \text{or } \perp, && \\
&\mathsf{oRW}_{t_0} : && (\mathsf{ost}_{t_0-1}, \mathsf{rdata}'_{t_0-1}, r_{t_0}) \mapsto (\mathsf{ost}_{t_0+1}, i'_{t_0+1}, \mathsf{wdata}'_{t_0+1}) && \text{or } \perp, && \text{for } 1 < t_0 < T_0, \\
&\mathsf{oRW}_{T_0} : && (\mathsf{ost}_{T_0-1}, \mathsf{rdata}'_{T_0-1}, r_{T_0}) \mapsto (i'_{T_0}, \mathsf{wdata}'_{T_0}, \mathsf{rdata}) && \text{or } \perp. &&
\end{aligned}$$

  Here,

  - $i \in [2^{\ell_{\mathrm{ADDR}}}]$ is the logical address to read from and write to,
  - $\mathsf{rdata} \in \{0,1\}^{\ell_{\mathrm{CELL}}}$ is the logical string that was read,

[13] Our LGRAM only works with deterministic RAM and dealing with garbling of randomized computation is cumbersome. An ORAM'ed machine is probabilistic and cannot be directly fed into LGRAM with lesser security to obtain LGRAM with better security, so the notion of ORAM'ed RAM does not simplify formalism and might lead to confusion (it bears the name of RAM yet cannot be used as an input to LGRAM). We choose to avoid this notion and define ORAM as a memory access translation mechanism.

- $\mathsf{wdata} \in \{0,1\}^{\ell_{\text{CELL}}}$ is the logical string to write,
- $r_1, \ldots, r_{T_0} \in \{0,1\}^{\ell_r}$ are the ORAM randomness,
- $\mathsf{ost}_1, \ldots, \mathsf{ost}_{T_0-1} \in \{0,1\}^{\ell_{\text{ost}}}$ are the ORAM states,
- $i'_1, \ldots, i'_{T_0} \in [S'_{\max}]$ are the physical addresses to read from and write to,
- $\mathsf{wdata}'_1, \ldots, \mathsf{wdata}'_{T_0} \in \{0,1\}^{\ell'_{\text{CELL}}}$ are the physical string to write.

The scheme must be *correct*, i.e., for all $\lambda, \ell_{\text{CELL}}, \ell_{\text{ADDR}}, T_{\max} \in \mathbb{N}$ and all sequence $\{(i_t, \mathsf{wdata}_t)\}_{t\in[T_{\max}]}$ of logical accesses, let

$$(S'_{\max}, 1^{\ell'_{\text{CELL}}}, 1^{\ell_{\text{ost}}}, \{\mathsf{oRW}_{t_0}\}_{t_0\in[T_0]}) \leftarrow \mathsf{MakeORAM}(1^\lambda, 1^{\ell_{\text{CELL}}}, 1^{\ell_{\text{ADDR}}}, T_{\max}),$$
$$r_{t,t_0} \xleftarrow{\$} \{0,1\}^{\ell_r} \qquad \text{for } t \in [T_{\max}], t_0 \in [T_0],$$

and consider the following process:

- Let $D_0 \leftarrow 0^{\ell_{\text{ADDR}}}$ and $D'_{0,0} \leftarrow 0^{S'_{\max}}$.
- For $t = 1, \ldots, T_{\max}$:
  - (Logical step.) Let $\mathsf{rdata}_t \leftarrow D_{t-1}[i_t]$ and let D_t be D_{t-1} with $D_t[i_t]$ replaced by wdata_t.
  - Perform T_0 physical steps by

$$\begin{aligned}
(\mathsf{ost}_1, i'_{t,1}, \mathsf{wdata}'_{t,1}) &\leftarrow \mathsf{oRW}_1(i_t, \mathsf{wdata}_t, r_{t,1}),\\
(\mathsf{ost}_{t_0}, i'_{t,t_0}, \mathsf{wdata}'_{t,t_0}) &\leftarrow \mathsf{oRW}_{t_0}(\mathsf{ost}_{t_0-1}, \mathsf{rdata}'_{t,t_0-1}, r_{t,t_0}) \qquad \text{for } 1 < t_0 < T_0,\\
(i'_{t,T_0}, \mathsf{wdata}'_{t,T_0}, \widetilde{\mathsf{rdata}}_t) &\leftarrow \mathsf{oRW}_{T_0}(\mathsf{ost}_{t,T_0-1}, \mathsf{rdata}'_{t,T_0-1}, r_{t,T_0}), \qquad \text{where}\\
D'_{t+1,0} \leftarrow D'_{t,T_0}, \qquad D'_{t,t_0} &\leftarrow D'_{t,t_0-1} \text{ with } D'_{t,t_0}[i'_{t,t_0}] \text{ replaced by } \mathsf{wdata}'_{t,t_0},\\
\mathsf{rdata}'_{t,t_0} &\leftarrow D'_{t,t_0}[i'_{t,t_0}].
\end{aligned}$$

It is required that (over the random choices of r's)

$$\Pr[\text{any of } \mathsf{oRW}_{t,t_0} \text{ outputs } \perp] \le 2^{-\lambda}, \qquad \text{and}$$
$$\Pr[(\text{some of } \mathsf{oRW}_{t,t_0} \text{ outputs } \perp) \vee (\mathsf{rdata}_t = \widetilde{\mathsf{rdata}}_t \text{ for all } t \in [T_{\max}])] = 1.$$

In our definition, error is indicated by $\perp$ from oRW_{t,t_0}'s. The correctness requirements postulates that it is negligibly likely that any error is reported and that every logical access is perfectly fulfilled until the first error is reported. The latter ensures that errors are always caught before they can corrupt computation.

Security. Following [CH16], we need an ORAM with localized randomness.

Definition 20 (ORAM localized randomness)**.** An ORAM scheme (Definition 19) has *localized randomness* if there exist efficient deterministic algorithms $\mathsf{PartRnd}, \mathsf{SimORAM}$ such that for all $\lambda, \ell_{\text{CELL}}, \ell_{\text{ADDR}}, T_{\max} \in \mathbb{N}$ and all sequence $\{(i_t, \mathsf{wdata}_t)\}_{t\in[T]}$ of logical accesses with $T \in [T_{\max}]$,

$$\mathsf{PartRnd}(1^\lambda, 1^{\ell_{\text{CELL}}}, 1^{\ell_{\text{ADDR}}}, T_{\max}, \{(i_t, \mathsf{wdata}_t)\}_{t\in[T]}) \to \{R_t\}_{t\in[T]}$$

satisfies

$$R_t \subseteq [T] \times [T_0] \text{ for all } t \in [T], \qquad \max_{t\in[T]} |R_t| \le \mathrm{poly}(\lambda, \ell_{\mathrm{CELL}}, \ell_{\mathrm{ADDR}}, \log T_{\max}),$$
$$\Pr\left[\begin{array}{r}\mathsf{SimORAM}(1^\lambda, 1^{\ell_{\mathrm{CELL}}}, 1^{\ell_{\mathrm{ADDR}}}, T_{\max}, t, t_0, \{r_i\}_{i\in R_t}) \neq i'_{t,t_0} \\ \text{for some } t \in [T], t_0 \in [T_0]\end{array}\right] \le 2^{-\lambda},$$

where the probability is over r's and the notations follow those in Definition 19.

3 Efficiency Trade-Offs of PHFE for RAM

3.1 Contention Between Storage Overhead and Decryption Time

In this section, we show that it is impossible to achieve

$$|\mathsf{sk}| = O(|f|^\alpha) \quad \text{and} \quad T_{\mathsf{Dec}} = O(T + |f|^\beta + |x| + |y|)$$

simultaneously for a secure PHFE for RAM when $\alpha, \beta < 1$, where polynomial factors in the security parameter are ignored. This leaves us with two candidate optima:

- $\alpha = 0$ and $\beta = 1$ for succinct keys; or
- $\alpha = 1$ and $\beta = 0$ for f-fast decryption.

Similarly, it is impossible to achieve

$$|\mathsf{ct}| = O(|x|^\alpha)\,\mathrm{poly}(|y|) \quad \text{and} \quad T_{\mathsf{Dec}} = O(T + |f| + |x|^\beta + |y|)$$

simultaneously if $\alpha, \beta < 1$, which implies a contention between succinct ciphertexts and x-fast decryption.

Formally, our theorems are slightly stronger than the discussion above:

Theorem 4 (contention of $|f|$-dependency between $|\mathsf{sk}|$ and T_{Dec}; ¶)**.** *For a secure full-fledged PHFE for RAM (Definitions 5, 7, and 10), if*

$$|\mathsf{sk}| \le |f|^\alpha(\lambda + |\varphi|)^C \quad \text{and} \quad T_{\mathsf{Dec}} \le (T + |f|^\beta + |y|)(\lambda + |\varphi| + |x|)^C$$

for infinitely many λ, where α, β, C are constants, then $\alpha \ge 1$ or $\beta \ge 1$.

Theorem 5 (contention of $|x|$-dependency between $|\mathsf{ct}|$ and T_{Dec})**.** *For a secure full-fledged PHFE for RAM (Definitions 5, 7, and 10), if*

$$|\mathsf{ct}| \le |x|^\alpha(\lambda + |\varphi| + |y|)^C \quad \text{and} \quad T_{\mathsf{Dec}} \le (T + |f| + |x|^\beta)(\lambda + |\varphi| + |y|)^C$$

for infinitely many λ, where α, β, C are constants, then $\alpha \ge 1$ or $\beta \ge 1$.

We will only prove Theorem 4. The proof of Theorem 5 is similar.

Proof (Theorem 4). Let $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a secure PHFE for RAM. Suppose for contradiction that $\alpha, \beta < 1 - 5\varepsilon$ for some $\varepsilon > 0$. By enlarging C as needed, we could assume $|\varphi| \le \lambda^C - \lambda - 1$ for all sufficiently large λ, where

$$\varphi = (M_\lambda, 2^\lambda), \quad f = R \in \{0,1\}^{\le 2^\lambda}, \quad x = \perp,$$

$$y = \begin{cases} (I, w) = (i_1, w[1], \ldots, i_n, w[n]) \in ([2^\lambda] \times \{0,1\})^{\leq 2^\lambda}; \\ z = (\bot,\ z[1], \ldots, \bot,\ z[n]) \in (\{\bot\} \times \{0,1\})^{\leq 2^\lambda}; \end{cases}$$

$$M^{f,x\|y}() = \begin{cases} (R[i_1] \oplus w[1], \ldots, R[i_n] \oplus w[n]), & \text{if } y = (I, w); \\ (\quad z[1] \quad, \ldots, \quad z[n] \quad), & \text{if } y = z. \end{cases}$$

Under appropriate encoding and step circuit design, y has exactly n cells and M halts in exactly $(n+1)$ steps.

We focus on the values of λ (hereafter, "λ with efficiency") such that

$$|\mathsf{sk}| \leq |f|^\alpha (\lambda + |\varphi|)^C \quad \text{and} \quad T_{\mathsf{Dec}} \leq (T + |f|^\beta + |y|)(\lambda + |\varphi| + |x|)^C$$

By setting

$$|R| = N = \lceil \lambda^{(C^2+1)/\varepsilon} \rceil, \qquad n = \lfloor N^{1-3\varepsilon} \rfloor,$$

we would have $n < N < 2^\lambda$ for sufficiently large λ. Consider the following adversary $\mathcal{A}$ (Definition 7):

- Upon launching, it computes φ define above and N, n, sets up the PHFE scheme for φ, and submits 1^{n+1} as the time bound.
- It samples $R \xleftarrow{\$} \{0,1\}^N$ and requests a key sk for $f = R$.
- It samples $w \xleftarrow{\$} \{0,1\}^n$ and a list I of n distinct random elements from $[N]$, sets

  $$z = (R[i_1] \oplus w[1], \ldots, R[i_n] \oplus w[n]).$$

  It challenges with

  $$x = \bot, \qquad y_0 = (I, w), \quad y_1 = z,$$

  and obtains a ciphertext ct encrypting either y_0 or y_1.
- It runs $\mathsf{Dec}^{f,x,\mathsf{sk},\mathsf{ct}}(\mathsf{mpk})$ and notes down the list L of indices into $R = f$ where it is read during decryption. $\mathcal{A}$ outputs 1 if and only if

  $$|L \cap I| > N^{1-4\varepsilon},$$

  where L and I are regarded as sets (unordered and deduplicated) for the intersection operation.

Clearly, $\mathcal{A}$ would be efficient and its challenge would satisfy the constraints of PHFE security for sufficiently large λ. We claim:

Claim 6 (¶). *For sufficiently large λ with efficiency,*

$$\Pr\left[\, |L \cap I| > N^{1-4\varepsilon} \text{ in } \mathsf{Exp}^0_{\mathrm{PHFE}} \,\right] \geq \frac{3}{4}.$$

Claim 7 (¶). *For sufficiently large λ with efficiency,*

$$\Pr\left[\, |L \cap I| > N^{1-4\varepsilon} \text{ in } \mathsf{Exp}^1_{\mathrm{PHFE}} \,\right] \leq \frac{1}{4}.$$

The two claims together would contradict the security of PHFE, as the advantage of $\mathcal{A}$ would be at least $\frac{1}{2}$ for infinitely many λ. Therefore, $\alpha \geq 1$ or $\beta \geq 1$. □

To prove Claim 6, we need the following lemma about incompressibility of information:

Lemma 8 ([DTT10])**.** *Suppose $E : S \times U \to V$ and $D : S \times V \to U$ are functions and $\mathcal{S}$ is a distribution over S, then*

$$|V| \geq |U| \cdot \Pr_{\substack{s \xleftarrow{\$} \mathcal{S} \\ u \xleftarrow{\$} U}} [D(s, E(s,u)) = u].$$

Proof (Claim 6). We use the PHFE scheme to compress a string u of length n. To encode, we embed u into a string R of length N at random locations (i.e., I) and generate a PHFE key for R. The encoding is the key plus some bits in R used during decryption. To decode, run the decryption algorithm. Lemma 8 will generate the following inequality equivalent to the desired one:

$$\Pr\left[\, |L \cap I| \leq \lfloor N^{1-4\varepsilon} \rfloor \text{ in } \mathsf{Exp}^0_{\mathrm{PHFE}} \,\right] \leq \frac{1}{4}.$$

Formally, let

$$\mathcal{S} = \left\{ \begin{pmatrix} \mathsf{mpk}, \mathsf{msk}, I, w, R', \\ r_{\mathsf{KeyGen}}, r_{\mathsf{Enc}}, r_{\mathsf{Dec}} \end{pmatrix} : \begin{array}{r} (\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(\varphi) \\ (I, w) \text{ as how } \mathcal{A} \text{ samples it} \\ R'[i] \xleftarrow{\$} \{0,1\} \text{ for } i \in [N] \setminus I \\ r_{\mathsf{KeyGen}}, r_{\mathsf{Enc}}, r_{\mathsf{Dec}} \xleftarrow{\$} \text{algorithm randomness} \end{array} \right\},$$

$$U = \{0,1\}^n, \qquad V = \{0,1\}^{\lfloor N^{1-4\varepsilon} \rfloor} \times \{0,1\}^{\lfloor N^{1-4\varepsilon} \rfloor}.$$

The encoding procedure $E(s,u)$ works as follows:

- Parse $I = (i_1, \ldots, i_n)$ and set

$$R[i] = \begin{cases} R'[i], & \text{if } i \in [N] \setminus I; \\ u[j], & \text{if } i = i_j. \end{cases}$$

- Run

$$\begin{aligned} \mathsf{sk} &\leftarrow \mathsf{KeyGen}(\mathsf{msk}, R; r_{\mathsf{KeyGen}}), \\ \mathsf{ct} &\leftarrow \mathsf{Enc}(\mathsf{mpk}, \perp, (I, w); r_{\mathsf{Enc}}), \\ u \oplus w &\leftarrow \mathsf{Dec}^{R, \perp, \mathsf{sk}, \mathsf{ct}}(\mathsf{mpk}; r_{\mathsf{Dec}}), \end{aligned}$$

  and note down the list $L = (\ell_1, \ldots)$ of indices into R read by Dec.

- Output $v = (v_1, v_2)$ with $v_1, v_2 \in \{0,1\}^{\lfloor N^{1-4\varepsilon} \rfloor}$ and

$$v_1 = 0^{\lfloor N^{1-4\varepsilon} \rfloor - |\mathsf{sk}| - 1} 1 \| \mathsf{sk},$$

$$v_2[i] = \begin{cases} R[\ell_j], & \text{if } |\{\ell_1, \ldots, \ell_{j-1}\} \cap I| = i - 1 \text{ and } |\{\ell_1, \ldots, \ell_{j-1}, \ell_j\} \cap I| = i; \\ 0, & \text{if no such } j \text{ exists.} \end{cases}$$

Here, v_1 is a fixed-length encoding of sk and is indeed well-defined since

$$|\mathsf{sk}| \le |f|^{\alpha}(\lambda + |\varphi|)^C \le N^{1-5\varepsilon}\big(\lambda + (\lambda^C - \lambda - 1)\big)^C \le N^{1-5\varepsilon}\lambda^{C^2} < \lfloor N^{1-4\varepsilon}\rfloor - 1$$

for sufficiently large λ with efficiency. The string v_2 records, *sequentially*, the bits in R at each *distinct* index read by Dec that are part of u and not known from R', for at most $\lfloor N^{1-4\varepsilon}\rfloor$ bits.

The decoding procedure $D(s, v)$ works as follows:

- Run $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \bot, (I, w); r_{\mathsf{Enc}})$.
- Parse $v = (v_1, v_2)$ and recover sk from v_1 as specified in E.
- Initialize j, an index into v_2, by $j \leftarrow 0$, and initialize R by

$$R[i] = \begin{cases} R'[i], & \text{if } i \in [N] \setminus I; \\ \bot, & \text{if } i \in I. \end{cases}$$

  Run $z \leftarrow \mathsf{Dec}^{R,\bot,\mathsf{sk},\mathsf{ct}}(\mathsf{mpk}; r_{\mathsf{Dec}})$ with R filled on the fly. When Dec reads $R[i]$:

  - if $R[i] = \bot$ and $j < \lfloor N^{1-4\varepsilon}\rfloor$, then let $j \leftarrow j + 1$ and set $R[i] \leftarrow v_2[j]$;
  - if $R[i] = \bot$ and $j = \lfloor N^{1-4\varepsilon}\rfloor$, then abort by outputting 0^n;
  - otherwise, $R[i] \neq \bot$, then just proceed without aborting;

  and return $R[i]$ to Dec if not aborting.
- Output $z \oplus w$.

D will fill v_2 into the correct indices of R since Enc and Dec are derandomized with the same randomness as in E.

The sampling of s, u and the setting of R in $E(s, u)$ simulate $\mathcal{A}$ in $\mathsf{Exp}^0_{\mathrm{PHFE}}$. If s and u are such that $|L \cap I| \le \lfloor N^{1-4\varepsilon}\rfloor$ in $E(s, u)$, then D will successfully recover u. By Lemma 8,

$$\begin{aligned} \Pr\big[\, |L \cap I| \le \lfloor N^{1-4\varepsilon}\rfloor \text{ in } \mathsf{Exp}^0_{\mathrm{PHFE}} \,\big] &= \Pr_{\substack{s \xleftarrow{\$} \mathcal{S} \\ u \xleftarrow{\$} U}} \big[\, |L \cap I| \le \lfloor N^{1-4\varepsilon}\rfloor \text{ in } E(s, u) \,\big] \\ &\le \Pr_{\substack{s \xleftarrow{\$} \mathcal{S} \\ u \xleftarrow{\$} U}} [D(s, E(s, u)) = u] \\ &\le \frac{|V|}{|U|} = \frac{2^{2\lfloor N^{1-4\varepsilon}\rfloor}}{2^n} = 2^{2\lfloor N^{1-4\varepsilon}\rfloor - \lfloor N^{1-3\varepsilon}\rfloor} \le \frac{1}{4} \end{aligned}$$

for sufficiently large λ with efficiency. □

Proof (Claim 7). For sufficiently large λ with efficiency,

$$\begin{aligned} |L| \le T_{\mathsf{Dec}} &\le (T + |f|^{\beta} + |y|)(\lambda + |\varphi| + |x|)^C \\ &\le \big((n+1) + N^{1-5\varepsilon} + n\big)\big(\lambda + (\lambda^C - \lambda - 1) + 1\big)^C \\ &\le \big(2N^{1-3\varepsilon} + N^{1-5\varepsilon} + 1\big)\lambda^{C^2} \le N^{1-2\varepsilon}. \end{aligned}$$

In $\mathsf{Exp}^1_{\mathrm{PHFE}}$, the input to Dec is independent of I, which only symbolically appears in ct as

$$y_1 = z = (R[i_1] \oplus w[1], \ldots, R[i_n] \oplus w[n])$$

and is fully hidden by the one-time pad w. Therefore, the list of indices into R read by Dec (i.e., L) is independent of I. Conditioned on L, the intersection size $|L \cap I|$ follows a hypergeometric distribution. By the law of total expectation,

$$\mathbb{E}\big[\, |L \cap I| \,\big] = \mathbb{E}\Big[\mathbb{E}\big[\, |L \cap I| \,\big|\, L \,\big]\Big] = \mathbb{E}\left[\frac{|I| \cdot |L|}{N}\right] \leq \frac{N^{1-3\varepsilon} \cdot N^{1-2\varepsilon}}{N} = N^{1-5\varepsilon}$$

for sufficiently large λ with efficiency, which implies, by Markov's inequality,

$$\Pr[\, |L \cap I| > N^{1-4\varepsilon} \text{ in } \mathsf{Exp}^1_{\mathrm{PHFE}} \,] \leq \frac{\mathbb{E}\big[\, |L \cap I| \,\big]}{N^{1-4\varepsilon}} \leq \frac{N^{1-5\varepsilon}}{N^{1-4\varepsilon}} = N^{-\varepsilon} \leq \frac{1}{4}. \qquad \square$$

3.2 Barrier to Time Optimality

In this section, we show that if we build PHFE schemes that have very efficient decryption, i.e. ones that are f-fast, or x-fast or y-fast naturally imply doubly efficient secret-key PIR protocols (SKDEPIR) which are currently known only from a very specific assumption [BIPW17,CHR17].

We start by defining syntax of a SKDEPIR, along with the correctness, efficiency and security notions we consider. These definitions are suitably adapted from [BIPW17, CHR17].

Definition 21. (Syntax of SKDEPIR) An SKDEPIR scheme consists of five probabilistic polynomial-time algorithms (Keygen, Process, Query, Resp, Dec) with the following specification:

- Keygen takes as input the security parameter 1^λ and samples a key k.
- Process takes as input a key k, and a database $\mathsf{DB} \in \{0,1\}^n$ and outputs a processed database $\widetilde{\mathsf{DB}}$.
- Query takes as input the key k, an index $i \in [n]$, and outputs a query σ (and possibly a state st).
- Resp is a deterministic algorithm that takes as input σ and $\widetilde{\mathsf{DB}}$ and outputs a server response ρ.
- Dec is a deterministic algorithm that takes as input $(\sigma, \rho, k, \mathsf{st})$ and outputs a data element in $\{0,1\}$.

Next, we define the correctness and the efficiency property of SKDEPIR.

Definition 22. (Correctness/Efficiency of SKDEPIR) Let $c > 0$ be a constant, $\lambda \in \mathbb{N}$ be the security parameter and $n = \lambda^c$. Let $\mathsf{DB} \in \{0,1\}^n$, $i \in [n]$ and consider the following process:

- Run $k \leftarrow \mathsf{Keygen}$, $\widetilde{\mathsf{DB}} \leftarrow \mathsf{Process}(k, \mathsf{DB})$ and $(\sigma, \mathsf{st}) \leftarrow \mathsf{Query}(k, i)$,

- Compute $\rho = \mathsf{Resp}(\sigma, \widetilde{\mathsf{DB}})$ and $m \leftarrow \mathsf{Dec}(\sigma, \rho, k, \mathsf{st})$.

Then, we say that SKDEPIR is correct if $m = \mathsf{DB}_i$ with probability at least $1 - \mathsf{negl}(\lambda)$ for some negligible negl where the probability is over the coins of the process. Further, we say that it is doubly-efficient if the running time of Resp with RAM access to $\widetilde{\mathsf{DB}}$ is polynomial in λ and logarithm of n.

We now define the security notion we consider. For strengthened notions of security see [BIPW17,CHR17].

