


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

Aayush Jain¹ Huijia Lin² 罗辑 (Ji Luo)² 

¹ Carnegie Mellon University, USA
aayushja@andrew.cmu.edu

² Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, USA
{rachel, luojij}@cs.washington.edu

1 November 2023

Abstract

We investigate the optimal (asymptotic) efficiency of functional encryption (FE) and attribute-based encryption (ABE) by proving inherent space-time trade-offs and constructing nearly optimal schemes. We consider the general notion of partially hiding functional encryption (PHFE), capturing both FE and ABE, and the most efficient computation model of random-access machines (RAM). In PHFE, a secret key sk_f is associated with a function f , whereas a ciphertext $ct_x(y)$ is tied to a public input x and encrypts a private input y . Decryption reveals $f(x, y)$ and nothing else about y .

We present the first PHFE for RAM solely based on the necessary assumption of FE for circuits. Significantly improving upon the efficiency of prior schemes, our construction achieves nearly optimal succinctness and computation time:

- Its secret key sk_f is of *constant size* (optimal), independent of the function description length $|f|$, i.e., $|sk_f| = \text{poly}(\lambda)$.
- Its ciphertext $ct_x(y)$ is *rate-2* in the private input length $|y|$ (nearly optimal) and *independent* of the public input length $|x|$ (optimal), i.e., $|ct_x(y)| = 2|y| + \text{poly}(\lambda)$.
- Decryption time is *linear* in the *instance* RAM running time T , plus the function and public/private input lengths, i.e., $T_{\text{Dec}} = (T + |f| + |x| + |y|) \text{poly}(\lambda)$.

As a corollary, we obtain the first ABE with both keys and ciphertexts being constant-size, while enjoying the best-possible decryption time matching the lower bound by Luo [ePrint '22]. We also separately achieve several other PHFE and ABE schemes.

We study the barriers to further efficiency improvements. We prove the first unconditional space-time trade-offs for (PH-)FE:

- No secure (PH-)FE can have $|sk_f|$ and T_{Dec} both sublinear in $|f|$.
- No secure PHFE can have $|ct_x(y)|$ and T_{Dec} both sublinear in $|x|$.

Our lower bounds apply even to the weakest secret-key 1-key 1-ciphertext selective schemes. Furthermore, we demonstrate a conditional barrier towards the optimal decryption time $T_{\text{Dec}} = T \text{poly}(\lambda)$ while keeping linear size dependency — any such (PH-)FE scheme implies doubly efficient private information retrieval (DE-PIR) with ideal efficiency, for which so far there is no satisfactory candidate.

This is the full version (a major revision) of [JLL23] (in the proceedings of Eurocrypt 2023, published by Springer, © IACR 2023, DOI [10.1007/978-3-031-30620-4_16](https://doi.org/10.1007/978-3-031-30620-4_16)).

Contents

1	Introduction	1
1.1	Our Results	2
1.2	What’s Next?	6
1.3	Technical Overview	7
1.4	Related Works	13
2	Preliminaries	16
2.1	Multi-Tape Random-Access Machine	16
2.2	Laconic Garbled RAM	19
2.3	Partially Hiding Functional Encryption and FE for Circuits	22
2.4	Universal RAM and PHFE for RAM	24
2.5	Indistinguishability Obfuscation	25
2.6	Laconic Oblivious Transfer	26
2.7	Garbled Circuits	28
2.8	Puncturable Pseudorandom Function	28
2.9	Secret-Key Encryption	29
2.10	Oblivious RAM	29
2.11	Primitives Related to Lower Bounds	32
3	Efficiency Trade-Offs of PHFE for RAM	33
3.1	Contention Between Storage Overhead and Decryption Time	34
3.2	Barrier to Fast Decryption	38
4	Bounded LGRAM with Fixed-Memory Security	41
4.1	Construction	41
4.2	Security	45
5	Transformations of LGRAM	54
5.1	Fixed-Memory to Fixed-Address	54
5.2	Fixed-Address to Full Security	58
5.3	Bounded to Unbounded	61
6	PHFE for RAM	64
6.1	Bounded Private Input	64
6.2	Full-Fledged PHFE for RAM	73
7	Applications	76
7.1	Rate-1 PHFE for RAM	76
7.2	ABE for RAM	76
7.3	Constant-Overhead $i\mathcal{O}$ for RAM	77
	References	78
	Appendix	
A	Generically Lifting DE-PIR Security	85
B	Efficiency, Security, and $\text{poly}(\lambda)$ Factors	88

1 Introduction

Functional encryption (FE) [BSW11,O’N10] and attribute-based encryption (ABE) [SW05, GPSW06] are powerful enhancement of public-key encryption with many fascinating applications. In this work, we investigate their best-possible efficiency, proving inherent space-time trade-offs for FE and presenting nearly optimal constructions of FE and ABE.