Definition 23. (Security of SKDEPIR) We say that SKDEPIR is secure if there exists a stateful p.p.t. simulator $\mathcal{S}$ such that for any stateful p.p.t. adversary $\mathcal{A}$ the following distributions are computationally indistinguishable. The first distribution is:

- $\mathcal{A}(1^\lambda)$ outputs 1^n, a database $\mathsf{DB} \in \{0,1\}^n$ and $i_1, \ldots, i_\ell$ where every $i_j \in [\ell]$.
- Run $k \leftarrow \mathsf{Keygen}$, $\widetilde{\mathsf{DB}} \leftarrow \mathsf{Process}(k, \mathsf{DB})$ and $(\sigma_j, \mathsf{st}_j) \leftarrow \mathsf{Query}(k, i_j)$ for $j \in [\ell]$,
- Output $(\mathsf{DB}, i_1, \ldots, i_\ell, \widetilde{\mathsf{DB}}, \sigma_1, \ldots, \sigma_\ell)$

The second distribution is:

- $\mathcal{A}(1^\lambda)$ outputs 1^n, a database $\mathsf{DB} \in \{0,1\}^n$ and $i_1, \ldots, i_\ell$ where every $i_j \in [\ell]$.
- The simulator outputs $(\widetilde{\mathsf{DB}}, \sigma_1, \ldots, \sigma_\ell) \leftarrow \mathcal{S}(\mathsf{DB}, \ell)$.
- Output $(\mathsf{DB}, i_1, \ldots, i_\ell, \widetilde{\mathsf{DB}}, \sigma_1, \ldots, \sigma_\ell)$

One could consider a stronger security definition where an adversary can ask for index queries adaptively. As of currently, this weaker definition is not known to be any easier to achieve. We will show that any PHFE scheme satisfying a weak form indistinguishability security, where the functions and the challenge messages are declared in the beginning, that is additionally f-fast/x-fast/y-fast implies such an SKDEPIR scheme. We now give overview of three (very similar) constructions. The first construction converts an x-fast PHFE scheme to an SKDEPIR scheme. The second, converts an f-fast PHFE scheme to an SKDEPIR scheme. Finally, the third scheme converts an y-fast PHFE scheme to an SKDEPIR scheme. Our main result is:

Theorem 9. *If there exists a* PHFE *scheme that is either x-fast or y-fast or f-fast as per definition 6, then, then there exist a secret-key doubly efficient PIR scheme as per Definitions 23 and 22.*

Overview We aim to show that an x-fast/f-fast/y-fast PHFE scheme imply a SKDEPIR scheme. We start by describing the main idea assuming x-fast PHFE scheme, and then we will sketch the ideas for the other cases. In a x-fast PHFE scheme the time it takes to compute decryption in the RAM model is proportional to $(T^{\beta_T}_{\varphi(f,x,y)} + |f|^{\beta_f} + |y|^{\beta_y})$ for some constants $\beta_T, \beta_f, \beta_y$ upto polynomial factors in λ. We will exploit the fact that there is no dependence on $|x|$ to turn this into a SKDEPIR scheme.

As a first attempt, we set DB as x, and y as empty. Then Process algorithm computes $\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$ as a preprocessing of the database. To look up DB_i, the Query algorithm computes a PHFE function key sk_{f_j} for the program that looks up and outputs DB_i and sends the key to the Server. Observe that Query algorithm runs in time polynomial in λ and $\log n$ as PHFE.KeyGen is a polynomial time in the description size of f and the security parameter.

Further, the Resp algorithm simply decrypts $\mathsf{ct}_{\mathsf{PHFE}}$ using sk_f. Observe that due to x- fastness, the time it takes to run Resp is also polynomial in λ and $\log n$. While this solves the core issue, we have completely missed one aspect. The scheme completely reveals the indices i to the Server as the function key sk_f is not guaranteed to hide the function description f. To resolve this issue, we observe that if we had a function hiding PHFE scheme, we would have been done. To enable this, we will use similar techniques as used to convert any functional encryption scheme to a function hiding functional encryption scheme [BS15]. Namely, we will compute a symmetric key encryption SKE of the index i (denoted as ctk_1). We will hardwire ctk_1 in the function secret key instead of the index i. The corresponding secret key $\mathsf{SKE.sk}_1$ will be put in the private component y, which will be used to decrypt ctk_1 to learn index i.

This might seem to be enough, but we face yet another issue. Learning DB_i in the clear upon decryption can reveal information about the index i to the Server. To fix this, the decryption will output the encryption of DB_i under another secret key $\mathsf{SKE.sk}_2$ of the secret-key encryption scheme. We will put this key in y component of the message along with a PRF key to derive randomness to compute the encryption. To make the proof go through, we need to use another encryption key $\mathsf{SKE.sk}_3$ and compute another ciphertext that will be hardwired in the functional secret key, along with a mode variable and a separate slot in y used in encryption. This is done in a standard fashion inspired by [BS15].

While these are the ideas to convert an x-fast PHFE scheme to a SKDEPIR, they immediately extend to converting a y-fast PHFE scheme to a SKDEPIR scheme. The idea is that while encrypting one could set $x = \perp$ and just use the hidden slot y alone. Now, the database DB will also be a part of the hidden input.

Conversion of an f-fast PHFE to SKDEPIR scheme is a slightly different. The idea is yet again a natural one. Instead of now encrypting DB inside the cipherthext, the client will process it as a part of a function key $\mathsf{sk}_{\mathsf{DB}}$. To look up an index i_j, the client will generate ciphertexts $\mathsf{ct}_{\mathsf{PHFE},j}$. These ciphertexts will encrypt a public component $x_j = \perp$ and a secret component $y_j = (\mathsf{SKE.sk}, r_j, i_j, \perp)$ where SKE.sk is a secret key of a secret key encryption scheme chosen once and for all by the Process algorithm and r_j is a randomness of length polynomial in λ, and $\perp$ represents an additional slot used in the proof. The function key is set so that upon decryption we get $\mathsf{SKE.Enc}(\mathsf{SKE.sk}, \mathsf{DB}_{i_j}; r_j)$. Now using SKE.sk client can learn DB_{i_j}. Observe that the function corresponding to the PHFE secret key on the input encrypted inside the ciphertexts takes time T which is polynomial in λ and $\log n$ in the RAM model. The length $|y_j|$ is also polynomial in λ and $\log n$. Therefore, the scheme satisfies the desired efficiency properties. Owing to similarities, we omit a formal treatment of this scheme. Below we give a formal description of how to convert an x-fast PHFE to a SKDEPIR scheme.

Ingredients for the constructions Our first ingredient is a PHFE scheme $\mathsf{PHFE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Keygen}, \mathsf{Dec})$ that is additionally x-fast. We also use a secret-key encryption scheme $\mathsf{SKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$.

Construction: x- fast PHFE to SKDEPIR

- Keygen : On input 1^λ, we run $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda)$. We also sample three secret keys for a secret-key encryption scheme $\{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}$. We additionally sample a PRF key k_{PRF} for a $\mathsf{PRF} = (\mathsf{Setup}, \mathsf{Eval})$ with input

Computation $\varphi(f, x, y)$

Hardwired: Ciphertexts ctk_1, ctk_3 and $t \in \{0,1\}^\lambda$
Input: $x = \mathsf{DB}$ and $y = (\mathsf{mode}, \alpha_1, \alpha_2, \alpha_3, \alpha_4)$
Output: ρ

- If $\mathsf{mode} = 0$, parse $\alpha_1 = \mathsf{SKE.sk}_1$, $\alpha = \mathsf{SKE.sk}_2$, $\alpha_3 = k_{\mathsf{PRF}}$ otherwise parse $\alpha_4 = \mathsf{SKE.sk}_3$.
- **Case** $\mathsf{mode} = 0$**:** Compute $i \leftarrow \mathsf{SKE.Dec}(\mathsf{SKE.sk}_1, \mathsf{ctk}_1)$ and output $\rho = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, \mathsf{DB}_i; \mathsf{PRF.Eval}(k_{\mathsf{PRF}}, t))$.
- **Case** $\mathsf{mode} = 1$**:** Output $\rho = \mathsf{SKE.Dec}(\mathsf{SKE.sk}_3, \mathsf{ctk}_3)$

Figure 2. The computation $\varphi(f, x, y)$.

and output lengths implicitly defined. The output of this algorithm is $k = (\mathsf{mpk}, \mathsf{msk}, \mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2, \mathsf{SKE.sk}_3, k_{\mathsf{PRF}})$.

- Process : On input k and $\mathsf{DB} \in \{0,1\}^n$, we set the public-input $x = \mathsf{DB}$ and $y = (\mathsf{mode} = 0, \alpha_1 = \mathsf{SKE.sk}_1, \alpha_2 = \mathsf{SKE.sk}_2, \alpha_3 = k_{\mathsf{PRF}}, \alpha_4 = \perp)$ (where the mode is a bit that can take two values in $\{0,1\}$ and $\perp$ will be used in the proof. The algorithm outputs $\widetilde{\mathsf{DB}} = (x = \mathsf{DB}, \mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y)))$.
- Query(k, i) : We sample a random string $t \leftarrow \{0,1\}^\lambda$ along with the following quantities:
  - Compute $\mathsf{ctk}_1 \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, i)$,
  - Compute $\mathsf{ctk}_2 \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, 0)$ and $\mathsf{ctk}_3 = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_2)$,
  - Compute $\mathsf{sk}_f \leftarrow \mathsf{PHFE.Keygen}(\mathsf{msk}, f)$ for a RAM program f where we describe the computation $\varphi(f, x, y)$ in Figure 2. Output of the algorithm is $\sigma = (f, \mathsf{sk}_f)$ and $\mathsf{st} = \perp$.
- Resp$(\sigma = (f, \mathsf{sk}_f), \widetilde{\mathsf{DB}} = \mathsf{DB}, \mathsf{ct}_{\mathsf{PHFE}})$: Output $\rho = \mathsf{PHFE.Dec}^{f, \mathsf{DB}, \mathsf{sk}_f, \mathsf{ct}_{\mathsf{PHFE}}}(1^\lambda)$.
- Dec$(\sigma = (f, \mathsf{sk}_f), \rho, k = (\mathsf{mpk}, \mathsf{msk}, \mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2, \mathsf{SKE.sk}_3, k_{\mathsf{PRF}}), \perp)$: Output $\mathsf{SKE.Dec}(\mathsf{SKE.sk}_2, \rho)$.

We now prove various properties associated below.

Correctness: The correctness follows from the correction of the SKE, and PHFE schemes. Note that Process encodes DB by computing a PHFE ciphertext $\mathsf{ct}_{\mathsf{PHFE}}$ encrypting DB in the public component and SKE secret keys $(\mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2)$ and a PRF key in the secret component. Query on input i, produces a PHFE function secret key that upon decrypting the ciphertext produces an SKE encryption of for any index i, under $\mathsf{SKE.sk}_2$. Resp then simply performs this decryption to come up with ρ, which is a SKE ciphertext encrypting DB_i using key $\mathsf{SKE.sk}_2$. Finally, Dec uses $\mathsf{SKE.sk}_2$ to learn DB_i.

Efficiency: We now bound the running time of Query and Resp.

- **Running time of** Query**:** Observe that Query computes a PHFE secret key sk_f for a function f that has a size polynomial in λ and $\log n$. This step takes time polynomial in λ and $\log n$ as KeyGen algorithms of PHFE takes polynomial time. Further, to construct f, the Query algorithm constructs ctk_1 and ctk_3, which also take time polynomial in λ and $\log n$.

- **Running time of** Resp**:** Resp simply performs $\mathsf{PHFE.Dec}^{f,\mathsf{DB},\mathsf{sk}_f,\mathsf{ct}_{\mathsf{PHFE}}}(1^\lambda)$. Since PHFE is x-fast the running time of decryption is $(T^{\beta_T}_{\varphi(f,x,y)} + |f|^{\beta_f} + |y|^{\alpha_y})\mathrm{poly}(\lambda, |\varphi|)$ for some constants $\beta_T, \beta_f, \beta_y$, since PHFE is x-fast. Observe that $T_{\varphi(f,x,y)}$ is bounded by polynomial in λ and $\log n$ as it consists of a constant number of SKE decryptions/encryption, a PRF computation, and a single lookup of DB at an index i. Furthermore, y is of size polynomial in λ as it consists of a PRF key along with SKE secret keys. Finally, f is of size polynomial in λ and $\log n$ as argued above. As a result, total time complexity of Resp in the RAM model is polynomial in λ and $\log n$.

We now prove security for the constructed SKDEPIR scheme.

Theorem 10. *Assuming* PRF *is a secure pseudorandom function,* SKE *is secure secret-key encryption scheme and* PHFE *is a secure partially hiding functional encryption scheme satisfying security Definition 7, the* SKDEPIR *scheme described above satisfies the security Definition 23.*

We prove this by providing a set of indistinguishable hybrids. The first hybrid corresponds to the first distribution of the security game, whereas the last hybrid corresponds to the second distribution. The simulator is implicit from the distribution in the last hybrid. We then formally describe our simulator. We use properties of various primitives involve to prove that the intermediate hybrids are indistinguishable.

- H_1 is the first distribution of the SKDEPIR game, where the adversary receives $\left(\mathsf{DB}, \{i_j\}_{j\in[\ell]}, \widetilde{\mathsf{DB}} = (\mathsf{DB}, \mathsf{ct}_{\mathsf{PHFE}}), \{\sigma_j = \left(f_j, \mathsf{sk}_{f_j}\right)\}_{j\in[\ell]}\right)$ where $\mathsf{ct}_{\mathsf{PHFE}}$, sk_{f_j} are computed as follows:

  $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda), \quad \{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}, \quad k_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
  $x = \mathsf{DB}, \quad y = (\mathsf{mode} = 0, \mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2, k_{\mathsf{PRF}}, \perp)$
  $\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$
  $\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, i_j), \quad \mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, 0),$
  $\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})$
  $t_j \leftarrow \{0,1\}^\lambda \quad \mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_{1,j}, \mathsf{ctk}_{3,j}, t_j])$

- H_2 In this hybrid, the only change is that $\mathsf{ctk}_{3,j}$ is now computed as an encryption of $\mathsf{ctk}_{2,j}$ that is computed differently. $\mathsf{ctk}_{2,j}$ is now computed as an SKE encryption

of DB_i using randomness derived from k_{PRF}:

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda), \quad \{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}, \quad k_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
$x = \mathsf{DB}, \quad y = (\mathsf{mode} = 0, \mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2, k_{\mathsf{PRF}}, \perp)$
$\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$
$\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, i_j), \quad \boxed{\mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, \mathsf{DB}_{i_j}; \mathsf{PRF.Eval}(k_{\mathsf{PRF}}, t_j))},$
$\boxed{\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})}$
$t_j \leftarrow \{0,1\}^\lambda \quad \mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_{1,j}, \mathsf{ctk}_{3,j}, t_j])$

Observe that H_1 and H_2 are indistinguishable due to the security of SKE. The key $\mathsf{SKE.sk}_3$ is hidden from the adversary.

- H_3 In this hybrid, the only change is that we now compute $\mathsf{ct}_{\mathsf{PHFE}}$ differently. The private component y is now designed so that it sets mode = 1. The secret keys $\mathsf{SKE.sk}_1, \mathsf{SKE.sk}_2$ and the PRF key k_{PRF} is removed and they are replaced with $\mathsf{SKE.sk}_3$:

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda), \quad \{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}, \quad k_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
$x = \mathsf{DB}, \quad \boxed{y = (\mathsf{mode} = 1, \perp, \perp, \perp, \mathsf{SKE.sk}_3)}$
$\boxed{\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))}$
$\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, i_j), \quad \mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, \mathsf{DB}_{i_j}; \mathsf{PRF.Eval}(k_{\mathsf{PRF}}, t_j)),$
$\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})$
$t_j \leftarrow \{0,1\}^\lambda \quad \mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_{1,j}, \mathsf{ctk}_{3,j}, t_j])$

Observe that H_2 and H_3 are indistinguishable due to the security of PHFE. Observe that in both the cases $\varphi(f_j, x, y)$ produce the same outcome $\mathsf{ctk}_{2,j}$. Therefore due to the security of the PHFE scheme these hybrids are computationally indistinguishable.

- H_4 In this hybrid, the only change is that now we encrypt $\mathsf{ctk}_{1,j}$ as an encryption of 0:

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda), \quad \{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}, \quad k_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
$x = \mathsf{DB}, \quad y = (\mathsf{mode} = 0, \perp, \perp, \perp, \mathsf{SKE.sk}_3)$
$\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$
$\boxed{\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, 0)}, \quad \mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, \mathsf{DB}_{i_j}; \mathsf{PRF.Eval}(k_{\mathsf{PRF}}, t_j)),$
$\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})$
$t_j \leftarrow \{0,1\}^\lambda \quad \mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_{1,j}, \mathsf{ctk}_{3,j}, t_j])$

Observe that H_3 and H_4 are indistinguishable due to the security of SKE. The key $\mathsf{SKE.sk}_1$ is hidden from the adversary.

- H_5 In this hybrid, the only change is that to compute $\mathsf{ctk}_{2,j}$ instead of deriving

randomness using the PRF key k_{PRF}:

$$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda), \quad \{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}, \quad k_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$$
$$x = \mathsf{DB}, \quad y = (\mathsf{mode} = 0, \bot, \bot, \bot, \mathsf{SKE.sk}_3)$$
$$\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$$
$$\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, 0), \quad \boxed{\mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, \mathsf{DB}_{i_j})},$$
$$\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})$$
$$t_j \leftarrow \{0,1\}^\lambda \quad \mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_{1,j}, \mathsf{ctk}_{3,j}, t_j])$$

Observe that H_4 and H_5 are indistinguishable due to the security of the PRF. The key k_{PRF} is hidden from the adversary and appears only in the form of values $\mathsf{PRF.Eval}(k_{\mathsf{PRF}}, t_j)$.

- H_6 In this hybrid, the only change is that $\mathsf{ctk}_{2,j}$ are now computed as an encryption of 0:

$$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PHFE.Setup}(1^\lambda), \quad \{\mathsf{SKE.sk}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)\}_{i\in[3]}, \quad k_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$$
$$x = \mathsf{DB}, \quad y = (\mathsf{mode} = 0, \bot, \bot, \bot, \mathsf{SKE.sk}_3)$$
$$\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$$
$$\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, 0), \quad \boxed{\mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, 0)},$$
$$\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})$$
$$t_j \leftarrow \{0,1\}^\lambda \quad \mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_{1,j}, \mathsf{ctk}_{3,j}, t_j])$$

Observe that H_5 and H_6 are indistinguishable due to the security of SKE. The key $\mathsf{SKE.sk}_2$ is hidden from the adversary.

From the description of the hybrid above, we now define the simulator $\mathcal{S}$ below:
Simulator $\mathcal{S}$:

- **Simulating $\widetilde{\mathsf{DB}}$:** We compute $\mathsf{ct}_{\mathsf{PHFE}} = \mathsf{PHFE.Enc}(\mathsf{mpk}, (x, y))$. where $x = \mathsf{DB}$ and $y = (\mathsf{mode} = 1, \bot, \bot, \bot, \mathsf{SKE.sk}_3)$.
- **Simulating sk_{f_j}:** We sample $t_j \leftarrow \{0,1\}^\lambda$, $\mathsf{ctk}_{1,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_1, 0)$ and $\mathsf{ctk}_{2,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_2, 0)$ and $\mathsf{ctk}_{3,j} = \mathsf{SKE.Enc}(\mathsf{SKE.sk}_3, \mathsf{ctk}_{2,j})$. Then, we set $\mathsf{sk}_{f_j} = \mathsf{PHFE.KeyGen}(\mathsf{msk}, f_j = f[\mathsf{ctk}_1, \mathsf{ctk}_3, t_j])$.

4 Bounded LGRAM with Fixed-Memory Security

In this section, we present our LGRAM with fixed-memory security. The construction is based on the works of [GS18a,GOS18,AL18,KNTY19].

Roughly speaking, [GS18a] constructs a circuit garbling scheme from *adaptively* secure LOT with *local* proof of security and lifts it to an adaptively secure scheme using somewhere equivocal encryption, which is further developed to obtain a garbled RAM scheme with fixed-memory security (or unprotected memory access) [GOS18]. The work of [AL18] modularizes the construction and shows how to obtain a succinct garbling scheme using obfuscation for circuits with polynomial-sized domains, which is in turn

$\mathsf{GenAugCPU}[\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, k](t)$

Hardwired.	$\mathsf{hk}_\tau, \mathsf{hk}_{\mathsf{work}}$,	LOT hash keys for the tapes;
	$h_\tau, \lvert D_\tau\rvert$,	LOT hashes and lengths of the input tapes;
	k,	PPRF key.
Input.	$t \in [T_{\max}]$,	current time (step number).
Output.	$\widehat{\mathsf{AugCPU}}_t$,	the garbled "augmented step circuit" at time t, computed as follows.

1. Obtain randomness from PPRF:
   $(\pi_t, \{L_{t,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}, r_t^{\mathsf{GC}}) \leftarrow \mathsf{PPRF.Eval}(k, t)$
   $(\pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}) \leftarrow \mathsf{PPRF.Eval}(k, t+1)$ with $r_{t+1}^{\mathsf{LOT}}, r_{t+1}^{\mathsf{GC}}$ removed
2. Garble AugCPU (Figure 4):
   $$\mathsf{AugCPU}_t \leftarrow \mathsf{AugCPU}\left[\begin{matrix}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \\ \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}\end{matrix}\right]$$
   output $\widehat{\mathsf{AugCPU}}_t \leftarrow \mathsf{GC.Garble}(\mathsf{AugCPU}_t, \pi_t, \{L_{t,i,b}\}_{i,b}; r_t^{\mathsf{GC}})$

Figure 3. The circuit GenAugCPU in Construction 1.

implied by FE for circuits. The work of [KNTY19] shows that *selectively* secure LOT suffices and that such an LOT is implied by FE for circuits.

All of the works consider garbling schemes with no public input, and the security notions for their final products are simulation-based. Consequently, the length of the garbling necessarily [AIKW13] grows linearly with the input and output lengths. In contrast, our notion of garbling has a short private input and we are interested in indistinguishability and succinctness.

4.1 Construction

Ingredients of Construction 1. Let

- $i\mathcal{O}$ be a circuit obfuscator,
- LOT an updatable LOT with fast initialization,

$$\mathsf{LOT} = (\mathsf{LOT.HashGen}, \mathsf{LOT.Hash}, \mathsf{LOT.SendRead}, \mathsf{LOT.RecvRead}, \mathsf{LOT.SendWrite}, \mathsf{LOT.RecvWrite}, \mathsf{LOT.Hash0s}),$$

- $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$ a circuit garbling scheme, and
- $\mathsf{PPRF} = (\mathsf{PPRF.Puncture}, \mathsf{PPRF.Eval})$ a puncturable PRF.

Construction 1 (bounded LGRAM with fixed-memory security)**.** Our bounded LGRAM works as follows:

$$\mathsf{AugCPU}_t: \quad \mathsf{AugCPU}\left[\begin{matrix}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}},\\ \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}\end{matrix}\right](w, \mathsf{st}_{t-1}, \mathsf{rdata}_{t-1}, h_{\mathsf{work},t-1})$$

Hardwired. $\mathsf{hk}_\tau, \mathsf{hk}_{\mathsf{work}}, h_\tau, |D_\tau|$, see Figure 3;
$\pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}$, to be selected for evaluating $\widehat{\mathsf{AugCPU}}_{t+1}$;
r_t^{LOT}, LOT randomness to select π, L for $\widehat{\mathsf{AugCPU}}_{t+1}$.

Input. $w, \mathsf{st}_{t-1}, \mathsf{rdata}_{t-1}$, short input, old state, last-read cell content;
$h_{\mathsf{work},t-1}$, LOT hash of the old working tape.

Output. $\mathsf{done}_t, \tau_t, i_t, \mathsf{wdata}_t, \mathsf{out}_t$, produced by $M^{D_1,\ldots,D_{\mathcal{T}}}(w)$ at time t;
selected (or LOT ciphertexts to select) π, L for $\widehat{\mathsf{AugCPU}}_{t+1}$;
computed as follows.

1. Run one step of M:

   $(\mathsf{done}_t, \mathsf{st}_t, \tau_t, i_t, \mathsf{wdata}_t, \mathsf{out}_t) \leftarrow \mathsf{CPU}(|D_1|, \ldots, |D_{\mathcal{T}}|, w, \mathsf{st}_{t-1}, \mathsf{rdata}_{t-1})$

   output $\mathsf{done}_t, \tau_t, i_t, \mathsf{wdata}_t, \mathsf{out}_t$

2. Split r_t^{LOT}, and $\pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}$ by the input components of AugCPU:

$$\left(\begin{matrix} r_{t,\mathsf{rct}}^{\mathsf{LOT}}, r_{t,\mathsf{wct}}^{\mathsf{LOT}}, & \pi_{t+1}^{w}, \pi_{t+1}^{\mathsf{st}}, \pi_{t+1}^{\mathsf{rdata}}, \pi_{t+1}^{h},\\ \{L_{t+1,i,b}^{w}\}_{i\in[\ell_{\mathsf{in}}],b}, & \{L_{t+1,i,b}^{\mathsf{st}}\}_{i\in[\ell_{\mathsf{st}}],b},\\ \{L_{t+1,i,b}^{\mathsf{rdata}}\}_{i\in[\ell_{\mathsf{CELL}}],b}, & \{L_{t+1,i,b}^{h}\}_{i\in[\ell_{\mathsf{HASH}}],b}\end{matrix}\right) \leftarrow (r_t^{\mathsf{LOT}}, \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b})$$

3. Select π, L for w, st_t if $\neg\mathsf{done}_t$:

   output $w \oplus \pi_{t+1}^{w}$, $\{L_{t+1,i,w[i]}^{w}\}_{i\in[\ell_{\mathsf{in}}]}$, $\mathsf{st}_t \oplus \pi_{t+1}^{\mathsf{st}}$, $\{L_{t+1,i,\mathsf{st}_t[i]}^{\mathsf{st}}\}_{i\in[\ell_{\mathsf{st}}]}$

4a. If $\tau_t \in [\mathcal{T}]$, select π, L for $D_{\tau_t}[i_t]\|0^{\ell_{\mathsf{CELL}}-\ell_{\mathsf{cell}}}$ and $h_{\mathsf{work},t} = h_{\mathsf{work},t-1}$:

$$\textbf{output } \mathsf{rct}_t \leftarrow \mathsf{LOT.SendRead}\left(\begin{matrix}\mathsf{hk}_{\tau_t}, h_{\tau_t}, i_t,\\ \{(b \oplus \pi_{t+1}^{\mathsf{rdata}}[i])\|L_{t+1,i,b}^{\mathsf{rdata}}\}_{i\in[\ell_{\mathsf{cell}}],b}\end{matrix}; r_{t,\mathsf{rct}}^{\mathsf{LOT}}\right)$$

and $\{(0 \oplus \pi_{t+1}^{\mathsf{rdata}}[i])\|L_{t+1,i,0}^{\mathsf{rdata}}\}_{i\in[\ell_{\mathsf{CELL}}]\setminus[\ell_{\mathsf{cell}}]}$

and $h_{\mathsf{work},t} \oplus \pi_{t+1}^{h}$, $\{L_{t+1,i,h_{\mathsf{work},t}[i]}^{h}\}_{i\in[\ell_{\mathsf{HASH}}]}$

4b. If $\tau_t = \mathsf{work}$, select π, L for $D_{\mathsf{work},t-1}[i_t]$ and updated $h_{\mathsf{work},t}$:

$$\textbf{output } \mathsf{rct}_t \leftarrow \mathsf{LOT.SendRead}\left(\begin{matrix}\mathsf{hk}_{\mathsf{work}}, h_{\mathsf{work},t-1}, i_t,\\ \{(b \oplus \pi_{t+1}^{\mathsf{rdata}}[i])\|L_{t+1,i,b}^{\mathsf{rdata}}\}_{i\in[\ell_{\mathsf{CELL}}],b}\end{matrix}; r_{t,\mathsf{rct}}^{\mathsf{LOT}}\right)$$

and $\mathsf{wct}_t \leftarrow \mathsf{LOT.SendWrite}\left(\begin{matrix}\mathsf{hk}_{\mathsf{work}}, h_{\mathsf{work},t-1}, i_t, \mathsf{wdata}_t,\\ \{(b \oplus \pi_{t+1}^{h}[i])\|L_{t+1,i,b}^{h}\}_{i\in[\ell_{\mathsf{HASH}}],b}\end{matrix}; r_{t,\mathsf{wct}}^{\mathsf{LOT}}\right)$

Figure 4. The circuit AugCPU in Construction 1.

- $\mathsf{Compress}(1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}, \tau, D_\tau)$ takes as input the cell length, the address length, the tape index, and the tape content. It runs

$$\mathsf{hk}_\tau \xleftarrow{\$} \mathsf{LOT.HashGen}(1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}), \qquad (h_\tau, \widehat{D}_\tau) \leftarrow \mathsf{LOT.Hash}(\mathsf{hk}_\tau, D_\tau),$$

and outputs $\mathsf{digest}_\tau = (\mathsf{hk}_\tau, h_\tau, |D_\tau|)$.

- $\mathsf{Garble}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$ takes as input an upper bound of the running time, the machine, and the input tape digests. It runs

$$\mathsf{hk}_{\mathsf{work}} \xleftarrow{\$} \mathsf{LOT.HashGen}(1^{\ell_{\mathrm{CELL}}}, 1^{\ell_{\mathrm{ADDR}}}), \qquad h_{\mathsf{work},0} \leftarrow \mathsf{LOT.Hash0s}(\mathsf{hk}_{\mathsf{work}}, T_{\max}),$$

samples PPRF key k, prepares $\mathsf{GenAugCPU}$ using M (Figure 3), and runs

$$\widetilde{\mathsf{GenAugCPU}} \xleftarrow{\$} i\mathcal{O}(\mathsf{GenAugCPU}[k, \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}]).$$

$\mathsf{GenAugCPU}$ is padded to size $\mathrm{poly}(\lambda, |M|, \log T_{\max})$ for the security proof to work. The algorithm computes using k and splits π, L for $\widetilde{\mathsf{AugCPU}}_1$ as defined in Figure 4,

$$\begin{aligned} \pi_1 :&\quad \pi_1^w, \pi_1^{\mathsf{st}}, \pi_1^{\mathsf{rdata}}, \pi_1^h, \\ \{L_{1,i,b}\}_{i,b} :&\quad \{L^w_{1,i,b}\}_{i\in[\ell_{\mathrm{in}}],b}, \{L^{\mathsf{st}}_{1,i,b}\}_{i\in[\ell_{\mathrm{st}}],b}, \{L^{\mathsf{rdata}}_{1,i,b}\}_{i\in[\ell_{\mathrm{CELL}}],b}, \{L^h_{1,i,b}\}_{i\in[\ell_{\mathrm{HASH}}],b}, \end{aligned}$$

sets $\mathsf{st}_0 = 0^{\ell_{\mathrm{st}}}$, $\mathsf{rdata}_0 = 0^{\ell_{\mathrm{CELL}}}$, and outputs

$$\widehat{M} = \left(\begin{array}{l} \mathsf{hk}_{\mathsf{work}}, \widetilde{\mathsf{GenAugCPU}}, \mathsf{st}_0 \oplus \pi_1^{\mathsf{st}}, \mathsf{rdata}_0 \oplus \pi_1^{\mathsf{rdata}}, h_{\mathsf{work},0} \oplus \pi_1^h, \\ \{L^{\mathsf{st}}_{1,i,\mathsf{st}_0[i]}\}_{i\in[\ell_{\mathrm{st}}]}, \{L^{\mathsf{rdata}}_{1,i,\mathsf{rdata}_0[i]}\}_{i\in[\ell_{\mathrm{CELL}}]}, \{L^h_{1,i,h_{\mathsf{work},0}[i]}\}_{i\in[\ell_{\mathrm{HASH}}]}, \end{array} \right),$$
$$\{L_{i,b} = (b \oplus \pi_1^w[i]) \| L^w_{1,i,b}\}_{i\in[\ell_{\mathrm{in}}],b\in\{0,1\}}.$$

- $\mathsf{Eval}^{D_1,\ldots,D_{\mathcal{T}}}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}, \{L_i\}_{i\in[\ell_{\mathrm{in}}]})$ takes as input the upper bound of the running time, the machine, the input tape digests, the garbled machine, and a set of labels. It parses $\widehat{M}$ as specified in Garble, and does the following:

  1. Initialize by

$$\begin{aligned} (h_\tau, \widehat{D}_\tau) &\leftarrow \mathsf{LOT.Hash}(\mathsf{hk}_\tau, D_\tau) \qquad \text{for } \tau \in [\mathcal{T}], \\ (h_{\mathsf{work},0}, \widehat{D}_{\mathsf{work},0}) &\leftarrow \mathsf{LOT.Hash}(\mathsf{hk}_{\mathsf{work}}, 0^{T_{\max}}), \\ \mathsf{st}_0 &\leftarrow 0^{\ell_{\mathrm{st}}}, \qquad \mathsf{rdata}_0 \leftarrow 0^{\ell_{\mathrm{CELL}}}, \qquad t \leftarrow 1. \end{aligned}$$

  2. Writing

$$\begin{aligned} X_t &= w \| \mathsf{st}_{t-1} \| \mathsf{rdata}_{t-1} \| h_{\mathsf{work},t-1}, \\ Y_t &= \text{parts of } (X_{t+1} \oplus \pi_{t+1}) \text{ and } L_{t+1} \text{ and } \mathsf{rct}_t, \mathsf{wct}_t, \end{aligned}$$

  run[14]

$$\begin{aligned} \widetilde{\mathsf{AugCPU}}_t &\leftarrow \widetilde{\mathsf{GenAugCPU}}(t), \\ (\mathsf{done}_t, \tau_t, i_t, \mathsf{wdata}_t, \mathsf{out}_t, Y_t) &\leftarrow \mathsf{GC.Eval}(\widetilde{\mathsf{AugCPU}}_t, X_t \oplus \pi_t, \{L_{t,i,X_t[i]}\}_i), \end{aligned}$$

  and halt if done_t is set. Otherwise, output out_t and continue.

[14] X_t itself is not needed for evaluation and is not supposed to be efficiently computable. The values of $(X_t \oplus \pi_t)$ and $\{L_{t,i,X_t[i]}\}_i$ for $t = 1$ are read from $\widehat{M}$, and those for $t > 1$ are computed by evaluating $\widetilde{\mathsf{AugCPU}}_{t-1}$ as explained later.

3. If $\tau_t \in [\mathcal{T}]$, pick rct_t from Y_t and run

$$\{(\mathsf{rdata}_t[i] \oplus \pi^{\mathsf{rdata}}_{t+1,i}) \| L^{\mathsf{rdata}}_{t+1,i,\mathsf{rdata}_t[i]}\}_{i\in[\ell_{\mathsf{cell}}]} \leftarrow \mathsf{LOT.RecvRead}^{\widehat{D}_{\tau_t}}(\mathsf{hk}_{\tau_t}, h_{\tau_t}, i_t, \mathsf{rct}_t).$$

Combine it with parts of $(X_{t+1} \oplus \pi_{t+1})$ and L_{t+1} already in Y_t to obtain $(X_{t+1} \oplus \pi_{t+1})$ and $\{L_{t+1,i,X_{t+1}[i]}\}_i$.

4. Otherwise, $\tau_t = \mathsf{work}$, then pick $\mathsf{rct}_t, \mathsf{wct}_t$ from Y_t, run

$$\{(\mathsf{rdata}_t[i] \oplus \pi^{\mathsf{rdata}}_{t+1,i}) \| L^{\mathsf{rdata}}_{t+1,i,\mathsf{rdata}_t[i]}\}_{i\in[\ell_{\mathsf{CELL}}]} \leftarrow \mathsf{LOT.RecvRead}^{\widehat{D}_{\mathsf{work},t-1}}(\mathsf{hk}_{\mathsf{work}}, h_{\mathsf{work},t-1}, i_t, \mathsf{rct}_t),$$

$$\{(h_{\mathsf{work},t} \oplus \pi^{h}_{t+1,i}) \| L^{h}_{t+1,i,h_{\mathsf{work},t}[i]}\}_{i\in[\ell_{\mathsf{HASH}}]} \leftarrow \mathsf{LOT.RecvWrite}^{\widehat{D}_{\mathsf{work},t-1}}\begin{pmatrix}\mathsf{hk}_{\mathsf{work}}, h_{\mathsf{work},t-1},\\ i_t, \mathsf{wdata}_t,\end{pmatrix} \mathsf{wct}_t\Bigg),$$

updating $\widehat{D}_{\mathsf{work},t-1}$ into $\widehat{D}_{\mathsf{work},t}$. Again, combine it with parts of $(X_{t+1} \oplus \pi_{t+1})$ and L_{t+1} already in Y_t to obtain $(X_{t+1} \oplus \pi_{t+1})$ and $\{L_{t+1,i,X_{t+1}[i]}\}_i$.

5. Let $t \leftarrow t+1$ and go back to step 2.

Correctness and Efficiency. They follow from those of all the ingredients and the invariant that AugCPU_t is evaluated on $w, \mathsf{st}_{t-1}, \mathsf{rdata}_{t-1}, h_{\mathsf{work},t-1}$ coinciding with those from the execution of M. We remark that the evaluation time is *not* instance-specific, because processing the empty working tape for LOT already takes time $T_{\max}\,\mathrm{poly}(\lambda, |M|)$.

4.2 Security

Theorem 11 (¶). *Suppose in Construction 1, $i\mathcal{O}$ is secure for polynomial-sized domains (Definition 11),* LOT *is read- and write-secure (Definitions 14 and 15), and* GC, PPRF *are secure (Definition 16 and 17), then the constructed scheme is fixed-memory secure (Definition 4).*

The proof follows the pebbling strategy in [GS18a]. Recall that in $\mathsf{Exp}^b_{\mathrm{LGRAM}}$, the LGRAM labels are selected for w_b, and each AugCPU_t performs a step of $M^{D_1,\ldots,D_{\mathcal{T}}}(w_b)$, for which we write $M^{\cdots}(w_b)$ hereafter in this proof. Along the hybrids from $\mathsf{Exp}^0_{\mathrm{LGRAM}}$ to $\mathsf{Exp}^1_{\mathrm{LGRAM}}$, we gradually switch the trailing steps from those for $M^{\cdots}(w_0)$ to those for $M^{\cdots}(w_1)$.

Specification of Hybrids. Each hybrid $\mathsf{H}_{\mathfrak{t},\mathfrak{s}}$ is indexed by a natural number $0 \le \mathfrak{t} \le T_{\max}+1$ and a set $\mathfrak{s} \subseteq [T_{\max}]$. The indices $\mathfrak{t}, \mathfrak{s}$ specify how AugCPU_t is garbled and what it does:

- when $t < \mathfrak{t}$, it runs the t^{th} step of $M^{\cdots}(w_0)$;
  - if $t \notin \mathfrak{s}$, the step is garbled normally;
  - if $t \in \mathfrak{s}$, the step is simulated with true randomness for labels and LOT encryption that selects the labels for AugCPU_{t+1};
- when $t = \mathfrak{t}$, it is simulated and outputs the labels so that $\mathsf{AugCPU}_{\mathfrak{t}+1}$ runs the $(\mathfrak{t}+1)^{\text{st}}$ step of $M^{\cdots}(w_1)$, i.e., it switches the execution from $M^{\cdots}(w_0)$ to $M^{\cdots}(w_1)$;
- when $t > \mathfrak{t}$, it is garbled normally but runs the t^{th} step of $M^{\cdots}(w_1)$, due to the behavior of $\mathsf{AugCPU}_{\mathfrak{t}}$.

There are two special cases:

- when $\mathsf{t} = 0$, we give the LGRAM labels for w_1 (think of them as the output of the zeroth step circuit) to the adversary so that AugCPU_1 runs the first step of $M^{\cdots}(w_1)$ when the LGRAM is evaluated normally;
- when $t > T$ for $t \in \{\mathsf{t}\} \cup \mathsf{s}$, neither execution has a t^{th} step, and we simulate this step as if it were the last step of the execution, whose output would not select any label for the $(t+1)^{\text{st}}$ step.[15]

In the parlance of [GS18a], consider a line graph where vertex t represents step t,

- a gray pebble is placed on $t \in \mathsf{s}$ if $t < \mathsf{t}$ (the step is being simulated), and
- a black pebble is placed on $t \geq \mathsf{t}$ (the step is done for good).

However, the first black pebble (on t) is special for us as step t facilitates the transition from $M^{\cdots}(w_0)$ to $M^{\cdots}(w_1)$. We will use an optimized pebbling sequence [GS18a] to connect the hybrids.

Formally, let

$$\begin{aligned}
\mathsf{stS}(M, D_1, \ldots, D_{\mathcal{T}}, w_0) &= (\mathsf{st}_{0,1}, \ldots, \mathsf{st}_{0,T-1}),\\
\mathsf{stS}(M, D_1, \ldots, D_{\mathcal{T}}, w_1) &= (\mathsf{st}_{1,1}, \ldots, \mathsf{st}_{1,T-1}),\\
\mathsf{addrS}(\ldots, w_0) = \mathsf{addrS}(\ldots, w_1) &= (\tau_1, i_1, \ldots, \tau_{T-1}, i_{T-1}),\\
\mathsf{writeS}(\ldots, w_0) = \mathsf{writeS}(\ldots, w_1) &= (\mathsf{wdata}_1, \ldots, \mathsf{wdata}_{T-1}),\\
\mathsf{outS}(\ldots, w_0) = \mathsf{outS}(\ldots, w_1) &= (\mathsf{out}_1, \ldots, \mathsf{out}_{T-1}).
\end{aligned}$$

We also write $D_{\mathsf{work},t}$ for the working tape content, rdata_t for the read cell content, and $h_\tau, h_{\mathsf{work},t}$ for the LOT hashes — due to the constraint of fixed-memory security, the values of them in the two executions coincide. Define

$$X_{b,t} = w_b \| \mathsf{st}_{b,t-1} \| \mathsf{rdata}_{t-1} \| h_{\mathsf{work},t-1}.$$

In $\mathsf{H}_{\mathsf{t},\mathsf{s}}$, we change $\widetilde{\mathsf{GenAugCPU}}$ in $\widehat{M}$ to

$$i\mathcal{O}\left(\mathsf{GenAugCPU}'\left[\begin{array}{l}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}},\\ \mathsf{t}, \mathsf{s}, \mathring{k}_{\{\mathsf{t}\}\cup\mathsf{s}}, \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathsf{s}}\end{array}\right]\right),$$

where

- $\mathsf{GenAugCPU}'$ is shown in Figure 5,
- $\mathring{k}_{\{\mathsf{t}\}\cup\mathsf{s}}$ is a PPRF key punctured at $\{\mathsf{t}\} \cup \mathsf{s}$,
- π_t's and $L_{t,i,b}$'s are random strings, and
- $\widetilde{\mathsf{AugCPU}}_t$'s are the simulated garbled step circuits.

$$\mathsf{GenAugCPU}'\left[\begin{matrix}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}},\\ \mathsf{t}, \mathfrak{s}, \mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}}, \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}\end{matrix}\right](t)$$

Hardwired. $\mathsf{hk}_\tau, \mathsf{hk}_{\mathsf{work}}, h_\tau, |D_\tau|$, see Figure 3;
$\mathsf{t}, \mathfrak{s}$, switching time, hardwired step numbers;
$\mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}}$, punctured PPRF key;
$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}$, hardwired randomness and garbled steps.

Input. $t \in [T_{\max}]$, current time (step number).

Output. $\widehat{\mathsf{AugCPU}}_t$, computed as follows.

1. Check for hardwired steps, if $t \in \{\mathsf{t}\} \cup \mathfrak{s}$:

   output hardwired $\widehat{\mathsf{AugCPU}}_t$ and **terminate**

2. Otherwise, obtain randomness, either hardwired or from PPRF:

   $$(\pi_t, \{L_{t,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}, r_t^{\mathsf{GC}}) \leftarrow \mathsf{PPRF.Eval}(\mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}}, t)$$

   $$(\pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}) \leftarrow \begin{cases}\text{hardwired}, & \text{if } t+1 \in \{\mathsf{t}\}\cup\mathfrak{s};\\ \mathsf{PPRF.Eval}(\mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}}, t+1) \text{ with } r_{t+1}^{\mathsf{LOT}}, r_{t+1}^{\mathsf{GC}} \text{ removed}, & \\ & \text{if } t+1 \notin \{\mathsf{t}\}\cup\mathfrak{s};\end{cases}$$

3. Garble AugCPU (Figure 4) as done in GenAugCPU (Figure 3):

   $$\mathsf{AugCPU}_t \leftarrow \mathsf{AugCPU}\left[\begin{matrix}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}},\\ \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}\end{matrix}\right]$$

   output $\widehat{\mathsf{AugCPU}}_t \leftarrow \mathsf{GC.Garble}(\mathsf{AugCPU}_t, \pi_t, \{L_{t,i,b}\}_{i,b}; r_t^{\mathsf{GC}})$

Figure 5. The circuit GenAugCPU′ in the proof of Theorem 11.

We still use $\pi_t, L_{t,i,b}$ for part of the PPRF output when $t \notin \{\mathsf{t}\} \cup \mathfrak{s}$.

$\widetilde{\mathsf{AugCPU}}_t$'s are generated by

$$\mathsf{GC}.\widetilde{\mathsf{Garble}}(1^{|\mathsf{AugCPU}|}, Z_{b,t}, X_{0,t} \oplus \pi_t, \{L_{t,i,X_{0,t}[i]}\}_i) \text{ with } b = \begin{cases} 0, & \text{if } t < \mathsf{t} \text{ and } t \in \mathfrak{s}; \\ 1, & \text{if } t = \mathsf{t}; \end{cases}$$

where $\mathsf{GC}.\widetilde{\mathsf{Garble}}$ is a simulator for GC, and $Z_{b,t}$ for $b \in \{0,1\}$ consists of $(\mathsf{done}_t, \tau_t, i_t, \mathsf{wdata}_t, \mathsf{out}_t)$, parts of $(X_{b,t+1} \oplus \pi_{t+1})$ and L_{t+1}, and $\widetilde{\mathsf{rct}}_t, \widetilde{\mathsf{wct}}_t$ (cf. Figure 4):

- If $\neg\mathsf{done}_t$, then $Z_{b,t}$ contains (dependent on b)

$$w_b \oplus \pi^w_{t+1}, \{L^w_{t+1,i,w_b[i]}\}_{i\in[\ell_{\mathrm{in}}]}, \quad \mathsf{st}_{b,t} \oplus \pi^w_{t+1}, \{L^{\mathsf{st}}_{t+1,i,\mathsf{st}_{b,t}[i]}\}_{i\in[\ell_{\mathrm{st}}]}.$$

- If $\tau_t \in [\mathcal{T}]$, then $Z_{b,t}$ contains (independent of b)

$$\begin{aligned} &\widetilde{\mathsf{rct}}_t \xleftarrow{\$} \mathsf{LOT.SimRead}\left(\begin{array}{l} \mathsf{hk}_{\tau_t}, D_{\tau_t}, i_t, \\ \{(\mathsf{rdata}_t[i] \oplus \pi^{\mathsf{rdata}}_{t+1}[i]) \| L^{\mathsf{rdata}}_{t+1,i,\mathsf{rdata}_t[i]}\}_{i\in[\ell_{\mathrm{cell}}]} \end{array}\right), \\ &\{(0 \oplus \pi^{\mathsf{rdata}}_{t+1}[i]) \| L^{\mathsf{rdata}}_{t+1,i,0}\}_{i\in[\ell_{\mathrm{CELL}}]\setminus[\ell_{\mathrm{cell}}]}, \\ &h_{\mathsf{work},t} \oplus \pi^h_{t+1}, \{L^h_{t+1,i,h_{\mathsf{work},t}[i]}\}_{i\in[\ell_{\mathrm{HASH}}]}. \end{aligned}$$

- If $\tau_t = \mathsf{work}$, then $Z_{b,t}$ contains (independent of b)

$$\begin{aligned} \widetilde{\mathsf{rct}}_t &\xleftarrow{\$} \mathsf{LOT.SimRead}\left(\begin{array}{l} \mathsf{hk}_{\mathsf{work}}, D_{\mathsf{work},t-1}, i_t, \\ \{(\mathsf{rdata}_t[i] \oplus \pi^{\mathsf{rdata}}_{t+1}[i]) \| L^{\mathsf{rdata}}_{t+1,i,\mathsf{rdata}_t[i]}\}_{i\in[\ell_{\mathrm{CELL}}]} \end{array}\right), \\ \widetilde{\mathsf{wct}}_t &\xleftarrow{\$} \mathsf{LOT.SimWrite}\left(\begin{array}{l} \mathsf{hk}_{\mathsf{work}}, D_{\mathsf{work},t-1}, i_t, \mathsf{wdata}_t, \\ \{(h_{\mathsf{work},t}[i] \oplus \pi^h_{t+1}[i]) \| L^h_{t+1,i,h_{\mathsf{work},t}[i]}\}_{i\in[\ell_{\mathrm{HASH}}]} \end{array}\right). \end{aligned}$$

Connecting the Hybrids. We start with the experiments in Definition 4:

Claim 12 (¶). $\mathsf{Exp}^0_{\mathrm{LGRAM}} \approx \mathsf{H}_{T_{\max}+1,\varnothing}$ *and* $\mathsf{Exp}^1_{\mathrm{LGRAM}} \approx \mathsf{H}_{0,\varnothing}$.

It remains to connect $\mathsf{H}_{T_{\max}+1,\varnothing}$ and $\mathsf{H}_{0,\varnothing}$, for which we employ the pebbling strategy of [GOS18]. We state our claims in parallel with the pebbling rules:

Claim 13 (Rule A; ¶). *A gray pebble can be placed on or removed from $t^\star$ if $t^\star = 1$ or a gray pebble is placed on $(t^\star - 1)$. Formally, $\mathsf{H}_{\mathsf{t},\mathfrak{s}} \approx \mathsf{H}_{\mathsf{t},\{t^\star\}\cup\mathfrak{s}}$ for $t^\star \notin \mathfrak{s}$ and $t^\star < \mathsf{t}$ if $t^\star = 1$ or $t^\star - 1 \in \mathfrak{s}$.*

Claim 14 (Rule B; ¶). *A gray pebble on $t^\star$ can be replaced by a black pebble if black pebbles are placed on all vertices greater than $t^\star$. Formally, $\mathsf{H}_{\mathsf{t},\mathfrak{s}} \approx \mathsf{H}_{\mathsf{t}-1,\mathfrak{s}\setminus\{\mathsf{t}-1\}}$ if $\mathsf{t}-1 \in \mathfrak{s}$ (here, $t^\star = \mathsf{t}-1$). Moreover, $\mathsf{H}_{1,\varnothing} \approx \mathsf{H}_{0,\varnothing}$.*

[15] This pretention trick unifies the usage of garbled circuit security. Since an out-of-range t^{th} step is pretended to repeat the last existent step, simulating it removes all the labels for the $(t+1)^{\mathrm{st}}$ step, and consequently (and inductively), we can pretend that the set of labels for the $(t+1)^{\mathrm{st}}$ step corresponding to the input to the last existent step were revealed, so that the $(t+1)^{\mathrm{st}}$ step can also be pretended to repeat the last step. Without pretending, we will need another security notion of garbled circuits, namely that it can be simulated without any output if no input labels are given. This notion holds for many known constructions, but is not implied by the usual version.