To this end, we consider partially hiding functional encryption (PHFE) [GVW15, AJL⁺19, JLMS19], a general notion capturing both FE and ABE. In PHFE, a secret key sk_f is associated with a function f , whereas a ciphertext $ct_x(y)$ is tied to a public input x and encrypts a private input y . Their decryption recovers the computation output $f(x, y)$. Collusion-resistant (indistinguishability-based) security ensures that given unboundedly (polynomially) many secret keys $\{sk_{f_q}\}_q$ for multiple functions $\{f_q\}_q$, ciphertexts $ct_x(y_0)$ and $ct_x(y_1)$ tied to the same public input x and encrypting different private inputs y_0, y_1 remain indistinguishable so long as none of the keys separate them by functionality, i.e., $f_q(x, y_0) = f_q(x, y_1)$ for all q . Put simply, the only information revealed about the private input y is the outputs $\{f_q(x, y)\}_q$.

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

*What is the best-possible asymptotic efficiency of PHFE?
Are there trade-offs among different aspects of efficiency?
Can we construct optimally efficient PHFE?*

We make progress towards answering the above questions.

For the lower bounds, we prove inherent trade-offs between the size of keys or ciphertexts and the decryption time, and show barriers towards achieving the optimal decryption time.

On the constructive front, we present the first collusion-resistant PHFE for RAM solely 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 nearly optimal 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]. By slightly tweaking the construction, we also obtain ABE with linear-size keys and/or ciphertexts and the optimal decryption time subject to the known lower bound.

Dream Efficiency. Before describing our results, we first picture the dream efficiency with respect to three important dimensions. Each dimension has been a consistent research theme across many primitives in cryptography. These dimensions are not disparate but closely related.

Efficient Computation Model. Using a realistic and efficient model helps accurately characterizing efficiency. For example, the time of computation in the clear serves as a baseline for decryption time. For this reason, functions should be represented by random-access machines (RAM), the most efficient computation model subsuming both circuits and Turing machines. It is also closer to real-world computers.

We consider a RAM U (fixed¹ at set-up time) with random access to three tapes, a function tape containing f , an input tape containing $x||y$, and a working tape. It may produce *arbitrarily long* output, e.g., one bit at *every* step. This flexible model captures many natural scenarios, e.g., binary search where the database could be part of f, x, y . It can emulate the evaluation of a circuit C on input (x, y) by putting the circuit description on the function tape. In ciphertext-policy ABE, each ciphertext is tied to a predicate P , which can be captured by setting $x = P$. These examples tell us that any or even all of f, x, y could be long, and we want to optimize the efficiency dependency on their lengths.

Succinctness. Enjoying low communication and storage overhead means having short master public key mpk , secret keys sk_f , and ciphertexts $\text{ct}_x(y)$. At the most basic level, the size of each component should be *polynomial* in the length of the information it is associated with — $|\text{mpk}| = O(1)$,² $|\text{sk}_f| = \text{poly}(|f|)$, and $|\text{ct}_x(y)| = \text{poly}(|x|, |y|)$, where $|f|, |x|, |y|$ are the description lengths of f, x, y , respectively — referred to as *polynomial efficiency*.³ However, there is much to be desired beyond this basic level of efficiency. For instance, linear efficiency means $|\text{sk}_f| = O(|f|)$ and $|\text{ct}| = O(|x| + |y|)$, and rate-1 efficiency *could* mean $|\text{sk}_f| = |f| + O(1)$ and $|\text{ct}| = |x| + |y| + O(1)$.