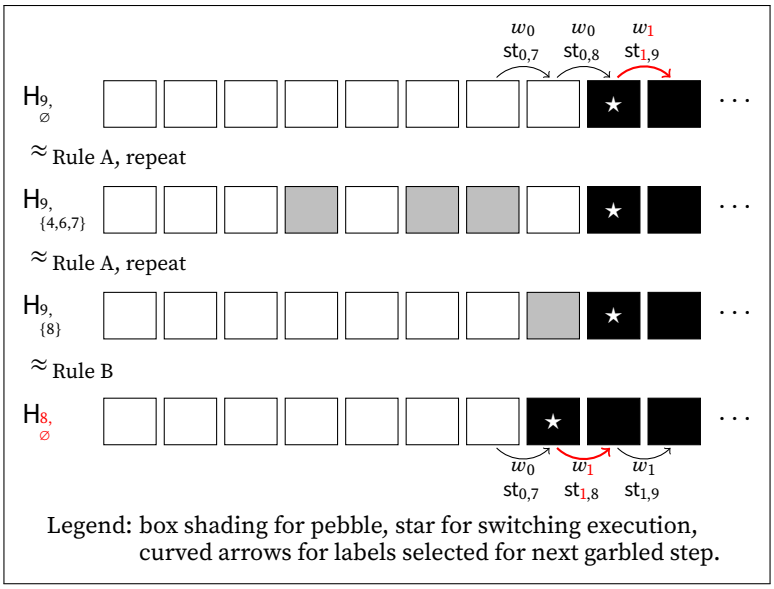

Figure 6. Typical hybrid transitions in the proof of Theorem 11.

Claim 14 is different from rule B in [GOS18] (no need for a gray pebble on $(t^\star - 1)$ for us) because the step corresponding to the first black pebble is always simulated.

It is helpful to consider a virtual vertex 0 (corresponding to $\widetilde{M}$ and the LGRAM labels), where a gray pebble is initially placed, and put 0 into $\mathfrak{s}$. With the virtual vertex, the separate case $t^\star = 1$ in rule A is just a special case of $t^\star - 1 = 0 \in \mathfrak{s}$, and the separate case $\mathsf{H}_{1,\varnothing} \approx \mathsf{H}_{0,\varnothing}$ in rule B becomes a special case of the general rule (which would read $\mathsf{H}_{1,\{0\}} \approx \mathsf{H}_{0,\varnothing}$ instead). However, for consistency with existing literature, we will go without the virtual vertex in the text below.

The hybrid sequence can be built using an optimized pebbling strategy:

Lemma 15 ([Ben89,GS18a])**.** *There exists an efficient algorithm that on input $1^{T_{\max}}$ computes a pebbling sequence for a line graph of size $T_{\max}$ with $\mathrm{O}(\log T_{\max})$ gray pebbles at any time during the $\mathrm{poly}(T_{\max})$ moves using rule A or B, starting with no pebbles and ending with black pebbles on all vertices. As a corollary, the efficient algorithm can compute*

$$\mathsf{t}_0 = T_{\max} + 1, \mathfrak{s}_0 = \varnothing, \qquad c_1, \mathsf{t}_1, \mathfrak{s}_1, \qquad c_2, \mathsf{t}_2, \mathfrak{s}_2, \qquad \ldots,$$
$$c_{\mathrm{poly}(T_{\max})-1}, \mathsf{t}_{\mathrm{poly}(T_{\max})-1} = 1, \mathfrak{s}_{\mathrm{poly}(T_{\max})-1} = \varnothing, \qquad c_{\mathrm{poly}(T_{\max})}, \mathsf{t}_{\mathrm{poly}(T_{\max})} = 0, \mathfrak{s}_{\mathrm{poly}(T_{\max})} = \varnothing,$$

such that $\max_j |\mathfrak{s}_j| = \mathrm{O}(\log T_{\max})$ and $\mathsf{H}_{\mathsf{t}_{j-1},\mathfrak{s}_{j-1}} \approx \mathsf{H}_{\mathsf{t}_j,\mathfrak{s}_j}$ by Claim $c_j \in \{13, 14\}$.

A typical sequence of hybrid transitions is depicted in Figure 6.

Proof (Theorem 11). By Claims 12, 13, and 14, and Lemma 15, $\mathsf{Exp}^0_{\mathrm{LGRAM}} \approx \mathsf{Exp}^1_{\mathrm{LGRAM}}$. □

We remark that for full rigor, $T_{\max}$ is a random variable, not a fixed number for each λ. It cannot be derandomized even in the non-uniform setting, because $\mathcal{A}$, given hk's returned for the input tapes, can choose $T_{\max}$ adaptively, whose randomness originates from that of the LGRAM scheme. This phenomenon is not uncommon and can be solved by either guessing $T_{\max}$ (or targeting a specific $T_{\max}$ in the non-uniform setting), or considering a polynomial upper bound of $T_{\max}$ and supplying appropriate definitions for hybrids with out-of-range indices.

Proof (Claim 12). Both follow from the security of $i\mathcal{O}$ for polynomial-sized domains. □

Proof (Claim 13). Let $\mathsf{t}, \mathfrak{s}, t^\star$ be such that $t^\star \notin \mathfrak{s}$ and $t^\star < \mathsf{t}$ and either $t^\star = 1$ or $t^\star - 1 \in \mathfrak{s}$. The hybrids below are mostly identical to $\mathsf{H}_{\mathsf{t},\mathfrak{s}}$ except the contents hardwired into GenAugCPU′ are modified. We specify them:

- G_0. This is just $\mathsf{H}_{\mathsf{t},\mathfrak{s}}$, where the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}.$$

- G_1. In this hybrid, the PPRF key k is punctured at $\{t^\star, \mathsf{t}\} \cup \mathfrak{s}$ instead of $\{\mathsf{t}\} \cup \mathfrak{s}$, and the $(t^\star)^{\text{th}}$ garbled step (but not its randomness) is hardwired into GenAugCPU′, i.e., the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \boxed{\{t^\star\} \cup \mathfrak{s}}, \quad \boxed{\mathring{k}_{\{\mathsf{t},t^\star\}\cup\mathfrak{s}}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}, \quad \boxed{\widehat{\mathsf{AugCPU}}_{t^\star},}$$

  where the newly hardwired information is computed using (Figure 5 Step 3)

$$(\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}, r^{\mathsf{LOT}}_{t^\star}, r^{\mathsf{GC}}_{t^\star}) \leftarrow \mathsf{PPRF.Eval}(k, t^\star),$$
$$\mathsf{AugCPU}_{t^\star} \leftarrow \mathsf{AugCPU}\left[\begin{matrix}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}},\\ \pi_{t^\star+1}, \{L_{t^\star+1,i,b}\}_{i,b}, r^{\mathsf{LOT}}_{t^\star}\end{matrix}\right],$$
$$\widehat{\mathsf{AugCPU}}_{t^\star} \leftarrow \mathsf{GC.Garble}(\mathsf{AugCPU}_{t^\star}, \pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}; r^{\mathsf{GC}}_{t^\star}).$$

  This is different from $\widehat{\mathsf{AugCPU}}_t$'s for $t \in \{\mathsf{t}\} \cup \mathfrak{s}$, which are simulated. The absence of $\pi_{t^\star}$ and $\{L_{t^\star,i,b}\}_{i,b}$ in GenAugCPU′ is fine, because they are only used in Step 2 (Figure 5) for $t = t^\star - 1$ and $t = t^\star$. In G_1, that branch is not taken for those two values of t as both of them are handled by Step 1. Therefore, the two versions of GenAugCPU′ in G_0 and G_1 have identical truth tables (and are padded appropriately to the same size), and $\mathsf{G}_0 \approx \mathsf{G}_1$ follows from the security of $i\mathcal{O}$ for polynomial-sized domains.

- G_2. In this hybrid, $\mathsf{PPRF.Eval}(k, t^\star)$ is replaced by a uniformly random string, i.e., the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \{t^\star\} \cup \mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t},t^\star\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}, \quad \widehat{\mathsf{AugCPU}}_{t^\star},$$

  where

$$(\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}, r^{\mathsf{LOT}}_{t^\star}, r^{\mathsf{GC}}_{t^\star}) \leftarrow \boxed{\text{uniformly random}},$$
$$\mathsf{AugCPU}_{t^\star} \leftarrow \mathsf{AugCPU}\left[\begin{matrix}\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}},\\ \pi_{t^\star+1}, \{L_{t^\star+1,i,b}\}_{i,b}, r^{\mathsf{LOT}}_{t^\star}\end{matrix}\right],$$
$$\widehat{\mathsf{AugCPU}}_{t^\star} \leftarrow \mathsf{GC.Garble}(\mathsf{AugCPU}_{t^\star}, \pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}; r^{\mathsf{GC}}_{t^\star}).$$

  $\mathsf{G}_1 \approx \mathsf{G}_2$ follows from the security of PPRF.

- G_3. In this hybrid, $\widetilde{\mathsf{AugCPU}}_{t^\star}$ is simulated instead of being normally generated, i.e., the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \{t^\star\}\cup\mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t},t^\star\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}, \quad \widetilde{\mathsf{AugCPU}}_{t^\star},$$

where

$$(\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}, r^{\mathsf{LOT}}_{t^\star}, \boxed{r^{\mathsf{GC}}_{t^\star}}) \leftarrow \text{uniformly random},$$
$$\widetilde{\mathsf{AugCPU}}_{t^\star} \xleftarrow{\$} \boxed{\mathsf{GC}.\widetilde{\mathsf{Garble}}(1^{|\mathsf{AugCPU}|}, Z_{0,t^\star}, X_{0,t^\star}\oplus\pi_{t^\star}, \{L_{t^\star,i,X_{0,t^\star}[i]}\}_i)},$$
$$Z_{0,t^\star} : \pi_{t^\star+1}, \{L_{t^\star+1,i,b}\}_{i,b} \begin{cases} \text{in the clear,} \\ \text{in } \mathsf{rct}_{t^\star} \leftarrow \mathsf{LOT}.\mathsf{SendRead}(\cdots; r^{\mathsf{LOT}}_{t^\star,\mathsf{rct}}), \\ \text{in } \mathsf{wct}_{t^\star} \leftarrow \mathsf{LOT}.\mathsf{SendWrite}(\cdots; r^{\mathsf{LOT}}_{t^\star,\mathsf{wct}}). \end{cases}$$

This is different from $Z_{b,t}$'s for $t \in \{\mathsf{t}\}\cup\mathfrak{s}$, which might contain simulated $\widetilde{\mathsf{rct}}_t$'s and $\widetilde{\mathsf{wct}}_t$'s. In G_2, the augmented step circuit $\mathsf{AugCPU}_{t^\star}$ is garbled with truly random $\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}, r^{\mathsf{GC}}_{t^\star}$. If $t^\star = 1$, then $\pi_1, \{L_{1,i,b}\}_{i,b}$ only additional appear in $\widetilde{M}$ and the LGRAM labels given to the adversary as $(X_{0,1}\oplus\pi_1)$ and $\{L_{1,i,X_{0,1}[i]}\}_i$. If $t^\star - 1 \in \mathfrak{s}$, then they are only additionally used to simulate

$$\widetilde{\mathsf{AugCPU}}_{t^\star-1} \xleftarrow{\$} \mathsf{GC}.\widetilde{\mathsf{Garble}}(\ldots, Z_{0,t^\star-1}, \ldots),$$

where $Z_{0,t^\star-1}$ contains only $(X_{0,t^\star}\oplus\pi_{t^\star})$ and $\{L_{t^\star,i,X_{0,t^\star}[i]}\}_i$, potentially in the clear, in $\widetilde{\mathsf{rct}}_{t^\star-1}$, in $\widetilde{\mathsf{wct}}_{t^\star-1}$, and by imagination.[16] Also, $\mathsf{AugCPU}_t(X_{0,t^\star}) = Z_{0,t^\star}$. It follows from the security of GC that $\mathsf{G}_2 \approx \mathsf{G}_3$.

- G_4. In this hybrid, $Z_{0,t^\star}$ contains $\widetilde{\mathsf{rct}}_{t^\star}, \widetilde{\mathsf{wct}}_{t^\star}$ simulated using $\mathsf{LOT}.\mathsf{SimRead}$ and $\mathsf{LOT}.\mathsf{SimWrite}$ as needed, instead of normally generated ones, i.e., the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \{t^\star\}\cup\mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t},t^\star\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}, \quad \widetilde{\mathsf{AugCPU}}_{t^\star},$$

where

$$(\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}, \boxed{r^{\mathsf{LOT}}_{t^\star}}) \leftarrow \text{uniformly random},$$
$$\widetilde{\mathsf{AugCPU}}_{t^\star} \xleftarrow{\$} \mathsf{GC}.\widetilde{\mathsf{Garble}}(1^{|\mathsf{AugCPU}|}, Z_{0,t^\star}, X_{0,t^\star}\oplus\pi_{t^\star}, \{L_{t^\star,i,X_{0,t^\star}[i]}\}_i),$$
$$Z_{0,t^\star} : \boxed{X_{0,t^\star+1}\oplus\pi_{t^\star+1}, \{L_{t^\star+1,i,X_{0,t^\star+1}[i]}\}_i} \begin{cases} \text{in the clear,} \\ \text{in } \widetilde{\mathsf{rct}}_{t^\star} \xleftarrow{\$} \boxed{\mathsf{LOT}.\mathsf{SimRead}(\cdots)}, \\ \text{in } \widetilde{\mathsf{wct}}_{t^\star} \xleftarrow{\$} \boxed{\mathsf{LOT}.\mathsf{SimWrite}(\cdots)}. \end{cases}$$

The LOT ciphertexts $\mathsf{rct}_{t^\star}, \mathsf{wct}_{t^\star}$ are encrypted using truly random $r^{\mathsf{LOT}}_{t^\star}$ in G_3. Each database is known before its LOT hash key is provided to the adversary: for an

[16] When $t^\star > T$, none of $\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}$ appears in $Z_{0,t^\star-1}$. We pretend $X_{0,t^\star} = X_{0,T}$ and that $(X_{0,t^\star}\oplus\pi_{t^\star})$ and $\{t^\star, i, X_{0,t^\star}[i]\}_i$ are revealed, imagining that the $(t^\star)^{\text{th}}$ step repeated the last existent step.

input tape, hk_τ is provided after D_τ is chosen; for the working tape, $D_{\mathsf{work},t^\star-1}$ and $i_{t^\star}, \mathsf{wdata}_{t^\star}$ are known once the adversary commits to the challenge M, w_0, w_1, after which $\mathsf{hk}_{\mathsf{work}}$ is provided. Therefore, it follows from the read/write security of LOT that $\mathsf{G}_3 \approx \mathsf{G}_4$.

- G_5. In this hybrid, we hardwire $\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b}$ into GenAugCPU′, i.e., the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \{t^\star\}\cup\mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t},t^\star\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}, \quad \boxed{\pi_{t^\star}, \{L_{t^\star,i,b}\}_{i,b},} \widetilde{\mathsf{AugCPU}}_{t^\star}.$$

  $\mathsf{G}_4 \approx \mathsf{G}_5$ follows from the security of $i\mathcal{O}$ for polynomial-sized domains.

By inspection, G_5 is just $\mathsf{H}_{\mathsf{t},\{t^\star\}\cup\mathfrak{s}}$. Therefore, $\mathsf{H}_{\mathsf{t},\mathfrak{s}} \equiv \mathsf{G}_0 \approx \mathsf{G}_5 \equiv \mathsf{H}_{\mathsf{t},\{t^\star\}\cup\mathfrak{s}}$. □

Proof (Claim 14). Let $\mathsf{t}, \mathfrak{s}$ be such that $\mathsf{t} = 1$ or $\mathsf{t}-1 \in \mathfrak{s}$. The hybrids are mostly identical to $\mathsf{H}_{\mathsf{t},\mathfrak{s}}$ except for the contents hardwired into GenAugCPU′:

- G_0. This is just $\mathsf{H}_{\mathsf{t},\mathfrak{s}}$, where the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{t\in\{\mathsf{t}\}\cup\mathfrak{s}}.$$

- G_1. In this hybrid, we no longer directly hardwire $\pi_t, \{L_{t,i,b}\}_{i,b}$ into GenAugCPU′ for $t = \mathsf{t}$, i.e., the hardwired information is

$$\{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \quad \mathsf{t}, \quad \mathfrak{s}, \quad \mathring{k}_{\{\mathsf{t}\}\cup\mathfrak{s}},$$
$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{\boxed{t\in\mathfrak{s}}}, \quad \boxed{\pi_{\mathsf{t}}, \{L_{\mathsf{t},i,b}\}_{i,b},} \widetilde{\mathsf{AugCPU}}_{\mathsf{t}},$$

  where

$$(\pi_{\mathsf{t}}, \{L_{\mathsf{t},i,b}\}_{i,b}) \xleftarrow{\$} \text{uniformly random};$$
$$\begin{cases} \widehat{M}, \text{LGRAM labels} \leftarrow X_{0,1}\oplus\pi_1, \{L_{1,i,X_{0,1}[i]}\}_i, \ldots, & \text{if } \mathsf{t}=1; \\ \widetilde{\mathsf{AugCPU}}_{\mathsf{t}-1} \xleftarrow{\$} \widetilde{\mathsf{GC.Garble}}\left(\begin{matrix} 1^{|\mathsf{AugCPU}|}, Z_{0,\mathsf{t}-1}, \\ X_{0,\mathsf{t}-1}\oplus\pi_{\mathsf{t}-1}, \{L_{\mathsf{t}-1,i,X_{0,\mathsf{t}-1}[i]}\}_i \end{matrix}\right), & \\ \quad Z_{0,\mathsf{t}-1} : X_{0,\mathsf{t}}\oplus\pi_{\mathsf{t}}, \{L_{\mathsf{t},i,X_{0,\mathsf{t}}[i]}\}_i \text{ in the clear}, \widetilde{\mathsf{rct}}_{\mathsf{t}-1}, \widetilde{\mathsf{wct}}_{\mathsf{t}-1}, & \text{if } \mathsf{t}-1\in\mathfrak{s}; \\ \widetilde{\mathsf{AugCPU}}_{\mathsf{t}} \xleftarrow{\$} \widetilde{\mathsf{GC.Garble}}\left(\begin{matrix} 1^{|\mathsf{AugCPU}|}, Z_{1,\mathsf{t}}, \\ X_{0,\mathsf{t}}\oplus\pi_{\mathsf{t}}, \{L_{\mathsf{t},i,X_{0,\mathsf{t}}[i]}\}_i \end{matrix}\right), & \\ \quad Z_{1,\mathsf{t}} : X_{1,\mathsf{t}+1}\oplus\pi_{\mathsf{t}+1}, \{L_{\mathsf{t}+1,i,X_{1,\mathsf{t}+1}[i]}\}_i \text{ in the clear}, \widetilde{\mathsf{rct}}_{\mathsf{t}}, \widetilde{\mathsf{wct}}_{\mathsf{t}}, & \text{if } \mathsf{t}\le T_{\max}. \end{cases}$$

  Note that GenAugCPU′ still indirectly contains $\pi_{\mathsf{t}}, \{L_{\mathsf{t},i,b}\}_{i,b}$ via $\widehat{M}$ and LGRAM labels, or $Z_{0,\mathsf{t}-1}$. Removal of their direct hardwiring does not alter the truth table of GenAugCPU′ as they are only used in Step 2 (Figure 5) for $t = \mathsf{t}$ and $t = \mathsf{t}-1$, a branch never taken for those values of t because they are handled by Step 1. By the security of $i\mathcal{O}$ for polynomial-sized domains, $\mathsf{G}_0 \approx \mathsf{G}_1$.

- G_2. In this hybrid, we change the input of $\mathsf{AugCPU}_\mathsf{t}$ from $X_{0,\mathsf{t}}$ to $X_{1,\mathsf{t}}$ by modifying the permuted input and the labels selected for $\widehat{\mathsf{AugCPU}}_\mathsf{t}$, i.e., the hardwired information is

$$\begin{array}{llll} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, & \mathsf{t}, & \mathsf{s}, & \mathring{k}_{\{\mathsf{t}\}\cup\mathsf{s}}, \\ \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\mathsf{s}}, & \widehat{\mathsf{AugCPU}}_\mathsf{t}, & & \end{array}$$

where

$$(\pi_\mathsf{t}, \{L_{\mathsf{t},i,b}\}_{i,b}) \xleftarrow{\$} \text{uniformly random};$$

$$\begin{cases} \widehat{M}, \text{LGRAM labels} \leftarrow \boxed{X_{1,1}\oplus\pi_1, \{L_{1,i,X_{1,1}[i]}\}_i}, \ldots, & \text{if } \mathsf{t}=1; \\ \widehat{\mathsf{AugCPU}}_{\mathsf{t}-1} \xleftarrow{\$} \widetilde{\mathsf{GC.Garble}}\left(\begin{array}{l} 1^{|\mathsf{AugCPU}|}, Z_{0,\mathsf{t}-1}, \\ X_{0,\mathsf{t}-1}\oplus\pi_{\mathsf{t}-1}, \{L_{\mathsf{t}-1,i,X_{0,\mathsf{t}-1}[i]}\}_i \end{array}\right), & \\ Z_{0,\mathsf{t}-1} : \boxed{X_{1,\mathsf{t}}\oplus\pi_1, \{L_{\mathsf{t},i,X_{1,\mathsf{t}}[i]}\}_i} \text{ in the clear}, \widetilde{\mathsf{rct}}_{\mathsf{t}-1}, \widetilde{\mathsf{wct}}_{\mathsf{t}-1}, & \text{if } \mathsf{t}-1\in\mathsf{s}; \\ \widehat{\mathsf{AugCPU}}_\mathsf{t} \xleftarrow{\$} \widehat{\mathsf{GC.Garble}}\left(\begin{array}{l} 1^{|\mathsf{AugCPU}|}, Z_{1,\mathsf{t}}, \\ \boxed{X_{1,1}\oplus\pi_1, \{L_{1,i,X_{1,1}[i]}\}_i} \end{array}\right), & \\ Z_{1,\mathsf{t}} : X_{1,\mathsf{t}+1}\oplus\pi_{\mathsf{t}+1}, \{L_{\mathsf{t}+1,i,X_{1,\mathsf{t}+1}[i]}\}_i \text{ in the clear}, \widetilde{\mathsf{rct}}_\mathsf{t}, \widetilde{\mathsf{wct}}_\mathsf{t}, & \text{if } \mathsf{t}\le T_{\max}. \end{cases}$$

This change does not alter the distribution, i.e., $\mathsf{G}_1 \equiv \mathsf{G}_2$, because π_t is a one-time pad that hides $X_{0,\mathsf{t}}$ or $X_{1,\mathsf{t}}$ and either set of $\{L_{\mathsf{t},i,b}\}_{i,b}$ chosen by $X_{0,\mathsf{t}}$ and $X_{1,\mathsf{t}}$ is simply random.

- G_3. In this hybrid, we undo the simulation of $\widehat{\mathsf{rct}}_\mathsf{t}$ and $\widehat{\mathsf{wct}}_\mathsf{t}$, i.e., the hardwired information is

$$\begin{array}{llll} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, & \mathsf{t}, & \mathsf{s}, & \mathring{k}_{\{\mathsf{t}\}\cup\mathsf{s}}, \\ \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\mathsf{s}}, & \widehat{\mathsf{AugCPU}}_\mathsf{t}, & & \end{array}$$

where

$$(\pi_\mathsf{t}, \{L_{\mathsf{t},i,b}\}_{i,b} \boxed{, r^{\mathsf{LOT}}_\mathsf{t}}) \xleftarrow{\$} \text{uniformly random};$$

$$\begin{cases} \widehat{M}, \text{LGRAM labels} \leftarrow X_{1,1}\oplus\pi_1, \{L_{1,i,X_{1,1}[i]}\}_i, \ldots, & \text{if } \mathsf{t}=1; \\ \widehat{\mathsf{AugCPU}}_{\mathsf{t}-1} \xleftarrow{\$} \widetilde{\mathsf{GC.Garble}}\left(\begin{array}{l} 1^{|\mathsf{AugCPU}|}, Z_{0,\mathsf{t}-1}, \\ X_{0,\mathsf{t}-1}\oplus\pi_{\mathsf{t}-1}, \{L_{\mathsf{t}-1,i,X_{0,\mathsf{t}-1}[i]}\}_i \end{array}\right), & \\ Z_{0,\mathsf{t}-1} : X_{1,\mathsf{t}}\oplus\pi_1, \{L_{\mathsf{t},i,X_{1,\mathsf{t}}[i]}\}_i \text{ in the clear}, \widetilde{\mathsf{rct}}_{\mathsf{t}-1}, \widetilde{\mathsf{wct}}_{\mathsf{t}-1}, & \text{if } \mathsf{t}-1\in\mathsf{s}; \\ \widehat{\mathsf{AugCPU}}_\mathsf{t} \xleftarrow{\$} \widehat{\mathsf{GC.Garble}}\left(\begin{array}{l} 1^{|\mathsf{AugCPU}|}, Z_{1,\mathsf{t}}, \\ X_{1,1}\oplus\pi_1, \{L_{1,i,X_{1,1}[i]}\}_i \end{array}\right), & \\ Z_{1,\mathsf{t}} : \boxed{\pi_{\mathsf{t}+1}, \{L_{\mathsf{t}+1,i,b}\}_{i,b}} \text{ in the clear}, \boxed{\mathsf{rct}_\mathsf{t}, \mathsf{wct}_\mathsf{t}}, & \text{if } \mathsf{t}\le T_{\max}. \end{cases}$$

$\mathsf{G}_2 \approx \mathsf{G}_3$ by the read/write security of LOT.

- G_4. In this hybrid, we undo the simulation of $\widehat{\mathsf{AugCPU}}_\mathsf{t}$, i.e., the hardwired information is

$$\begin{array}{llll} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, & \mathsf{t}, & \mathsf{s}, & \mathring{k}_{\{\mathsf{t}\}\cup\mathsf{s}}, \\ \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\mathsf{AugCPU}}_t\}_{t\in\mathsf{s}}, & \widehat{\mathsf{AugCPU}}_\mathsf{t}, & & \end{array}$$

where

$$(\pi_t, \{L_{t,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}, r_t^{\mathsf{GC}}) \xleftarrow{\$} \text{uniformly random};$$

$$\begin{cases} \widehat{M}, \text{LGRAM labels} \leftarrow X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i, \ldots, & \text{if } t = 1; \\ \widetilde{\mathsf{AugCPU}}_{t-1} \xleftarrow{\$} \widetilde{\mathsf{GC.Garble}}\left(\begin{matrix} 1^{|\mathsf{AugCPU}|}, Z_{0,t-1}, \\ X_{0,t-1} \oplus \pi_{t-1}, \{L_{t-1,i,X_{0,t-1}[i]}\}_i \end{matrix}\right), & \\ Z_{0,t-1} : X_{1,t} \oplus \pi_1, \{L_{t,i,X_{1,t}[i]}\}_i \text{ in the clear}, \widetilde{\mathsf{rct}}_{t-1}, \widetilde{\mathsf{wct}}_{t-1}, & \text{if } t-1 \in \mathfrak{s}; \\ \mathsf{AugCPU}_t \leftarrow \mathsf{AugCPU}\left[\begin{matrix} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \\ \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}, r_t^{\mathsf{LOT}} \end{matrix}\right], & \\ \widetilde{\mathsf{AugCPU}}_t \leftarrow \boxed{\mathsf{GC.Garble}(\mathsf{AugCPU}_t, \pi_t, \{L_{t,i,b}\}_{i,b}; r_t^{\mathsf{GC}})}, & \text{if } t \le T_{\max}. \end{cases}$$

Since $\mathsf{AugCPU}_t(X_{1,t}) = Z_{1,t}$, it follows from the security of GC that $\mathsf{G}_3 \approx \mathsf{G}_4$.

- G_5. In this hybrid, we change the randomness for step t to the PPRF evaluation, i.e., the hardwired information is

$$\begin{matrix} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, & t, & \mathfrak{s}, & \mathring{k}_{\{t\}\cup\mathfrak{s}}, \\ \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{t\in\mathfrak{s}}, & \widetilde{\mathsf{AugCPU}}_t, & & \end{matrix}$$

where

$$(\pi_t, \{L_{t,i,b}\}_{i,b}, r_t^{\mathsf{LOT}}, r_t^{\mathsf{GC}}) \xleftarrow{\$} \boxed{\mathsf{PPRF.Eval}(k, t)};$$

$$\begin{cases} \widehat{M}, \text{LGRAM labels} \leftarrow X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i, \ldots, & \text{if } t = 1; \\ \widetilde{\mathsf{AugCPU}}_{t-1} \xleftarrow{\$} \widetilde{\mathsf{GC.Garble}}\left(\begin{matrix} 1^{|\mathsf{AugCPU}|}, Z_{0,t-1}, \\ X_{0,t-1} \oplus \pi_{t-1}, \{L_{t-1,i,X_{0,t-1}[i]}\}_i \end{matrix}\right), & \\ Z_{0,t-1} : X_{1,t} \oplus \pi_1, \{L_{t,i,X_{1,t}[i]}\}_i \text{ in the clear}, \widetilde{\mathsf{rct}}_{t-1}, \widetilde{\mathsf{wct}}_{t-1}, & \text{if } t-1 \in \mathfrak{s}; \\ \mathsf{AugCPU}_t \leftarrow \mathsf{AugCPU}\left[\begin{matrix} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, \\ \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}, r_t^{\mathsf{LOT}} \end{matrix}\right], & \\ \widetilde{\mathsf{AugCPU}}_t \leftarrow \mathsf{GC.Garble}(\mathsf{AugCPU}_t, \pi_t, \{L_{t,i,b}\}_{i,b}; r_t^{\mathsf{GC}}), & \text{if } t \le T_{\max}. \end{cases}$$

$\mathsf{G}_4 \approx \mathsf{G}_5$ by the security of PPRF.

- G_6. In this hybrid, we no longer puncture the PPRF key at t and no longer hardwire the randomness nor the garbled step for t, i.e., the hardwired information is

$$\begin{matrix} \{\mathsf{hk}_\tau, h_\tau, |D_\tau|\}_{\tau\in[\mathcal{T}]}, \mathsf{hk}_{\mathsf{work}}, & t-1, & \mathfrak{s}\setminus\{t-1\}, & \boxed{\mathring{k}_{\{t-1\}\cup(\mathfrak{s}\setminus\{t-1\})}}, \\ \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widetilde{\mathsf{AugCPU}}_t\}_{\boxed{t\in\{t-1\}\cup(\mathfrak{s}\setminus\{t-1\})}}, & & & \end{matrix}$$

and $\widehat{M}$ and the LGRAM labels given to the adversary contain $X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i$ if $t = 1$. By the security of $i\mathcal{O}$ for polynomial-sized domains, $\mathsf{G}_5 \approx \mathsf{G}_6$.

By inspection, G_6 is just $\mathsf{H}_{t-1,\mathfrak{s}\setminus\{t-1\}}$. Therefore, $\mathsf{H}_{t,\mathfrak{s}} \equiv \mathsf{G}_0 \approx \mathsf{G}_6 \equiv \mathsf{H}_{t-1,\mathfrak{s}\setminus\{t-1\}}$. □

5 Transformations of LGRAM

In this section, we describe the transformations of LGRAM to upgrade its security and functionality. We adapt known transformations [CH16] to our syntax of bounded LGRAM to go from fixed-memory security, to fixed-address section (Section 5.1), and to full security (Section 5.2). We employ the powers-of-two transformation [AL18] to construct an unbounded LGRAM from a bounded one (Section 5.3).

5.1 Fixed-Memory to Fixed-Address

Given an LGRAM with fixed-memory security, we construct one with fixed-address security. Recall that in such a scheme, we want to hide the write sequence, consisting of the contents written to the working tape. The addresses where each read and write occurs are not protected, though. We employ a transformation due to [CH16] upgrading fixed-memory security to fixed-address security, by encrypting the data before they are written to the working tape. In more details, given a RAM M, we construct another machine M', which runs two copies of M in parallel and encrypts the two logical working tapes of M using two puncturable PRF keys. Normally, only one copy is used and the other mimics its memory access pattern. The idle copy is used in the security proof.

Ingredients of Construction 2. Let

- $\mathsf{LGRAM}' = (\mathsf{Compress}', \mathsf{Garble}', \mathsf{Eval}')$ be an LGRAM with fixed-memory security, and
- $\mathsf{PPRF} = (\mathsf{PPRF.Puncture}, \mathsf{PPRF.Eval})$ a puncturable PRF.

Construction 2 (LGRAM with fixed-address security)**.** Our scheme works as follows:

- $\mathsf{Compress}(1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, \tau, D_\tau)$ is the same as $\mathsf{Compress}'$.
- $\mathsf{Garble}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$ prepares M' from M as shown in Figure 7, samples $\beta \xleftarrow{\$} \{0,1\}$ and two random PPRF keys k_0, k_1, runs

$$(\widehat{M}', \{L'_{i,b}\}_{i,b}) \xleftarrow{\$} \mathsf{Garble}'(T_{\max}, M', \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}),$$

  and splits $\{L'_{i,b}\}_{i,b}$, according to the part of input to M' they select, into

$$\{L^{\mathrm{B}}_b\}_b, \{L^{W_0}_{i,b}\}_{i,b}, \{L^{W_1}_{i,b}\}_{i,b}, \{L^{K_0}_{i,b}\}_{i,b}, \{L^{K_1}_{i,b}\}_{i,b}, \quad \{L^{T'}_{i,b}\}_{i,b}, \{L^{\mathsf{wdata}'}_{i,b}\}_{i,b}.$$

  The algorithm sets and outputs

$$\widehat{M} = (\widehat{M}', \{\text{labels for } \mathrm{B} = \beta, K_0 = k_0, K_1 = k_1, T' = 0, \mathsf{wdata}' = \perp\}),$$
$$L_{i,b} = L^{W_0}_{i,b} \| L^{W_1}_{i,b} \qquad \text{for } i \in [\ell_{\text{in}}], b \in \{0,1\}.$$

- $\mathsf{Eval}^{D_1,\ldots,D_\mathcal{T}}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}, \{L_i\}_i)$ combines the labels for $\widehat{M}'$ in $\widehat{M}$ with $\{L_i\}_i$ to recover $\{L'_i\}_i$, and runs and outputs

$$(\mathsf{Eval}')^{D_1,\ldots,D_\mathcal{T}}(T_{\max}, M', \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}', \{L'_i\}_i).$$

Machine M' Transformed from M

Lengths.
$\ell'_{\text{in}} = \text{poly}(\lambda, |M|, \log T_{\text{max}})$, $\ell'_{\text{st}} = \text{poly}(\lambda, |M|, \log T_{\text{max}})$,
$\ell'_{\text{addr}} = \ell_{\text{addr}}$, $\ell'_{\text{cell}} = \ell_{\text{cell}}$,
$\ell'_{\text{ADDR}} = \ell_{\text{ADDR}}$, $\ell'_{\text{CELL}} = 2\ell_{\text{CELL}} + O(\log T_{\text{max}})$.

Input (w')**.**
- B, bit indicating the active copy;
- W_0, W_1, short inputs to the two copies of M;
- K_0, K_1, PPRF keys for the two copies of M;
- T', progress of security proof;
- wdata', hardwired content of one cell of M'.

State (st')**.**
- $t-1$, current time step minus one (initially 0);
- $\widetilde{\text{st}}_{0,t-1}, \widetilde{\text{st}}_{1,t-1}$, optional old states of the two copies of M;
- τ_{t-1}, i_{t-1}, location of the last-read cell.

Steps. Each step of M' executes one step in each copy of M.

1. Translate rdata' for M' into rdata for M:
   if $t = 1$ or $\tau_{t-1} \in [\mathcal{T}]$:
   parse rdata'_{t-1} into $\widetilde{\text{rdata}}_{0,t-1} \| 0^{\ell'_{\text{CELL}} - \ell_{\text{CELL}}} = \widetilde{\text{rdata}}_{1,t-1} \| 0^{\ell'_{\text{CELL}} - \ell_{\text{CELL}}}$
   if $t \neq 1$ and $\tau_{t-1} = \text{work}$:
   compute $\widetilde{\text{rdata}}_{0,t-1}, \widetilde{\text{rdata}}_{1,t-1}$

$$\begin{cases} \text{from } \text{rdata}'_{t-1} = t' \| \big(\widetilde{\text{rdata}}_{0,t-1} \oplus \text{PPRF.Eval}(k_0, t')\big) \| \big(\widetilde{\text{rdata}}_{1,t-1} \oplus \text{PPRF.Eval}(k_1, t')\big), & \\ & \text{if } t' \neq 0; \\ \text{as } 0^{\ell_{\text{CELL}}}, 0^{\ell_{\text{CELL}}}, & \text{otherwise;} \end{cases}$$

2. Execute one step in the active copy:
   $(\text{done}_t, \widetilde{\text{st}}_{\text{B},t}, \tau_t, i_t, \widetilde{\text{wdata}}_{\text{B},t}, \text{out}_t) \leftarrow \text{CPU}(\ell_1, \ldots, \ell_{\mathcal{T}}, w_{\text{B}}, \widetilde{\text{st}}_{\text{B},t-1}, \widetilde{\text{rdata}}_{\text{B},t-1})$
3. Optionally execute one step in the inactive copy:
$$(\widetilde{\text{st}}_{1-\text{B},t}, \widetilde{\text{wdata}}_{1-\text{B},t}) \leftarrow \begin{cases} (0^{\ell_{\text{st}}}, 0^{\ell_{\text{CELL}}}), & \text{if } t > T'; \\ \text{from CPU}\left(\ell_1, \ldots, \ell_{\mathcal{T}}, w_{1-\text{B}}, \widetilde{\text{st}}_{1-\text{B},t-1}, \widetilde{\text{rdata}}_{1-\text{B},t-1}\right), & \text{if } t \leq T'. \end{cases}$$
4. Translate wdata for M into wdata' for M':
$$\text{wdata}'_t \leftarrow \begin{cases} \perp, & \text{if } \tau_t \in [\mathcal{T}]; \\ \text{wdata}', & \text{if } t = \text{work and } t = T' \text{ and } \text{wdata}' \neq \perp; \\ t \| \big(\widetilde{\text{wdata}}_{0,t} \oplus \text{PPRF.Eval}(k_0, t)\big) \| \big(\widetilde{\text{wdata}}_{1,t} \oplus \text{PPRF.Eval}(k_1, t)\big), & \\ & \text{otherwise;} \end{cases}$$
5. Output and finish:
   output $(\text{done}_t, \text{newst}', \tau_t, i_t, \text{wdata}'_t, \text{out}_t)$
   where newst′ contains incremented t and $\widetilde{\text{st}}_{0,t}, \widetilde{\text{st}}_{1,t}, \tau_t, i_t$

Figure 7. The machine M' in Construction 2.

Correctness and Efficiency. By construction, during evaluation, the labels selected for M' correspond to

$$\mathrm{B} = \beta, \quad W_0 = w, W_1 = w, \quad K_0 = k_0, K_1 = k_1, \quad T' = 0, \mathsf{wdata}' = \bot.$$

The execution of M' has the following important invariants:

- $\widetilde{\mathsf{st}}_{\beta,t} = \mathsf{st}_t$, where st_t is from $M^{\cdots}(w)$.
- $D'_{\mathsf{work},t}[i]$ for all t, i is either all-zero if never touched (never read nor written to) or

  $$t' \| \big(D_{0,\mathsf{work},t}[i] \oplus \mathsf{PPRF.Eval}(k_0, t')\big) \| \big(D_{1,\mathsf{work},t}[i] \oplus \mathsf{PPRF.Eval}(k_1, t')\big),$$

  where $t' \neq 0$ is the last time when cell i on the working tape of M was touched, $D_{\beta,\mathsf{work},t}$ is $D_{\mathsf{work},t}$ of $M^{\cdots}(w)$, and $D_{1-\beta,\mathsf{work},t}$ is all-zero. In all cases, $\widetilde{\mathsf{rdata}}_{\beta,t} = \mathsf{rdata}_t$, where rdata_t is from $M^{\cdots}(w)$.

Correctness follows from that of LGRAM′ and those invariants, together with the other logics in M'. For efficiency, clearly $|M'| = \mathrm{poly}(\lambda, |M|, \log T_{\max})$, and M' has the same running time as M when supplied with the input specified above.

Theorem 16 (¶). *Suppose in Construction 2,* LGRAM′ *is fixed-memory secure (Definition 3 or 4) and* PPRF *is secure (Definition 17), then the constructed scheme is fixed-address secure (Definition 3 or 4) and inherits the (un-)boundedness of* LGRAM′.

Proof (Theorem 16). $\mathsf{Exp}^b_{\mathrm{LGRAM}}$ of the constructed scheme corresponds to $\mathsf{Exp}^b_{\mathrm{LGRAM}}$ of LGRAM′ with machine M' and input

$$\mathrm{B} = \beta \xleftarrow{\$} \{0,1\}, \quad W_0 = w_b, K_0 = k_0, \quad W_1 = w_b, K_1 = k_1, \quad T' = 0, \mathsf{wdata}' = \bot,$$

where k_0, k_1 are random PPRF keys. We will consider hybrids that effectively changes the input to M' by altering $\widehat{M}$ and $\{L_i\}_i$ given to the adversary, so we describe them by specifying the changed input.

- H^b_0. This is just $\mathsf{Exp}^b_{\mathrm{LGRAM}}$, i.e.,

$$\mathrm{B} = \beta, \quad W_\beta = w_b, \quad K_\beta = k_\beta, \quad W_{1-\beta} = w_b, \quad K_{1-\beta} = k_{1-\beta}, \quad T' = 0, \quad \mathsf{wdata}' = \bot.$$

- H^b_1. In this hybrid, we change $W_{1-\mathrm{B}}$ to w_{1-b}, i.e.,

$$\mathrm{B} = \beta, \quad W_\beta = w_b, \quad K_\beta = k_\beta, \quad W_{1-\beta} = \boxed{w_{1-b}}, \quad K_{1-\beta} = k_{1-\beta}, \quad T' = 0, \quad \mathsf{wdata}' = \bot.$$

  The only place M' uses $W_{1-\mathrm{B}}$ is the branch $t > T'$ in Step 3 of M'. But $T' = 0$ and that branch is never taken. Therefore, for each $b \in \{0,1\}$, the executions of M' in H^b_0 and H^b_1 satisfy the condition of fixed-memory security of LGRAM′, and $\mathsf{H}^b_0 \approx \mathsf{H}^b_1$.

- $\mathsf{H}^b_{2,\mathrm{t}}$ for $\mathrm{t} = 0, \ldots, T$, where T is the (common) running time of $M^{\cdots}(w_0)$ and $M^{\cdots}(w_1)$. In this hybrid, we increase T' to t, i.e.,

$$\mathrm{B} = \beta, \quad W_\beta = w_b, \quad K_\beta = k_\beta, \quad W_{1-\beta} = w_{1-b}, \quad K_{1-\beta} = k_{1-\beta}, \quad T' = \boxed{\mathrm{t}}, \quad \mathsf{wdata}' = \bot.$$

  Clearly, $\mathsf{H}^b_1 \equiv \mathsf{H}^b_{2,0}$ for each $b \in \{0,1\}$.

Claim 17 (¶). $\mathsf{H}^b_{2,\mathsf{t}-1} \equiv \mathsf{H}^b_{2,\mathsf{t}}$ *for all* $\mathsf{t} \in [T]$ *and* $b \in \{0,1\}$.

- H^b_3. In this hybrid, we set T' to T, i.e.,

$$\mathsf{B} = \beta, \qquad W_\beta = w_b, \quad K_\beta = k_\beta, \qquad W_{1-\beta} = w_{1-b}, \quad K_{1-\beta} = k_{1-\beta}, \quad T' = \boxed{T}, \ \mathsf{wdata}' = \bot.$$

  By definition, $\mathsf{H}^b_{2,T} \equiv \mathsf{H}^b_3$ for each $b \in \{0,1\}$.

- H^b_4. In this hybrid, we rename $(\beta \oplus b)$ to γ. It can be described as

$$\mathsf{B} = \boxed{\gamma \oplus b}, \ W_{\boxed{\gamma}} = w_{\boxed{0}}, K_{\boxed{\gamma}} = k_{\boxed{0}}, \quad W_{\boxed{1-\gamma}} = w_{\boxed{1}}, \quad K_{\boxed{1-\gamma}} = k_{\boxed{1}}, \quad T' = T, \quad \mathsf{wdata}' = \bot.$$

  Since the change is syntactical, $\mathsf{H}^b_3 \equiv \mathsf{H}^b_4$ for each $b \in \{0,1\}$.

In H^0_4 and H^1_4, the machine M' fully executes both $M^{\cdots}(w_0)$ on the copy γ and $M^{\cdots}(w_1)$ on the copy $(1-\gamma)$. Their only difference is from which execution M' takes $(\mathsf{done}_t, \tau_t, i_t, \mathsf{out}_t)$. The two executions of M satisfy the constraint of fixed-address security, so their $(\mathsf{done}_t, \tau_t, i_t, \mathsf{out}_t)$ sequences are identical. Therefore, $\mathsf{H}^0_4 \approx \mathsf{H}^1_4$ by the fixed-memory security of LGRAM′.

We conclude that $\mathsf{Exp}^0_{\mathsf{LGRAM}} \equiv \mathsf{H}^0_0 \approx \mathsf{H}^1_0 \equiv \mathsf{Exp}^1_{\mathsf{LGRAM}}$ for the constructed scheme.

It is clear that the time of Eval is instance-specific if so is that of Eval′. Note that the number of hybrids is polynomial in (a polynomial upper bound of) the instance running time T, not $T_{\max}$. Therefore, the proof shows that LGRAM inherits the (un-)boundedness of the security of LGRAM′. □

Proof (Claim 17). We further alter the input to M' to show $\mathsf{H}^b_{2,\mathsf{t}-1} \approx \mathsf{H}^b_{2,\mathsf{t}}$.

- G_0. This is just $\mathsf{H}^b_{2,\mathsf{t}-1}$, described by

$$\mathsf{B} = \beta, \quad W_\beta = w_b, \quad K_\beta = k_\beta, \quad W_{1-\beta} = w_{1-b}, \quad K_{1-\beta} = k_{1-\beta}, \qquad T' = \mathsf{t}-1, \quad \mathsf{wdata}' = \bot.$$

- G_1. In this hybrid, we puncture $k_{1-\beta}$ at t, hardwire $\mathsf{wdata}'_\mathsf{t}$, and increment T' to activate the hardwiring at the right time step. The hybrid is described by

$$\mathsf{B} = \beta, \quad W_\beta = w_b, \quad K_\beta = k_\beta, \quad W_{1-\beta} = w_{1-b}, \quad K_{1-\beta} = \boxed{\mathring{k}_{1-\beta,\{\mathsf{t}\}}}, \quad T' = \boxed{\mathsf{t}},$$

$$\mathsf{wdata}' = \boxed{\begin{cases} \bot, & \text{if } i_\mathsf{t} \in [\mathcal{T}]; \\ \mathsf{t} \| (\mathsf{wdata}_{b,\mathsf{t}} \oplus \mathsf{PRF}(k_0, \mathsf{t})) \| \mathsf{PRF}(k_1, \mathsf{t}), & \text{if } i_\mathsf{t} = \mathsf{work} \text{ and } \beta = 0; \\ \mathsf{t} \| \mathsf{PRF}(k_0, \mathsf{t}) \| (\mathsf{wdata}_{b,\mathsf{t}} \oplus \mathsf{PRF}(k_1, \mathsf{t})), & \text{if } i_\mathsf{t} = \mathsf{work} \text{ and } \beta = 1; \end{cases}}$$

  where $\mathsf{wdata}_{b,\mathsf{t}}$ is from $M^{\cdots}(w_b)$ and $\widetilde{\mathsf{wdata}}_{\beta,\mathsf{t}} = \mathsf{wdata}_{b,\mathsf{t}}$ in the execution of M'. $\mathsf{G}_0 \approx \mathsf{G}_1$ by the fixed-memory security of LGRAM′.

- G_2. In this hybrid, we make the portion of wdata' for the copy $(1-\beta)$ random, i.e.,

$$\mathsf{B} = \beta, \quad W_\beta = w_b, \quad K_\beta = k_\beta, \quad W_{1-\beta} = w_{1-b}, \quad K_{1-\beta} = \mathring{k}_{1-\beta,\{\mathsf{t}\}}, \qquad T' = \mathsf{t},$$

$$\mathsf{wdata}' = \begin{cases} \bot, & \text{if } i_\mathsf{t} \in [\mathcal{T}]; \\ \mathsf{t} \| (\mathsf{wdata}_{b,\mathsf{t}} \oplus \mathsf{PRF}(k_0, \mathsf{t})) \| \boxed{\mathsf{random}}, & \text{if } i_\mathsf{t} = \mathsf{work} \text{ and } \beta = 0; \\ \mathsf{t} \| \boxed{\mathsf{random}} \| (\mathsf{wdata}_{b,\mathsf{t}} \oplus \mathsf{PRF}(k_1, \mathsf{t})), & \text{if } i_\mathsf{t} = \mathsf{work} \text{ and } \beta = 1. \end{cases}$$

  Note that the altered portion is $\mathsf{PPRF.Eval}(k_{1-\beta}, \mathsf{t})$ in G_1. By the security of PPRF, it follows that $\mathsf{G}_1 \approx \mathsf{G}_2$.

- G_3. In this hybrid, we encode $\mathsf{wdata}_{1-b,\mathsf{t}}$ in the portion of wdata' for the copy $(1-\beta)$. It is described by

$$\mathsf{B}=\beta,\quad W_\beta=w_b,\quad K_\beta=k_\beta,\quad W_{1-\beta}=w_{1-b},\quad K_{1-\beta}=\mathring{k}_{1-\beta,\{\mathsf{t}\}},\quad T'=\mathsf{t},$$
$$\mathsf{wdata}'=\begin{cases}\bot, & \text{if } i_\mathsf{t}\in[\mathcal{T}];\\ \mathsf{t}\|(\mathsf{wdata}_{b,\mathsf{t}}\oplus\mathsf{PRF}(k_0,\mathsf{t}))\|\boxed{(\mathsf{wdata}_{1-b,\mathsf{t}}\oplus\mathsf{PRF}(k_1,\mathsf{t}))}, & \text{if } i_\mathsf{t}=\mathsf{work} \text{ and } \beta=0;\\ \mathsf{t}\|\boxed{(\mathsf{wdata}_{1-b,\mathsf{t}}\oplus\mathsf{PRF}(k_0,\mathsf{t}))}\|(\mathsf{wdata}_{b,\mathsf{t}}\oplus\mathsf{PRF}(k_1,\mathsf{t})), & \text{if } i_\mathsf{t}=\mathsf{work} \text{ and } \beta=1.\end{cases}$$

  $\mathsf{G}_2\approx\mathsf{G}_3$ by the security of PPRF.

- G_4. In this hybrid, we undo puncturing and hardwiring, i.e.,

$$\mathsf{B}=\beta,\quad W_\beta=w_b,\quad K_\beta=k_\beta,\quad W_{1-\beta}=w_{1-b},\quad K_{1-\beta}=\boxed{k_{1-\beta}},\quad T'=\mathsf{t},$$
$$\mathsf{wdata}'=\boxed{\bot}.$$

  $\mathsf{G}_3\approx\mathsf{G}_4$ by the fixed-memory security of LGRAM′, analogously to $\mathsf{G}_0\approx\mathsf{G}_1$.

By inspection, G_4 is just $\mathsf{H}^b_{2,\mathsf{t}}$. Therefore, $\mathsf{H}^b_{2,\mathsf{t}-1}\equiv\mathsf{G}_0\approx\mathsf{G}_4\equiv\mathsf{H}^b_{2,\mathsf{t}}$. □

5.2 Fixed-Address to Full Security

Given an LGRAM with fixed-address security, we construct one with full security with a transformation due to [CH16] using ORAM.

There are two technical differences of our LGRAM from existing notions of garbled RAM that are relevant to this transformation. First, the input tapes are compressed, and their digests cannot be an ORAM'ed version of their contents, because that would make the digests as long as the tape contents and our digests must be reusable while the usual ORAM is not. Second, the garbling procedure is only allowed to run in time $\mathrm{poly}(\lambda,|M|,\log T_{\max})$, which is not sufficient to initialize an ORAM for the working tape.

The first issue is resolved by copying the input tapes to the working tape before the actual computation is performed, and reading from the copy on the working tape whenever the actual computation wants to read from an input tape. During the copying stage, the access pattern to the tapes is simply sequential reading and writing, independent of the computation. The second issue is resolved by delaying ORAM initialization to evaluation time. Recall that our notion of ORAM (Definition 19) starts the physical execution with an empty physical tape, implicitly requiring on-demand initialization. This is matched by our notion of LGRAM (Definition 2), where the initial working tape is all-zero.

Similar to the transformation from fixed-memory security to fixed-address security (Section 5.1), we run two copies of the underlying machine in parallel to facilitate the security proof.

Ingredients of Construction 3. Let

- $\mathsf{LGRAM}'=(\mathsf{Compress}',\mathsf{Garble}',\mathsf{Eval}')$ be an LGRAM with fixed-address security,
- MakeORAM an ORAM with localized randomness with $(\mathsf{PartRnd},\mathsf{SimORAM})$, and
- $\mathsf{PPRF}=(\mathsf{PPRF.Puncture},\mathsf{PPRF.Eval})$ a puncturable PRF.

Machine M' Transformed from M

Lengths. $\ell'_{\mathrm{in}} = \mathrm{poly}(\lambda, |M|, \log T_{\max})$, $\ell'_{\mathrm{st}} = \mathrm{poly}(\lambda, |M|, \log T_{\max})$,
$\ell'_{\mathrm{addr}} = \ell_{\mathrm{addr}}$, $\ell'_{\mathrm{cell}} = \ell_{\mathrm{cell}}$,
$\ell'_{\mathrm{ADDR}} = \lceil \log_2(2S'_{\max}) \rceil$, ℓ'_{CELL} determined by MakeORAM.

Input (w')**.** $\mathrm{B}, W_0, W_1, K_0, K_1, T'$, see Figure 7;
R', hardwired ORAM randomness if K_{B} is punctured;
K', PPRF key for ORAM simulation;
addrS', T_0 hardwired addresses of M'.

State (st')**.** e, flag indicating whether an error is detected;
$t-1$, current time period minus one (initially 0);
$t_0 - 1$, current time step minus one (initially 0) modulo the period $(2T_0 + 4)$;
$\widetilde{\mathsf{st}}_{0,t-1}, \widetilde{\mathsf{rdata}}_{0,t-1}, \mathsf{ost}_{0,t-1,t_0-1}, \widetilde{\mathsf{st}}_{1,t-1}, \widetilde{\mathsf{rdata}}_{1,t-1}, \mathsf{ost}_{1,t-1,t_0-1}$, for the two copies of M and their ORAM;
$\mathsf{done}'_t, \mathsf{out}'_t$, output of M from the copy B of M.

Steps. Each step of the two copies of M is usually simulated by $(2T_0 + 4)$ steps of M'.

1. Decide what to do in this period:

   if $\widetilde{\ell}_{<\tau} = \sum_{\pi<\tau} |D_\pi| < t \le \sum_{\pi\le\tau} |D_\pi|$ for some $\tau \in [\mathcal{T}]$:

   copy $D_\tau[t - \widetilde{\ell}_{<\tau}]$ to ORAM for both copies of M

   else: **execute** or **simulate** one step of M

2a. Copy one cell from input tape to ORAM:

   1 step to **read** the cell

   $(T_0 + T_0)$ steps to ORAM-**write** for the copy 0 and 1

   1 step to **report** potential error, 2 step unused

2b. Execute or simulate one step of M:

   $2(1 + T_0)$ step for one step of M then ORAM-R/W for the copy 0 and/or 1

   the copy $(1 - \mathrm{B})$ is $\begin{cases} \text{simulated using } K', & \text{if } t - \widetilde{\ell}_{<\mathcal{T}+1} > T'; \\ \text{hardwired in } \mathsf{addrS}', & \text{if } t - \widetilde{\ell}_{<\mathcal{T}+1} = T'; \end{cases}$

   1 step to **report** potential error, 1 step to **output** $\mathsf{done}'_t, \mathsf{out}'_t$

3. Error handling and other details:

   ORAM randomness for non-simulated steps is from K_0, K_1, R'

   addresses of ORAM for the copy 0 (resp. 1) are mapped to those of M' by $i \mapsto 2i - 1$ (resp. $i \mapsto 2i$; i.e., interleaved)

   if an error is detected by ORAM

   idle and **report** it at the end of this period

   use ℓ_{in} extra steps to **output** W_{B} and **halt**

Figure 8. The machine M' in Construction 3.

Construction 3. Our scheme works as follows:

- $\mathsf{Compress}(1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}, \tau, D_\tau)$ runs

$$\mathsf{digest}'_\tau \xleftarrow{\$} \mathsf{Compress}'(1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}, \tau, D_\tau)$$

  and outputs $\mathsf{digest}_\tau = (|D_\tau|, \mathsf{digest}'_\tau)$.

- $\mathsf{Garble}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$ runs

$$(S'_{\max}, 1^{\ell'_{\mathrm{CELL}}}, 1^{\ell_{\mathrm{r}}}, 1^{\ell_{\mathrm{ost}}}, \{\mathsf{oRW}_{t_0}\}_{t_0\in[T_0]}) \leftarrow \mathsf{MakeORAM}\left(1^{\ell_{\mathrm{CELL}}}, 1^{\ell_{\mathrm{ADDR}}}, \sum_{\tau\in[\mathcal{T}]} |D_\tau| + T_{\max}\right),$$

  prepares M' as shown in Figure 8. It sets

$$T'_{\max} = \max\left\{2S'_{\max},\ \ell_{\mathrm{in}} + (2T_0+4)\left(T_{\max} + \sum_{\tau\in[\mathcal{T}]} |D_\tau|\right)\right\},$$

  runs

$$(\widehat{M}', \{L'_{i,b}\}_{i,b}) \xleftarrow{\$} \mathsf{Garble}'(T'_{\max}, M', \{\mathsf{digest}'_\tau\}_{\tau\in[\mathcal{T}]}),$$

  splits the labels according to the part of input to M' they select into

$$\{L^{\mathrm{B}}_b\}_b, \{L^{W_0}_{i,b}\}_{i,b}, \{L^{W_1}_{i,b}\}_{i,b}, \{L^{K_0}_{i,b}\}_{i,b}, \{L^{K_1}_{i,b}\}_{i,b}, \quad \{L^{T'}_{i,b}\}_{i,b}, \{L^{R'}_{i,b}\}_{i,b}, \{L^{K'}_{i,b}\}_{i,b}, \{L^{\mathsf{addrS}'}_{i,b}\}_{i,b},$$

  samples $\beta \xleftarrow{\$} \{0,1\}$ and three random PPRF keys k_0, k_1, k', and sets and outputs

$$\widehat{M} = (\widehat{M}', \{\text{labels for } \mathrm{B} = \beta, K_0 = k_0, K_1 = k_1, T' = 0, R' = \bot, K' = k', \mathsf{addrS}' = \bot\}),$$
$$L_{i,b} = L^{W_0}_{i,b} \| L^{W_1}_{i,b} \qquad \text{for } i \in [\ell_{\mathrm{in}}], b \in \{0,1\}.$$

- $\mathsf{Eval}^{D_1,\ldots,D_{\mathcal{T}}}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}, \{L_i\}_i)$ combines the labels for $\widehat{M}'$ in $\widehat{M}$ with $\{L_i\}_i$ to recover $\{L'_i\}_i$, and runs

$$(\mathsf{Eval}')^{D_1,\ldots,D_{\mathcal{T}}}(T_{\max}, M', \{\mathsf{digest}'_\tau\}_{\tau\in[\mathcal{T}]}, \widehat{M}', \{L'_i\}_i).$$

  If the evaluation does not report any ORAM error, the algorithm removes the empty, placeholder output (all steps in the periods of copying the input tapes, and $(2T_0+3)$ steps in the periods of executing M) from Eval', and use the stripped version as *the* output. Otherwise, the algorithm will obtain w after the error is reported, and it evaluates $M^{D_1,\ldots,D_{\mathcal{T}}}(w)$ in the clear and outputs it.

Correctness and Efficiency. Correctness and efficiency follow from those of the underlying ingredients as well as the error reporting mechanism of ORAM used by M'.

Theorem 18 (¶)**.** *Suppose in Construction 3,* LGRAM′ *is fixed-address secure (Definition 3 or 4),* MakeORAM *has localized randomness with* PartRnd, SimORAM *(Definition 20), and* PPRF *is secure (Definition 17), then the constructed scheme is (fully) secure (Definition 3 or 4) and inherits the (un-)boundedness of* LGRAM′.

Proof (Theorem 18). The proof is analogous to that of Theorem 16. We only demonstrate the key differences:

- When the randomness (e.g., after puncturing a PPRF key) is altered, M' could halt early due to error arising from the unfortunate randomness. We must argue that such case only happens with negligible probability. ORAM guarantees $2^{-\lambda}$ error (overflow) probability over the entire execution if the randomness is truly random, and by PRF security, the error probability is negligible when the randomness is pseudorandom (or a mix of pseudorandomness and true randomness, as needed in certain hybrids).
- The proof corresponding to Claim 17 is slightly more involved. Let R_t be the set produced by PartRnd, which is the index set of the randomness precisely affecting the access pattern for logical step t. The core idea is to puncture k_{1-B} at R_t and argue it can be simulated using SimORAM due to the localized randomness property. However, $\mathsf{PPRF.Eval}(k_{1-B}, i)$ for $i \in R_t$ is written into the working tape of M' at various time steps and locations, which are difficult to track. This is where the fixed-address property helps. It can be used to "decouple" or "couple" what is written during previous steps and where is read for logical step t. Following [CH19; Section 6.3], the transition sequence is as follows:
  1. (Fixed-address security.) Puncture k_{1-B} at R_t, puncture k' at t, hardwire $\{\mathsf{PPRF.Eval}(k_{1-B}, i)\}_{i \in R_t}$ into R', and hardwire $\mathsf{SimORAM}(\ldots, \mathsf{PPRF.Eval}(k', t))$ into addrS′.
  2. (PPRF security.) Change $\mathsf{PPRF.Eval}(k', t)$ in addrS′ into true randomness.
  3. (Fixed-address security.) Change $\{\mathsf{PPRF.Eval}(k_{1-B}, i)\}_{i \in R_t}$ in R' into the same true randomness used by addrS′. This step "couples" the previously written content and the later physical addresses.
  4. (PPRF security.) Change the common true randomness in both R and addrS′ into $\{\mathsf{PPRF.Eval}(k_{1-B}, i)\}_{i \in R_t}$.
  5. (Fixed-address security.) Undo puncturing and hardwiring. □

5.3 Bounded to Unbounded

So far we only obtain bounded LGRAM from Constructions 1, 2, and 3. While the actual evaluation time of the resultant scheme is instance-dependent (scaling with T instead of $T_{\max}$) and it works even if the machine access memory at large addresses (both are syntactical restrictions for bounded LGRAM), its security proof requires $T_{\max}$ to be polynomially bounded. To get rid of this restriction, we apply the transformation due to [AL18]. The idea is to generate a series of LGRAM garblings with $T'_{\max,e} = 2^e$ for $e \in [\log T_{\max}]$, each encrypted under a different key. The e^{th} garbling simulates the execution, but if the time exceeds 2^e, it outputs the secret key that decrypts the next garbling, and the evaluation can be retried. This avoids the exponential security loss because we can apply the LGRAM security to the garblings with $T'_{\max} < 2T$ and argue that the other, larger garblings are hidden by encryption, removing the need to apply the LGRAM security on those large garblings.

Machine M' Transformed from M and $T_{\max}$

Lengths. $\ell'_{\mathrm{in}} = \ell_{\mathrm{in}} + \mathrm{poly}(\lambda)$, $\ell'_{\mathrm{st}} = \ell_{\mathrm{st}} + \mathrm{O}(\log T_{\max}) + \mathrm{poly}(\lambda, |M|)$,
$\ell'_{\mathrm{addr}} = \ell_{\mathrm{addr}}$, $\ell'_{\mathrm{cell}} = \ell_{\mathrm{cell}}$,
$\ell'_{\mathrm{ADDR}} = \mathrm{poly}(\lambda, |M|, \log T_{\max})$, $\ell'_{\mathrm{CELL}} = \mathrm{poly}(\lambda, |M|, \log T_{\max})$.

Input (w')**.**
- W, short input to M;
- E, attempted time limit of M;
- K, secret key of SKE.

State (st')**.**
- $t-1$, current time period minus one (initially 0);
- $t_0 - 1$, current time step minus one (initially 0) modulo the period;
- st, state of M.

Steps. Each step of M is simulated by one period of M'.

1. When $t \le 2^E$:
   - 1 step for one step of M
   - $\mathrm{poly}(\lambda, |M|, \log T_{\max})$ steps for the memory operation of M (using a deterministic balanced binary tree of capacity 2^E to implement a virtual sparse memory for M)
   - **halt** if M halts
2. When $2^E < t < 2^E + \mathrm{poly}(\lambda)$:
   - **report** *runningtimeexceeded*
   - **output** K

Figure 9. The machine M' in Construction 4.

Ingredients of Construction 4. Let

- $\mathsf{LGRAM}' = (\mathsf{Compress}', \mathsf{Garble}', \mathsf{Eval}')$ be a bounded LGRAM, and
- $\mathsf{SKE} = (\mathsf{SKE.Gen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ a secret-key encryption scheme.

Construction 4. Our scheme works as follows:

- $\mathsf{Compress}(1^{\ell_{\mathrm{addr}}}, 1^{\ell_{\mathrm{cell}}}, \tau, D_\tau)$ is the same as $\mathsf{Compress}'$.
- $\mathsf{Garble}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]})$ prepares M' from M and $T_{\max}$ as shown in Figure 9. For $e \in [\lceil \log_2 T_{\max} \rceil]$, it samples SKE key k_e and runs

$$(\widehat{M}'_e, \{L'_{e,i,b}\}_{i,b}) \xleftarrow{\$} \mathsf{Garble}'(T'_{\max,e}, M', \{\mathsf{digest}_\tau\}_{\tau\in[\mathcal{T}]}) \qquad \text{for } e \in [\lceil \log_2 T_{\max} \rceil],$$

where $T'_{\max,e} = 2^e + \mathrm{poly}(\lambda, |M|, \log T_{\max})$. The algorithm encrypts $\widehat{M}'_e$ and $L'_{e,i,b}$ by

$$\widetilde{M}'_e \xleftarrow{\$} \mathsf{SKE.Enc}(k_e, \widehat{M}_e), \qquad \widetilde{L}'_{e,i,b} \xleftarrow{\$} \mathsf{SKE.Enc}(k_e, L'_{e,i,b}),$$

and splits the encrypted labels into $\{\widetilde{L}^W_{e,i,b}\}_{e,i,b}, \{\widetilde{L}^E_{e,i,b}\}_{e,i,b}, \{\widetilde{L}^K_{e,i,b}\}_{e,i,b}$ by the portion of input to M' they correspond to. Defining $k_{\lceil \log_2 T_{\max} \rceil + 1} = \bot$, it sets and outputs

$$\widehat{M} = \left(k_1, \left\{\widetilde{M}'_e,\ \widetilde{L}^E_{e,i,b}\text{'s and } \widetilde{L}^K_{e,i,b}\text{'s for } E = e, K = k_{e+1}\right\}_{e \in [\lceil \log_2 T_{\max} \rceil]}\right),$$
$$L_{i,b} = \widetilde{L}^W_{1,i,b} \| \cdots \| \widetilde{L}^W_{\lceil \log_2 T_{\max} \rceil, i, b}.$$

- $\mathsf{Eval}^{D_1,\ldots,D_{\mathcal{T}}}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau \in [\mathcal{T}]}, \widehat{M}, \{L_i\}_i)$ parses $\widehat{M}$ and L_i's as defined by Garble. For $e = 1, \ldots, \lceil \log_2 T_{\max} \rceil$, it retrieves k_e, either from $\widehat{M}$ if $e = 1$, or from the previous call to Eval′ if $e > 1$, and runs

$$(\mathsf{Eval}')^{D_1,\ldots,D_{\mathcal{T}}}(T_{\max}, M, \{\mathsf{digest}_\tau\}_{\tau \in [\mathcal{T}]}, \mathsf{SKE.Dec}(k_e \widetilde{M}'_e), \{\mathsf{SKE.Dec}(k_e, \widetilde{L}_{e,i})\}_i)$$
$$\to \begin{cases} \mathsf{outS}, & \text{if running time is not exceeded;} \\ (\text{prefix of } \mathsf{outS},\ k_{e+1}), & \text{if running time is exceeded.} \end{cases}$$

  The algorithm removes the empty, placeholder output from outS, use the stripped version as the output, and halt, if the running time is not exceeded. Otherwise, it attempts the next e.

Correctness and Efficiency. Those are clear by inspection.

Theorem 19 (¶). *Suppose in Construction 4,* LGRAM′ *is a (fixed-memory, fixed-address, fully) secure bounded LGRAM (Definition 4) and* SKE *is secure (Definition 18), then the constructed scheme is an unbounded LGRAM with the same level of security as* LGRAM′ *(Definition 3).*

Proof (Theorem 19). Since M' uses a deterministic balanced binary tree to implement the virtual sparse memory for M, it preserves the constraints of fixed-memory, fixed-address, or full security. Let T be the instance running time and $\bar{e} = \lceil \log_2 T \rceil$. We consider the following hybrids:

- H^b_{-1}. This is just $\mathsf{Exp}^b_{\mathrm{LGRAM}}$ for the constructed scheme, where

$$\widehat{M} : k_1,\ \left\{\widetilde{M}'_e,\ \widetilde{L}^E_{e,i,b}\text{'s and } \widetilde{L}^K_{e,i,b}\text{'s for } E = e, K = k_{e+1}\right\}_{e \in [\lceil \log_2 T_{\max} \rceil]},$$
$$L_i : \quad \left\{\widetilde{L}^W_{e,i,b}\text{'s for } W = w_b\right\}_{e \in [\lceil \log_2 T_{\max} \rceil]}.$$

- $\mathsf{H}^b_{\mathfrak{e}}$ for $\mathfrak{e} = 0, \ldots, \lceil \log_2 T_{\max} \rceil$. In this hybrid, we remove $k_{\bar{e}+1}$ as well as all the unused garblings up to time bound $2^{\mathfrak{e}}$ as follows.

$$\widehat{M} : k_1, \begin{cases} \widetilde{M}'_e,\ \widetilde{L}^E_{e,i,b}\text{'s and } \widetilde{L}^K_{e,i,b}\text{'s for } E = e, K = k_{e+1}, & \text{if } e < \bar{e}; \\ \widetilde{M}'_{\bar{e}},\ \widetilde{L}^E_{\bar{e},i,b}\text{'s and } \widetilde{L}^K_{\bar{e},i,b}\text{'s for } E = \bar{e}, K = \boxed{\bot}, & \text{if } e = \bar{e}; \\ \boxed{\text{random}}, & \text{if } e > \bar{e} \text{ and } e \le \mathfrak{e}; \\ \widetilde{M}'_e,\ \widetilde{L}^E_{e,i,b}\text{'s and } \widetilde{L}^K_{e,i,b}\text{'s for } E = e, K = k_{e+1}, & \text{if } e > \bar{e} \text{ and } e > \mathfrak{e}; \end{cases}$$
$$L_i : \begin{cases} \widetilde{L}^W_{e,i,b}\text{'s for } W = w_b, & \text{if } e \le \bar{e}; \\ \boxed{\text{random}}, & \text{if } e > \bar{e} \text{ and } e \le \mathfrak{e}; \\ \widetilde{L}^W_{e,i,b}\text{'s for } W = w_b, & \text{if } e > \bar{e} \text{ and } e > \mathfrak{e}. \end{cases}$$

To see indistinguishability:

- $\mathsf{H}^b_{-1} \approx \mathsf{H}^b_0$. Compared to H^b_{-1}, in H^b_0, the garbling $\widehat{M}'_{\bar{e}}$ has K in its input changed from $k_{\bar{e}+1}$ to $\bot$. Since $2^{\bar{e}} \geq T$, the execution of M in that garbling does not output K and this change satisfies the constraint of LGRAM security. Moreover, $T'_{\max,\bar{e}} = 2^{\bar{e}} \leq 2T$ is polynomially bounded. Therefore, $\mathsf{H}^b_{-1} \approx \mathsf{H}^b_0$ by the bounded security of LGRAM'.
- $\mathsf{H}^b_{\mathfrak{e}-1} \approx \mathsf{H}^b_{\mathfrak{e}}$. The two hybrids are different only when $\mathfrak{e} > \bar{e}$, in which case the ciphertexts under $k_{\mathfrak{e}}$ changes into random strings and $k_{\mathfrak{e}}$ is not used in the two hybrids. Therefore, $\mathsf{H}^b_{\mathfrak{e}-1} \approx \mathsf{H}^b_{\mathfrak{e}}$ by the ciphertext pseudorandomness of SKE.
- $\mathsf{H}^0_{\lceil \log_2 T_{\max} \rceil} \approx \mathsf{H}^1_{\lceil \log_2 T_{\max} \rceil}$. The difference is that the $\bar{e}$ non-erased garblings have W in their inputs changed between w_0 and w_1. By how M' works, this change satisfies the constraint of LGRAM security. Moreover, all of them have $T'_{\max,e} \leq 2T$ polynomially bounded. Therefore, $\mathsf{H}^0_{\lceil \log_2 T_{\max} \rceil} \approx \mathsf{H}^1_{\lceil \log_2 T_{\max} \rceil}$ follows from a standard hybrid argument by the bounded security of LGRAM'.

We conclude that $\mathsf{Exp}^0_{\mathrm{LGRAM}} \equiv \mathsf{H}^0_{-1} \approx \mathsf{H}^1_{-1} \equiv \mathsf{Exp}^1_{\mathrm{LGRAM}}$. □

6 PHFE for RAM

6.1 Bounded Private Input

In this section, we build a PHFE for RAM with bounded private input that is f- and x-succinct and has linear-time KeyGen and Enc and whose Dec runs in time $\mathrm{O}(T + |f| + |x|)$, ignoring polynomial factors in the security parameter. (See Definitions 5, 7, and 9.)

Recall that in PHFE for RAM with bounded private input, the functionality (resp. key, ciphertext) is associated with some RAM M and (up to exponential) time bound $T_{\max}$ (resp. f of arbitrary length, x of arbitrary length and y of some fixed polynomial length) and decryption yields $M^{f,x}(y)$ if the execution halts in time at most $T_{\max}$.

Ingredients of Construction 5. Let

- $\mathsf{ckt} = (\mathsf{ckt.Setup}, \mathsf{ckt.KeyGen}, \mathsf{ckt.Enc}, \mathsf{ckt.Dec})$ be an FE scheme for circuits,
- $\mathsf{LGRAM} = (\mathsf{LGRAM.Compress}, \mathsf{LGRAM.Garble}, \mathsf{LGRAM.Eval})$ a 2-tape LGRAM scheme,
- $\mathsf{PPRF} = (\mathsf{PPRF.Puncture}, \mathsf{PPRF.Eval})$ a puncturable PRF, and
- $\mathsf{SKE} = (\mathsf{SKE.Gen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ a secret-key encryption scheme.

Construction 5 (PHFE for RAM with bounded private input)**.** It works as follows:

- $\mathsf{Setup}(M, T_{\max})$ runs

$$(\mathsf{ckt.mpk}, \mathsf{ckt.msk}) \xleftarrow{\$} \mathsf{ckt.Setup}(1^{\cdots}, 1^{\cdots}),$$

  and outputs $(\mathsf{mpk}, \mathsf{msk}) = \big((T_{\max}, M, \mathsf{ckt.mpk}), \mathsf{ckt.msk}\big)$, where the input/circuit sizes given to ckt.Setup are appropriately chosen (see below).

- $\mathsf{KeyGen}(\mathsf{msk}, f)$ samples random strings $s_{\mathsf{PPRF}}, \mathsf{idx}'_{\mathsf{sk}}, \mathsf{lgram}'_{\mathsf{sk}}$, runs

$$\mathsf{digest}_f \xleftarrow{\$} \mathsf{LGRAM.Compress}(1^{\ell_{\mathrm{cell}}}, 1^{\ell_{\mathrm{addr}}}, 1, f),$$
$$\mathsf{ckt.sk} \xleftarrow{\$} \mathsf{ckt.KeyGen}(\mathsf{ckt.msk}, \mathsf{GenLGRAM}[\mathsf{digest}_f, s_{\mathsf{PPRF}}, \mathsf{idx}'_{\mathsf{sk}}, \mathsf{lgram}'_{\mathsf{sk}}]),$$

and outputs $\mathsf{sk} = (\mathsf{digest}_f, s_{\mathsf{PPRF}}, \mathsf{idx}'_{\mathsf{sk}}, \mathsf{lgram}'_{\mathsf{sk}}, \mathsf{ckt.sk})$. where GenLGRAM is in Figure 10.

- $\mathsf{Enc}(\mathsf{mpk}, x, y)$ samples random string k_{PPRF}, runs

$$\mathsf{digest}_x \xleftarrow{\$} \mathsf{LGRAM.Compress}(1^{\ell_{\mathsf{cell}}}, 1^{\ell_{\mathsf{addr}}}, 2, x),$$
$$\mathsf{ckt.ct} \xleftarrow{\$} \mathsf{ckt.Enc}\big(\mathsf{ckt.mpk}, \bot, (\mathsf{digest}_x, y, k_{\mathsf{PPRF}}, \mathsf{normal}, \overbrace{\bot, \bot, \bot, \bot, \bot, \bot}^{y', \mathsf{idx}_{\mathsf{ct}}, \mathsf{idx}_{\mathsf{H}}, k'_{\mathsf{idx}}, k'_{\mathsf{lgram}}, \mathsf{lgram}_{\mathsf{ct}}})\big),$$

  and outputs $\mathsf{ct} = (\mathsf{digest}_x, \mathsf{ckt.ct})$.

- $\mathsf{Dec}^{f,x,\mathsf{sk},\mathsf{ct}}(\mathsf{mpk})$ parses $\mathsf{mpk}, \mathsf{sk}, \mathsf{ct}$ as defined earlier, runs

$$(\widehat{M}, \{L_i\}_i) \xleftarrow{\$} \mathsf{ckt.Dec}^{\mathsf{GenLGRAM}[\mathsf{digest}_f, s_{\mathsf{PPRF}}, \mathsf{idx}'_{\mathsf{sk}}, \mathsf{lgram}'_{\mathsf{sk}}], \bot, \mathsf{ckt.sk}, \mathsf{ckt.ct}}(\mathsf{ckt.mpk}),$$

  and outputs $\mathsf{LGRAM.Eval}^{f,x}(T_{\max}, M, \mathsf{digest}_f, \mathsf{digest}_x, \widehat{M}, \{L_i\}_i)$.

Correctness and Efficiency. To ensure correctness, it suffices to set the input/circuit sizes of the underlying FE for circuits to be $\mathrm{poly}(\lambda, M, \log T_{\max})$ — the normal branch of GenLGRAM is just LGRAM.Eval, which runs in time $\mathrm{poly}(\lambda, M, \log T_{\max})$. This (order of) value is also sufficient for the security proof to go through. By the efficiency guarantees of its ingredients, Construction 5 is f- and x-succinct, has linear-time KeyGen and Enc, and its Dec runs in time $(T + |f| + |x|)\,\mathrm{poly}(\lambda, M, T_{\max})$.

Theorem 20 (¶). *Suppose in Construction 5,* ckt, LGRAM, PPRF *are secure (Definitions 7, 3, and 17) and* SKE *has pseudorandom ciphertexts (Definition 18), then the constructed scheme is secure (Definition 7).*

We prove security by hybridizing over the keys. We denote by

$$\mathsf{digest}_{f_q}, s_{\mathsf{PPRF},q}, \mathsf{idx}'_{\mathsf{sk},q}, \mathsf{lgram}'_{\mathsf{sk},q}$$

the content hardwired into GenLGRAM for the q^{th} secret key sk_q. In the hybrids,

$$y', \mathsf{idx}_{\mathsf{ct}}, \mathsf{idx}_{\mathsf{H}}, \mathsf{lgram}_{\mathsf{ct}}, k'_{\mathsf{idx}}, k'_{\mathsf{lgram}}$$

in the challenge ciphertext ct are used, and its decryption behavior is controlled by mode and those values. The strategies of handling pre- and post-challenge keys are different. At a high level, they work as follows:

- $\mathsf{idx}'_{\mathsf{sk},q}$ is an encryption of q so that each key "knows" its ordinal number.
- $\mathsf{idx}_{\mathsf{ct}}$ is the ordinal number of the challenge ciphertext.
- $\mathsf{idx}_{\mathsf{H}}$ indicates the progress of the proof and increases from 0 to Q:
  - initially, $q > \mathsf{idx}_{\mathsf{H}}$, decrypting ct by sk_q yields an LGRAM garbling for y_0;
  - when transitioning, $q = \mathsf{idx}_{\mathsf{H}}$, the behavior depends on mode and the relative timing of ct and sk_q,
    * when mode = hardwire, decryption takes the hardwired LGRAM garbling from either $\mathsf{lgram}'_{\mathsf{sk},q}$ (in sk_q) or $\mathsf{lgram}_{\mathsf{ct}}$ (in ct), whichever is generated later the security game (decided by comparing $\mathsf{idx}_{\mathsf{ct}}$ and q);

$$\mathsf{GenLGRAM}\left[\begin{matrix}\mathsf{digest}_f, s_{\mathsf{PPRF}},\\ \mathsf{idx}'_{\mathsf{sk}}, \mathsf{lgram}'_{\mathsf{sk}}\end{matrix}\right]\left(\begin{matrix}\mathsf{digest}_x, y, k_{\mathsf{PPRF}}, \mathsf{mode},\\ y', \mathsf{idx}_{\mathsf{ct}}, \mathsf{idx}_{\mathsf{H}}, k'_{\mathsf{idx}}, k'_{\mathsf{lgram}}, \mathsf{lgram}_{\mathsf{ct}}\end{matrix}\right)$$

Hardwired.	digest_f,	LGRAM digest of f (tape 1);
	s_{PPRF},	PPRF input for LGRAM randomness;
	$\mathsf{idx}'_{\mathsf{sk}}$,	encrypted ordinal (q^{th} query) of this key;
	$\mathsf{lgram}'_{\mathsf{sk}}$,	encrypted hardwired LGRAM garbling.
Input.	digest_x, y,	LGRAM digest of x (tape 2), short input;
	k_{PPRF},	PPRF key for LGRAM randomness;
	mode,	mode of ciphertext and security proof;
	y',	alternative short input;
	$\mathsf{idx}_{\mathsf{ct}}$,	ordinal of the challenge ciphertext (number of pre-challenge keys);
	$\mathsf{idx}_{\mathsf{H}}$,	progress of security proof (number of keys using y');
	$k'_{\mathsf{idx}}, k'_{\mathsf{lgram}}$,	SKE keys to decrypt $\mathsf{idx}'_{\mathsf{sk}}, \mathsf{lgram}'_{\mathsf{sk}}$;
	$\mathsf{lgram}_{\mathsf{ct}}$,	hardwired LGRAM garbling.
Output.	lgram,	LGRAM garbling, computed as follows.

1. If mode = normal (garbling is for y):

$$(\widehat{M}, \{L_{i,b}\}_{i,b}) \leftarrow \mathsf{LGRAM.Garble}\left(\begin{matrix}T_{\max}, M, \mathsf{digest}_f, \mathsf{digest}_x;\\ \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, s_{\mathsf{PPRF}})\end{matrix}\right)$$

output $\mathsf{lgram} = (\widehat{M}, \{L_{i,y[i]}\}_i)$

2. Otherwise, this key is decrypting the challenge ciphertext:

$\mathsf{idx}_{\mathsf{sk}} \leftarrow \mathsf{SKE.Dec}(k'_{\mathsf{idx}}, \mathsf{idx}'_{\mathsf{sk}})$

3a. If mode = hybrid or $\mathsf{idx}_{\mathsf{sk}} \neq \mathsf{idx}_{\mathsf{H}}$ (garbling is for y or y'):

$$(\widehat{M}, \{L_{i,b}\}_{i,b}) \leftarrow \mathsf{LGRAM.Garble}\left(\begin{matrix}T_{\max}, M, \mathsf{digest}_f, \mathsf{digest}_x;\\ \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, s_{\mathsf{PPRF}})\end{matrix}\right)$$

$$\textbf{output } \mathsf{lgram} = \begin{cases}(\widehat{M}, \{L_{i,y[i]}\}_i), & \text{if } \mathsf{idx}_{\mathsf{H}} < \mathsf{idx}_{\mathsf{sk}};\\ (\widehat{M}, \{L_{i,\boxed{y'}[i]}\}_i), & \text{if } \mathsf{idx}_{\mathsf{H}} \geq \mathsf{idx}_{\mathsf{sk}}.\end{cases}$$

3b. If mode = hardwire and $\mathsf{idx}_{\mathsf{sk}} = \mathsf{idx}_{\mathsf{H}}$ (garbling is hardwired):

$$\textbf{output } \mathsf{lgram} = \begin{cases}\mathsf{lgram}_{\mathsf{ct}}, & \text{if } \mathsf{idx}_{\mathsf{ct}} \geq \mathsf{idx}_{\mathsf{sk}};\\ \mathsf{SKE.Dec}(k'_{\mathsf{lgram}}, \mathsf{lgram}'_{\mathsf{sk}}), & \text{if } \mathsf{idx}_{\mathsf{ct}} < \mathsf{idx}_{\mathsf{sk}}.\end{cases}$$

Figure 10. The circuit GenLGRAM in Construction 5.

* when mode = hybrid, the behavior is the same as when $q < \mathsf{idx}_{\mathsf{H}}$;

– eventually, $q < \mathsf{idx}_{\mathsf{H}}$, decrypting ct by sk_q yields an LGRAM garbling for y_1.

Proof (Theorem 20). Let Q_1, Q_2 be the number of secret key queries in **Query I** and **Query II**, respectively. Writing k_{idx} for a random key of SKE and k for a random (non-punctured) key of PPRF, we list our hybrids.

- H_0. This is just $\mathsf{Exp}^0_{\mathrm{PHFE}}$, described as

$$\begin{array}{rlll}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{random}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}, & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: & y \leftarrow y_0, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \text{normal},\\
& y' \leftarrow \bot, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \bot, & \mathsf{idx}_{\mathsf{H}} \leftarrow \bot,\\
& k'_{\mathsf{idx}} \leftarrow \bot, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}$$

- H_1. In this hybrid, we sample $s_{\mathsf{PPRF},q}$'s without replacement, i.e.,

$$\begin{array}{rlll}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \boxed{\text{distinct}}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}, & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: & y \leftarrow y_0, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \text{normal},\\
& y' \leftarrow \bot, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \bot, & \mathsf{idx}_{\mathsf{H}} \leftarrow \bot,\\
& k'_{\mathsf{idx}} \leftarrow \bot, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}$$

  H_0 and H_1 are statistically indistinguishable.

- H_2. In this hybrid, we change $\mathsf{idx}'_{\mathsf{sk},q}$ into an encryption of q (padded to some fixed $\mathrm{poly}(\lambda)$ bits), i.e.,

$$\begin{array}{rlll}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \boxed{\mathsf{SKE.Enc}(k_{\mathsf{idx}}, q)}, & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: & y \leftarrow y_0, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \text{normal},\\
& y' \leftarrow \bot, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \bot, & \mathsf{idx}_{\mathsf{H}} \leftarrow \bot,\\
& k'_{\mathsf{idx}} \leftarrow \bot, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}$$

  $\mathsf{H}_1 \approx \mathsf{H}_2$ follows from the ciphertext pseudorandomness of SKE, as the key k_{idx} is not used anywhere else.

- H_3. In this hybrid, we prepare for hybridizing over the keys. H_3 is described as

$$\begin{array}{rlll}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: & y \leftarrow y_0, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \boxed{\text{hybrid}},\\
& y' \leftarrow \boxed{y_1}, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \boxed{Q_1}, & \mathsf{idx}_{\mathsf{H}} \leftarrow \boxed{0},\\
& k'_{\mathsf{idx}} \leftarrow \boxed{k_{\mathsf{idx}}}, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}$$

  It is readily verified that the decryption outcome (as perceived by ckt) does not change. Therefore, $\mathsf{H}_2 \approx \mathsf{H}_3$ follows from the security of ckt.

- $\mathsf{H}_{4,\mathfrak{q}}$ for $q = 0, \ldots, Q_1 + Q_2$. In this hybrid, $\mathsf{idx}_{\mathsf{H}}$ is incremented to q, i.e.,

$$\mathsf{sk}_q: \quad s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, \qquad \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), \qquad \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};$$

$$
\begin{array}{lccc}
\mathsf{ct}: & y \leftarrow y_0, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \mathsf{hybrid}, \\
& y' \leftarrow y_1, & \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, & \mathsf{idx}_{\mathsf{H}} \leftarrow \boxed{q}, \\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}
$$

$\mathsf{H}_{4,0}$ is just H_3. The proofs of indistinguishability between $\mathsf{H}_{4,\mathfrak{q}-1}$ and $\mathsf{H}_{4,\mathfrak{q}}$ depend on whether $\mathfrak{q} \le Q_1$ or $\mathfrak{q} > Q_1$.

Claim 21 (¶). $\mathsf{H}_{4,\mathfrak{q}-1} \approx \mathsf{H}_{4,\mathfrak{q}}$ *for* $\mathfrak{q} = 1, \ldots, Q_1$.

Claim 22 (¶). $\mathsf{H}_{4,\mathfrak{q}-1} \approx \mathsf{H}_{4,\mathfrak{q}}$ *for* $\mathfrak{q} = Q_1 + 1, \ldots, Q_1 + Q_2$.

- H_5. In this hybrid, we finish the hybrid argument over the keys. H_5 is described as

$$
\begin{array}{lccc}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\
\mathsf{ct}: & y \leftarrow y_0, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \mathsf{hybrid}, \\
& y' \leftarrow y_1, & \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, & \mathsf{idx}_{\mathsf{H}} \leftarrow \boxed{Q_1 + Q_2}, \\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}
$$

  H_5 is just H_{4,Q_1+Q_2}.

- H_6. In this hybrid, the challenge ciphertext becomes a normal one for y_1, i.e.,

$$
\begin{array}{lccc}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\
\mathsf{ct}: & y \leftarrow \boxed{y_1}, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \boxed{\mathsf{normal}}, \\
& y' \leftarrow \boxed{\bot}, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \boxed{\bot}, & \mathsf{idx}_{\mathsf{H}} \leftarrow \boxed{\bot}, \\
& k'_{\mathsf{idx}} \leftarrow \boxed{\bot}, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}
$$

  $\mathsf{H}_5 \approx \mathsf{H}_6$ follows from the security of ckt, analogously to $\mathsf{H}_2 \approx \mathsf{H}_3$.

- H_7. In this hybrid, we revert $\mathsf{idx}'_{\mathsf{sk},q}$ to random, i.e.,

$$
\begin{array}{lccc}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \boxed{\text{random}}, & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\
\mathsf{ct}: & y \leftarrow y_1, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \mathsf{normal}, \\
& y' \leftarrow \bot, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \bot, & \mathsf{idx}_{\mathsf{H}} \leftarrow \bot, \\
& k'_{\mathsf{idx}} \leftarrow \bot, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}
$$

  $\mathsf{H}_6 \approx \mathsf{H}_7$ by the ciphertext pseudorandomness of SKE, analogously to $\mathsf{H}_1 \approx \mathsf{H}_2$.

- H_8. In this hybrid, $s_{\mathsf{PPRF},q}$'s are sampled as specified by KeyGen, i.e.,

$$
\begin{array}{lccc}
\mathsf{sk}_q: & s_{\mathsf{PPRF},q} \xleftarrow{\$} \boxed{\text{random}}, & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}, & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\
\mathsf{ct}: & y \leftarrow y_1, & k_{\mathsf{PPRF}} \leftarrow k, & \mathsf{mode} \leftarrow \mathsf{normal}, \\
& y' \leftarrow \bot, & \mathsf{idx}_{\mathsf{ct}} \leftarrow \bot, & \mathsf{idx}_{\mathsf{H}} \leftarrow \bot, \\
& k'_{\mathsf{idx}} \leftarrow \bot, & k'_{\mathsf{lgram}} \leftarrow \bot, & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{array}
$$

  H_8 is statistically indistinguishable from H_7.

By inspection, H_8 is just $\mathsf{Exp}^1_{\mathrm{PHFE}}$, and therefore, $\mathsf{Exp}^0_{\mathrm{PHFE}} \equiv \mathsf{H}_0 \approx \mathsf{H}_8 \equiv \mathsf{Exp}^1_{\mathrm{PHFE}}$. □

Proof (Claim 21). To show indistinguishability between $\mathsf{H}_{4,\mathfrak{q}-1}$ and $\mathsf{H}_{4,\mathfrak{q}}$ when $\boxed{\mathfrak{q} \le Q_1}$, we temporarily hardwire the LGRAM garbling yielded by decryption ct using $\mathsf{sk}_\mathfrak{q}$ into $\boxed{\mathsf{lgram}_{\mathsf{ct}}}$, which is generated when both $f_\mathfrak{q}$ and x, y_0, y_1 are known.

Let $\mathring{k}_{\{\mathfrak{q}\}}$ be k punctured at the singleton set $\{\mathfrak{q}\}$. We specify the hybrids.

- G_0. This is just $\mathsf{H}_{4,\mathfrak{q}-1}$, described as

$$\begin{aligned}
\mathsf{sk}_q:\quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}:\quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow k, && \mathsf{mode} \leftarrow \mathsf{hybrid},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q}-1,\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \bot, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot.
\end{aligned}$$

- G_1. In this hybrid, we puncture k_{PPRF}, hardwire the decryption result of ct by $\mathsf{sk}_\mathfrak{q}$ into ct at $\mathsf{lgram}_{\mathsf{ct}}$, indicating hardwiring using mode, and increment $\mathsf{idx}_{\mathsf{H}}$, i.e.,

$$\begin{aligned}
\mathsf{sk}_q:\quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}:\quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \boxed{\mathring{k}_{\{\mathfrak{q}\}}}, && \mathsf{mode} \leftarrow \boxed{\text{hardwire}},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \boxed{\mathfrak{q}},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \bot, &&
\end{aligned}$$

$$\mathsf{lgram}_{\mathsf{ct}} \leftarrow \boxed{(\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from } \mathsf{LGRAM.Garble}\left(\begin{matrix} T_{\max}, M, \mathsf{digest}_{f_\mathfrak{q}}, \mathsf{digest}_x; \\ \mathsf{PPRF.Eval}(k, s_{\mathsf{PPRF},\mathfrak{q}}) \end{matrix}\right)}.$$

$\mathsf{G}_0 \approx \mathsf{G}_1$ follows from the security of ckt.

- G_2. In this hybrid, we change the LGRAM randomness from $\mathsf{PPRF.Eval}(k, \mathfrak{q})$ to true randomness, i.e.,

$$\begin{aligned}
\mathsf{sk}_q:\quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}:\quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \mathring{k}_{\{\mathfrak{q}\}}, && \mathsf{mode} \leftarrow \text{hardwire},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \bot, &&
\end{aligned}$$

$$\mathsf{lgram}_{\mathsf{ct}} \leftarrow (\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from } \mathsf{LGRAM.Garble}\left(\begin{matrix} T_{\max}, M, \mathsf{digest}_{f_\mathfrak{q}}, \mathsf{digest}_x; \\ \boxed{\text{random}} \end{matrix}\right).$$

$\mathsf{G}_1 \approx \mathsf{G}_2$ follows from the security of PPRF.

- G_3. In this hybrid, we change the hardwired LGRAM garbling into one for y_1, i.e.,

$$\begin{aligned}
\mathsf{sk}_q:\quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}:\quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \mathring{k}_{\{\mathfrak{q}\}}, && \mathsf{mode} \leftarrow \text{hardwire},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},
\end{aligned}$$

$$k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, \qquad k'_{\mathsf{lgram}} \leftarrow \bot,$$
$$\mathsf{lgram}_{\mathsf{ct}} \leftarrow (\widehat{M}, \{L_{i,\boxed{y_1}[i]}\}_i) \text{ from } \mathsf{LGRAM.Garble}\left(\begin{matrix} T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \\ \text{random} \end{matrix}\right).$$

$\mathsf{G}_2 \approx \mathsf{G}_3$ follows from the security of LGRAM.

- G_4. In this hybrid, the hardwired LGRAM garbling is reverted to be generated with pseudorandomness $\mathsf{PPRF.Eval}(k, \mathfrak{q})$, i.e.,

$$\begin{aligned} \mathsf{sk}_q: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), & & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\ \mathsf{ct}: \quad & y \leftarrow y_0, & & k_{\mathsf{PPRF}} \leftarrow \mathring{k}_{\{\mathfrak{q}\}}, & & \mathsf{mode} \leftarrow \mathsf{hardwire}, \\ & y' \leftarrow y_1, & & \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, & & \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q}, \\ & k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, & & k'_{\mathsf{lgram}} \leftarrow \bot, & & \end{aligned}$$
$$\mathsf{lgram}_{\mathsf{ct}} \leftarrow (\widehat{M}, \{L_{i,y_1[i]}\}_i) \text{ from } \mathsf{LGRAM.Garble}\left(\begin{matrix} T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \\ \boxed{\mathsf{PPRF.Eval}(k, s_{\mathsf{PPRF},\mathfrak{q}})} \end{matrix}\right).$$

$\mathsf{G}_3 \approx \mathsf{G}_4$ follows from the security of PPRF.

- G_5. In this hybrid, k_{PPRF} is no longer punctured and hardwiring is undone, i.e.,

$$\begin{aligned} \mathsf{sk}_q: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), & & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\ \mathsf{ct}: \quad & y \leftarrow y_0, & & k_{\mathsf{PPRF}} \leftarrow \boxed{k}, & & \mathsf{mode} \leftarrow \mathsf{hybrid}, \\ & y' \leftarrow y_1, & & \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, & & \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q}, \\ & k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, & & k'_{\mathsf{lgram}} \leftarrow \bot, & & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \boxed{\bot}. \end{aligned}$$

$\mathsf{G}_4 \approx \mathsf{G}_5$ follows from the security of ckt.

By inspection, G_5 is exactly $\mathsf{H}_{4,\mathfrak{q}}$. Therefore, $\mathsf{H}_{4,\mathfrak{q}-1} \equiv \mathsf{G}_0 \approx \mathsf{G}_5 \equiv \mathsf{H}_{4,\mathfrak{q}}$. □

Proof (Claim 22). To show indistinguishability between $\mathsf{H}_{4,\mathfrak{q}-1}$ and $\mathsf{H}_{4,\mathfrak{q}}$ when $\boxed{\mathfrak{q} > Q_1}$, we temporarily hardwire the LGRAM garbling yielded by decryption ct using $\mathsf{sk}_{\mathfrak{q}}$ into $\boxed{\mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}}}$, generated when both x, y_0, y_1 and $f_{\mathfrak{q}}$ are known. Let $\mathring{k}_{\{\mathfrak{q}\}}$ be k punctured at $\{\mathfrak{q}\}$ and k_{lgram} a random secret key of SKE. We specify the hybrids.

- G_0. This is just $\mathsf{H}_{4,\mathfrak{q}-1}$, described as

$$\begin{aligned} \mathsf{sk}_{q\neq\mathfrak{q}}: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, & & \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), & & \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random}; \\ \mathsf{ct}: \quad & y \leftarrow y_0, & & k_{\mathsf{PPRF}} \leftarrow k, & & \mathsf{mode} \leftarrow \mathsf{hybrid}, \\ & y' \leftarrow y_1, & & \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, & & \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q}-1, \\ & k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, & & k'_{\mathsf{lgram}} \leftarrow \bot, & & \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot; \\ \mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, & & \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), & & \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \text{random}. \end{aligned}$$

- G_1. In this hybrid, we encrypt the LGRAM garbling into $\mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}}$, i.e.,

$$\mathsf{sk}_{q\neq\mathfrak{q}}: \quad s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, \qquad \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), \qquad \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};$$

$$
\begin{aligned}
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow k, && \mathsf{mode} \leftarrow \mathsf{hybrid},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q}-1,\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \bot, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;\\
\mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), &&\\
& \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \boxed{\mathsf{SKE.Enc}\left(k_{\mathsf{lgram}}, \begin{array}{r}(\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from}\\ \mathsf{LGRAM.Garble}\left(T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \right.\\ \left.\mathsf{PPRF.Eval}(k, s_{\mathsf{PPRF},\mathfrak{q}})\right)\end{array}\right)}.
\end{aligned}
$$

$\mathsf{G}_0 \approx \mathsf{G}_1$ by the ciphertext pseudorandomness of SKE.

- G_2. In this hybrid, we puncture k at $\{\mathfrak{q}\}$, activate the hardwiring using k'_{lgram}, mode, and increment $\mathsf{idx}_{\mathsf{H}}$, i.e.,

$$
\begin{aligned}
\mathsf{sk}_{q\neq\mathfrak{q}}: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \boxed{\mathring{k}_{\{\mathfrak{q}\}}}, && \mathsf{mode} \leftarrow \boxed{\mathsf{hardwire}},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \boxed{\mathfrak{q}},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \boxed{k_{\mathsf{lgram}}}, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;\\
\mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), &&\\
& \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}\left(k_{\mathsf{lgram}}, \begin{array}{r}(\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from}\\ \mathsf{LGRAM.Garble}\left(T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \right.\\ \left.\mathsf{PPRF.Eval}(k, s_{\mathsf{PPRF},\mathfrak{q}})\right)\end{array}\right).
\end{aligned}
$$

$\mathsf{G}_1 \approx \mathsf{G}_2$ follows from the security of ckt.

- G_3. In this hybrid, we generate the hardwired garbling with true randomness instead of $\mathsf{PPRF.Eval}(k, \mathfrak{q})$, i.e.,

$$
\begin{aligned}
\mathsf{sk}_{q\neq\mathfrak{q}}: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \mathring{k}_{\{\mathfrak{q}\}}, && \mathsf{mode} \leftarrow \mathsf{hardwire},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow k_{\mathsf{lgram}}, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;\\
\mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), &&\\
& \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}\left(k_{\mathsf{lgram}}, \begin{array}{r}(\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from}\\ \mathsf{LGRAM.Garble}\left(T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \right.\\ \left.\boxed{\text{random}}\right)\end{array}\right).
\end{aligned}
$$

$\mathsf{G}_2 \approx \mathsf{G}_3$ follows from the security of PPRF.

- G_4. In this hybrid, the hardwired garbling becomes one for y_1, i.e.,

$$
\mathsf{sk}_{q\neq\mathfrak{q}}: \quad s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, \quad \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), \quad \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};
$$

$$
\begin{aligned}
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \mathring{k}_{\{\mathfrak{q}\}}, && \mathsf{mode} \leftarrow \mathsf{hardwire},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow k_{\mathsf{lgram}}, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;\\
\mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), &&\\
& \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}\left(k_{\mathsf{lgram}}, \begin{array}{c}(\widehat{M}, \{L_{i,\boxed{y_1}[i]}\}_i) \text{ from}\\ \mathsf{LGRAM.Garble}\left(T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \begin{array}{c}\\ \text{random}\end{array}\right)\end{array}\right).&&&&
\end{aligned}
$$

$\mathsf{G}_3 \approx \mathsf{G}_4$ follows from the security of LGRAM.

- G_5. In this hybrid, the randomness for generating the hardwired LGRAM garbling is reverted to be pseudorandom, i.e.,

$$
\begin{aligned}
\mathsf{sk}_{q\neq\mathfrak{q}}: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \mathring{k}_{\{\mathfrak{q}\}}, && \mathsf{mode} \leftarrow \mathsf{hardwire},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow k_{\mathsf{lgram}}, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;\\
\mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), &&\\
& \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}\left(k_{\mathsf{lgram}}, \begin{array}{c}(\widehat{M}, \{L_{i,y_1[i]}\}_i) \text{ from}\\ \mathsf{LGRAM.Garble}\left(T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \begin{array}{c}\\ \boxed{\mathsf{PPRF.Eval}(k, \mathfrak{q})}\end{array}\right)\end{array}\right).&&&&
\end{aligned}
$$

$\mathsf{G}_4 \approx \mathsf{G}_5$ follows from the security of PPRF.

- G_6. In this hybrid, the hardwiring is deactivated by reverting most of the changes made in the transition from G_1 to G_2, i.e.,

$$
\begin{aligned}
\mathsf{sk}_{q\neq\mathfrak{q}}: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow \boxed{k}, && \mathsf{mode} \leftarrow \boxed{\mathsf{hybrid}},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \boxed{\bot}, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;\\
\mathsf{sk}_{\mathfrak{q}>Q_1}: \quad & s_{\mathsf{PPRF},\mathfrak{q}} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, \mathfrak{q}), &&\\
& \mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}} \xleftarrow{\$} \mathsf{SKE.Enc}\left(k_{\mathsf{lgram}}, \begin{array}{c}(\widehat{M}, \{L_{i,y_1[i]}\}_i) \text{ from}\\ \mathsf{LGRAM.Garble}\left(T_{\max}, M, \mathsf{digest}_{f_{\mathfrak{q}}}, \mathsf{digest}_x; \begin{array}{c}\\ \mathsf{PPRF.Eval}(k, \mathfrak{q})\end{array}\right)\end{array}\right).&&&&
\end{aligned}
$$

$\mathsf{G}_5 \approx \mathsf{G}_6$ follows from the security of ckt.

- G_7. In this hybrid, we revert $\mathsf{lgram}'_{\mathsf{sk},\mathfrak{q}}$ to random, i.e.,

$$
\begin{aligned}
\mathsf{sk}_{q\neq\mathfrak{q}}: \quad & s_{\mathsf{PPRF},q} \xleftarrow{\$} \text{distinct}, && \mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q), && \mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \text{random};\\
\mathsf{ct}: \quad & y \leftarrow y_0, && k_{\mathsf{PPRF}} \leftarrow k, && \mathsf{mode} \leftarrow \mathsf{hybrid},\\
& y' \leftarrow y_1, && \mathsf{idx}_{\mathsf{ct}} \leftarrow Q_1, && \mathsf{idx}_{\mathsf{H}} \leftarrow \mathfrak{q},\\
& k'_{\mathsf{idx}} \leftarrow k_{\mathsf{idx}}, && k'_{\mathsf{lgram}} \leftarrow \bot, && \mathsf{lgram}_{\mathsf{ct}} \leftarrow \bot;
\end{aligned}
$$

$\mathsf{sk}_{q>Q_1}$: $s_{\mathsf{PPRF},q} \xleftarrow{\$}$ distinct, $\mathsf{idx}'_{\mathsf{sk},q} \xleftarrow{\$} \mathsf{SKE.Enc}(k_{\mathsf{idx}}, q)$, $\mathsf{lgram}'_{\mathsf{sk},q} \xleftarrow{\$} \boxed{\perp}$.

$\mathsf{G}_6 \approx \mathsf{G}_7$ by the ciphertext pseudorandomness of SKE.

By inspection, G_7 is exactly $\mathsf{H}_{4,q}$. Therefore, $\mathsf{H}_{4,q-1} \equiv \mathsf{G}_0 \approx \mathsf{G}_7 \equiv \mathsf{H}_{4,q}$. □

6.2 Full-Fledged PHFE for RAM

In this section, we build a full-fledged PHFE for RAM that is f-succinct and has rate-2 ciphertext and linear-time KeyGen and Enc and whose Dec runs in time $O(T + |f| + |x| + |y|)$, ignoring polynomial factors in the security parameter. (See Definitions 5, 7, and 10.)

Recall that in full-fledged PHFE for RAM, the functionality (resp. key, ciphertext) is associated with some RAM M and (up to exponential) time bound $T_{\max}$ (resp. f, (x, y), each of arbitrary length) and decryption yields $M^{f,x\|y}()$ if the execution halts in time at most $T_{\max}$.

Ingredients of Construction 6. Let

- $\mathsf{PHFE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ be a PHFE scheme for RAM with bounded private input that is f'- and x'-succinct and has linear-time KeyGen′ and Enc′ and whose Dec′ runs in time $(T' + |f'| + |x'|)\,\mathrm{poly}(\lambda, M', \log T'_{\max})$, and
- PRF a pseudorandom function.

Construction 6 (full-fledged PHFE for RAM)**.** Our scheme works as follows:

- $\mathsf{Setup}(M, T_{\max})$ runs and outputs
$$(\mathsf{mpk}, \mathsf{msk}) = (\mathsf{mpk}', \mathsf{msk}') \xleftarrow{\$} \mathsf{Setup}'(M', T'_{\max}),$$
where M' is shown in Figure 11 and $T'_{\max} = T_{\max} + \mathrm{poly}(\lambda, \log T_{\max})$.
- $\mathsf{KeyGen}(\mathsf{msk}, f)$ pads f by setting
$$f' \leftarrow (f[1]\|0^{\ell_{\mathrm{cell}}})\| \cdots \|(\overbrace{f[|f|]\|0^{\ell_{\mathrm{cell}}}}^{\text{one cell on an input tape of } M'}),$$
and runs and outputs
$$\mathsf{sk} = \mathsf{sk}' \xleftarrow{\$} \mathsf{KeyGen}'(\mathsf{msk}', f').$$
- $\mathsf{Enc}(\mathsf{mpk}, x, y)$ samples $\beta \xleftarrow{\$} \{0,1\}$, a PRF key k_β, and a string $w_{1-\beta}$ of the same length as y. It pads x and appends to it an interleaved version of w_0, w_1, where w_β encrypts y, by setting
$$\begin{aligned} w_\beta[i] &\leftarrow y[i] \oplus \mathsf{PRF}(k_\beta, i) \qquad \text{for } i \in [|y|], \\ w &\leftarrow \underbrace{(w_0[1]\|w_1[1])}_{\text{one cell on an input tape of } M'}\| \cdots \|(w_0[|y|]\|w_1[|y|]), \\ x' &\leftarrow \overbrace{(x[0]\|0^{\ell_{\mathrm{cell}}})}\| \cdots \|(x[|x|]\|0^{\ell_{\mathrm{cell}}})\|w, \qquad y' \leftarrow (|x|, \beta, k_\beta). \end{aligned}$$
The algorithm runs $\mathsf{ct}' \xleftarrow{\$} \mathsf{Enc}'(\mathsf{mpk}', x', y')$ and outputs $\mathsf{ct} = (\mathsf{ct}', w)$.
- $\mathsf{Dec}^{f,x,\mathsf{sk},\mathsf{ct}}(\mathsf{mpk})$ prepares oracles for f', x' from f, x, ct as specified in KeyGen, Enc. It runs and outputs
$$(\mathsf{Dec}')^{f',x',\mathsf{sk}',\mathsf{ct}'}(\mathsf{mpk}').$$

2-Tape RAM M'

Lengths. $\ell'_{\mathrm{st}} = \ell_{\mathrm{st}} + \mathrm{poly}(\lambda, |M|, \log T_{\max})$,
the state of M, the location of the last-read cell;
$\ell'_{\mathrm{in}} = \mathrm{poly}(\lambda, |M|, \log T_{\max}) + 1 + \mathrm{poly}(\lambda, |M|, \log T_{\max})$,
to encode $|x|$, choice bit β, and PRF key k_β;
$\ell'_{\mathrm{addr}} = \ell_{\mathrm{addr}} + 1$, $\ell'_{\mathrm{cell}} = 2\ell_{\mathrm{cell}}$,
twice as many cells with each cell twice as large
on an input tape (to encode x, y_0, y_1 in x');
$\ell'_{\mathrm{ADDR}} = \ell_{\mathrm{ADDR}}$, $\ell'_{\mathrm{CELL}} = \ell_{\mathrm{CELL}}$,
exactly the working tape of M.

Input. $w' = (|x|, \beta, k_\beta)$, length of x, choice bit, PRF key for y or y_β;
$D'_1 = f'$, $\ell'_1 = \ell_1$, padded version of f;
$D'_2 = x'$, $\ell'_2 = |x| + |y|$,
padded version of x followed by
interleaved encryption of y or y_0, y_1.

Steps. Each step of M' replicates one step of M
by translating its memory access as follows.

1. If M reads $D_1[i] = f[i]$:
read $D'_1[i] = f'[i] = f[i] \| 0^{\ell_{\mathrm{cell}}}$ in this step
provide $f[i]$ in the next step (similar below)

2a. If M reads $D_2[i] = x[i]$ for $i \in [|x|]$:
read $D'_2[i] = x'[i] = x[i] \| 0^{\ell_{\mathrm{cell}}}$
provide $x[i]$

2b. If M reads $D_2[|x| + i] = y[i]$ for $i \in [|y|]$:
read $D'_2[|x| + i] = w[i] = w_0[i] \| w_1[i]$
provide $w_\beta[i] \oplus \mathsf{PRF}(k_\beta, i)$

3. If M reads $D_{\mathsf{work}}[i]$:
read and **provide** $D'_{\mathsf{work}}[i]$

Figure 11. The machine M' in Construction 6.

Correctness and Efficiency. Correctness is immediate by inspection. The construction scheme incurs an additional storage twice as long as y in ct, no additional machine time, and constant-factor additional decryption time. Since $|M'| = \mathrm{poly}(\lambda, |M|, \log T_{\mathrm{max}})$ and $T'_{\mathrm{max}} = T_{\mathrm{max}} + \mathrm{poly}(\lambda, M, \log T_{\mathrm{max}})$, with parameters for PHFE' denoted with primes,

$$\begin{aligned}
|\mathsf{sk}| = \quad |\mathsf{sk}'| &= \mathrm{poly}(\lambda, |M'|, \log T'_{\mathrm{max}}) \\
&= \mathrm{poly}(\lambda, |M|, \log T_{\mathrm{max}}), \\
|\mathsf{ct}| = 2|y| + |\mathsf{ct}'| &= 2|y| + \mathrm{poly}(\lambda, |M'|, \log T'_{\mathrm{max}}) \\
&= 2|y| + \mathrm{poly}(\lambda, |M|, \log T_{\mathrm{max}}), \\
T_{\mathsf{KeyGen}} = \quad T'_{\mathsf{KeyGen}} &= |f'| \,\mathrm{poly}(\lambda, |M'|, \log T'_{\mathrm{max}}) \\
&= |f| \,\mathrm{poly}(\lambda, |M|, \log T_{\mathrm{max}}), \\
T_{\mathsf{Enc}} = \quad T'_{\mathsf{Enc}} &= |x'| \,\mathrm{poly}(\lambda, |M'|, \log T'_{\mathrm{max}}) \\
&= (|x| + |y|) \,\mathrm{poly}(\lambda, |M|, \log T_{\mathrm{max}}), \\
T_{\mathsf{Dec}} = \quad T'_{\mathsf{Dec}} &= (T' + |f'| + |x'|) \,\mathrm{poly}(\lambda, |M'|, \log T'_{\mathrm{max}}) \\
&= (T + |f| + |x| + |y|) \,\mathrm{poly}(\lambda, |M|, \log T_{\mathrm{max}}).
\end{aligned}$$

Theorem 23 (¶). *Suppose in Construction 6,* $\mathsf{PHFE}', \mathsf{PRF}$ *are secure (Definitions 7 and 17), then the constructed scheme is secure (Definition 7).*

Proof (Theorem 23). Let $\beta \xleftarrow{\$} \{0,1\}$ be a random bit and $k_\beta, k_{1-\beta}$ two random PRF keys. We describe how the challenge ciphertext is generated in each hybrid.

- H_0^b. This is just $\mathsf{Exp}^b_{\mathrm{PHFE}}$, where

$$\begin{aligned}
&w_\beta[i] \leftarrow y_b[i] \oplus \mathsf{PRF}(k_\beta, i), \qquad w_{1-\beta}[i] \xleftarrow{\$} \text{random}, \\
&x' \leftarrow (x[1] \| 0^{\ell_{\mathrm{cell}}}) \| \cdots \| (x[|x|] \| 0^{\ell_{\mathrm{cell}}}) \| (w_0[1] \| w_1[i]) \| \cdots \| (w_0[|y|] \| w_1[|y|]), \\
&\mathsf{ct}' \xleftarrow{\$} \mathsf{Enc}'\big(\mathsf{mpk}, x', (|x|, \beta, k_\beta)\big).
\end{aligned}$$

- H_1^b. In this hybrid, we make $w_{1-\beta}$ an encryption of y_{1-b} under $k_{1-\beta}$, i.e.,

$$\begin{aligned}
&w_\beta[i] \leftarrow y_b[i] \oplus \mathsf{PRF}(k_\beta, i), \qquad \boxed{w_{1-\beta}[i] \leftarrow y_{1-b}[i] \oplus \mathsf{PRF}(k_{1-\beta}, i)}, \\
&x' \leftarrow (x[1] \| 0^{\ell_{\mathrm{cell}}}) \| \cdots \| (x[|x|] \| 0^{\ell_{\mathrm{cell}}}) \| (w_0[1] \| w_1[i]) \| \cdots \| (w_0[|y|] \| w_1[|y|]), \\
&\mathsf{ct}' \xleftarrow{\$} \mathsf{Enc}'\big(\mathsf{mpk}, x', (|x|, \beta, k_\beta)\big).
\end{aligned}$$

  $\mathsf{H}_0^b \approx \mathsf{H}_1^b$ for each $b \in \{0,1\}$ by the security of PRF.

- H_2^b. In this hybrid, we rename $(\beta \oplus b)$ to γ, making it

$$\begin{aligned}
&w_{\boxed{\gamma}}[i] \leftarrow y_{\boxed{0}}[i] \oplus \mathsf{PRF}(k_{\boxed{\gamma}}, i), \qquad w_{\boxed{1-\gamma}}[i] \leftarrow y_{\boxed{1}}[i] \oplus \mathsf{PRF}(k_{\boxed{1-\gamma}}, i), \\
&x' \leftarrow (x[1] \| 0^{\ell_{\mathrm{cell}}}) \| \cdots \| (x[|x|] \| 0^{\ell_{\mathrm{cell}}}) \| (w_0[1] \| w_1[i]) \| \cdots \| (w_0[|y|] \| w_1[|y|]), \\
&\mathsf{ct}' \xleftarrow{\$} \mathsf{Enc}'\big(\mathsf{mpk}, x', (|x|, \boxed{\gamma \oplus b}, k_{\boxed{\gamma \oplus b}})\big),
\end{aligned}$$

  where $\gamma \xleftarrow{\$} \{0,1\}$. This change is conceptual, hence $\mathsf{H}_1^b \equiv \mathsf{H}_2^b$ for each $b \in \{0,1\}$.

$\mathsf{H}_2^0 \approx \mathsf{H}_2^1$ follows from the security of PHFE'. Therefore, $\mathsf{Exp}^0_{\mathrm{PHFE}} \equiv \mathsf{H}_0^0 \approx \mathsf{H}_0^1 \equiv \mathsf{Exp}^1_{\mathrm{PHFE}}$. □

Acknowledgement. The authors were supported by NSF grants CNS-1528178, CNS-1929901, CNS-1936825 (CAREER), CNS-2026774, a Hellman Fellowship, a JP Morgan AI Research Award, the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236, and a subcontract No. 2017-002 through Galois. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[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, Part II*, volume 9986 of *LNCS*, pages 3–30. Springer, Heidelberg, October / November 2016.

[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, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 112–141. Springer, Heidelberg, December 2019.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332. Springer, Heidelberg, August 2019.

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 252–279. Springer, Heidelberg, August 2017.

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Heidelberg, November 2018.

[ALdP11] Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 90–108. Springer, Heidelberg, March 2011.

[AM18] Shweta Agrawal and Monosij Maitra. FE and iO for turing machines from minimal assumptions. In Amos Beimel and Stefan Dziembowski, editors,

TCC 2018, Part II, volume 11240 of *LNCS*, pages 473–512. Springer, Heidelberg, November 2018.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Heidelberg, January 2016.

[AT20] Nuttapong Attrapadung and Junichi Tomida. Unbounded dynamic predicate compositions in ABE from standard assumptions. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 405–436. Springer, Heidelberg, December 2020.

[Att16] Nuttapong Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 591–623. Springer, Heidelberg, December 2016.

[Ben89] Charles Henry Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[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 ACM STOC*, pages 439–448. ACM Press, June 2015.

[BHMW21] Elette Boyle, Justin Holmgren, Fermi Ma, and Mor Weiss. On the security of doubly efficient PIR. Cryptology ePrint Archive, Report 2021/1113, 2021. https://eprint.iacr.org/2021/1113.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 662–693. Springer, Heidelberg, November 2017.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In Yevgeniy Dodis and Jesper Buus Nielsen, editors,

TCC 2015, Part II, volume 9015 of *LNCS*, pages 306–324. Springer, Heidelberg, March 2015.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[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*, pages 179–190. ACM, January 2016.

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Adaptive succinct garbled RAM or: How to delegate your database. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 61–90. Springer, Heidelberg, October / November 2016.

[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, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Heidelberg, August 2017.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016*, pages 169–178. ACM, January 2016.

[CH19] Ran Canetti and Justin Holmgren. Succinct garbled RAM. Cryptology ePrint Archive, Report 2015/388, version 20190517:011532, 2019. https://eprint.iacr.org/archive/2015/388/20190517:011532.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 429–437. ACM Press, June 2015.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 694–726. Springer, Heidelberg, November 2017.

[DIJ+13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 519–535. Springer, Heidelberg, August 2013.

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor,

CRYPTO 2010, volume 6223 of *LNCS*, pages 649–665. Springer, Heidelberg, August 2010.

[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 FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[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, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 515–544. Springer, Heidelberg, August 2018.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 419–442. Springer, Heidelberg, October / November 2016.

[GS18a] Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 535–565. Springer, Heidelberg, April / May 2018.

[GS18b] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 425–454. Springer, Heidelberg, November 2018.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015.

[GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 700–730. Springer, Heidelberg, May / June 2022.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 251–281. Springer, Heidelberg, May 2019.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 60–73, New York, NY, USA, 2021. Association for Computing Machinery.

[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, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

[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 ACM STOC*, pages 419–428. ACM Press, June 2015.

[KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Simple and generic constructions of succinct functional encryption. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 187–217. Springer, Heidelberg, March 2018.

[KNTY19] Fuyuki Kitagawa, Ryo Nishimaki, Keisuke Tanaka, and Takashi Yamakawa. Adaptively secure and succinct functional encryption: Improving security and efficiency, simultaneously. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 521–551. Springer, Heidelberg, August 2019.

[LL20] Huijia Lin and Ji Luo. Succinct and adaptively secure ABE for ABP from k-lin. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 437–466. Springer, Heidelberg, December 2020.

[LLL22] Hanjun Li, Huijia Lin, and Ji Luo. ABE for circuits with constant-size secret keys and adaptive security. Cryptology ePrint Archive, Report 2022/659, 2022. https://eprint.iacr.org/2022/659.

[LM16] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 443–468. Springer, Heidelberg, October / November 2016.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav

Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 630–660. Springer, Heidelberg, August 2017.

[Luo22] Ji Luo. Ad hoc (decentralized) broadcast, trace, and revoke. Cryptology ePrint Archive, Report 2022/925, 2022. https://eprint.iacr.org/2022/925.

[LZ17] Qipeng Liu and Mark Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 138–169. Springer, Heidelberg, November 2017.

[NY15] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 565–584. Springer, Heidelberg, August 2015.

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

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[Tak14] Katsuyuki Takashima. Expressive attribute-based encryption with constant-size ciphertexts from the decisional linear assumption. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 298–317. Springer, Heidelberg, September 2014.

[YAHK14] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. A framework and compact constructions for non-monotonic attribute-based encryption. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 275–292. Springer, Heidelberg, March 2014.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[ZGT+16] Kai Zhang, Junqing Gong, Shaohua Tang, Jie Chen, Xiangxue Li, Haifeng Qian, and Zhenfu Cao. Practical and efficient attribute-based encryption with constant-size ciphertexts in outsourced verifiable computation. In Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang, editors, *ASIACCS 16*, pages 269–279. ACM Press, May / June 2016.