In fact, even smaller parameters are possible. Since (PH-)FE does not aim to hide the function f , it is allowed to put the 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 aim for secret keys of size *independent* of that of the function — i.e., $|\text{sk}_f| = O(1)$ — 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 private input length $|y|$ — i.e., $|\text{ct}_x(y)| = |y| + O(1)$. In summary:

$$\underline{\text{IDEAL COMPONENT SIZES.}} \quad |\text{mpk}| = O(1), \quad |\text{sk}_f| = O(1), \quad |\text{ct}_x(y)| = |y| + O(1).$$

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

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

$$\underline{\text{IDEAL DECRYPTION TIME.}} \quad T_{\text{Dec}} = O(T).$$

1.1 Our Results

Is the dream efficiency attainable simultaneously in all of the three dimensions? Towards understanding this, we present both new constructions and lower bounds.

¹We can think of U as a universal RAM.

²In this introduction, $O(\cdot)$ hides a multiplicative factor of $\text{poly}(\lambda)$.

³It may appear that polynomial efficiency is the bare minimum. However, it is possible to consider components whose sizes depend on an upper bound of the length of some information *not* tied to them. Many early schemes are as such, e.g., the FE scheme of [GGH⁺13] has $|\text{mpk}| = O(\text{poly}(\max |y|))$, and the celebrated ABE scheme by [BGG⁺14] has $|\text{mpk}|, |\text{ct}|$ growing polynomially with the maximum depth of the computation. When a scheme requires fixing an upper bound on parameter Z (e.g., input length, depth, or size), it is said to be Z -bounded.

New PHFE for RAM with (Nearly) Optimal Succinctness. Starting from polynomially secure bounded FE for circuits, i.e., all of $|\text{mpk}|, |\text{sk}_f|, |\text{ct}(y)|$ are just $\text{poly}(|f|, |y|)$, which in turn can be constructed from well-studied assumptions [JLS21, JLS22], we construct an adaptively secure (unbounded) PHFE for RAM with nearly optimal succinctness.

Theorem 25. *Assuming polynomially secure FE for circuits, there exists an adaptively secure PHFE for RAM with*

$$|\text{mpk}| = O(1), \quad |\text{sk}_f| = O(1), \quad |\text{ct}_x(y)| = 2|y| + O(1), \quad T_{\text{Dec}} = O(T + |f| + |x| + |y|).$$

Our construction gives the first collusion-resistant (PH-)FE for RAM, and also the first (PH-)FE for any model of computation (e.g., circuit or TM) with nearly optimal succinctness. The only gap to optimality is that the ciphertext is rate-2 in $|y|$ instead of rate-1. We can tweak our construction for *the* optimal succinctness, at the cost of both adaptive security and long output:

Corollary 26. *Assuming polynomially secure FE for circuits, there exists a semi-adaptively secure PHFE for decisional (1-bit output) RAM with*

$$|\text{mpk}| = O(1), \quad |\text{sk}_f| = O(1), \quad |\text{ct}_x(y)| = |y| + O(1), \quad T_{\text{Dec}} = O(T + |f| + |x| + |y|).$$

Prior schemes work with either circuits [GGH⁺13, JLS21, JLS22] or Turing machines [AS16, AM18, KNTY19], except for the recent concurrent and independent work of [ACFQ22], which also constructs FE for RAM. All of them only achieve polynomial efficiency as summarized in Table 1. We further discuss related works in Section 1.4.

As a corollary, we obtain the first ABE for RAM from falsifiable assumptions, and the first 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 obfuscation. In terms of succinctness, existing 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 (see discussion in [LLL22]). The state-of-the-art is summarized in Table 2.

Table 1. Comparison among some (PH-)FE schemes. All rows except this work are FE, and this work is PHFE. C is the circuit. T is the instance running time of TM/RAM. All $\text{poly}(\cdot)$ and $O(\cdot)$ implicitly contains λ . For assumptions, “ $i\mathcal{O}$ ” means indistinguishability obfuscation, FE is always for circuits, “sls” means sublinearly succinct, “subexp” means subexponentially secure, and “PK-DE-PIR” means public-key doubly efficient private information retrieval.

reference	functionality	$ \text{sk} $	$ \text{ct} $	T_{Dec}	adaptive	assumption
[GGH ⁺ 13]	circuit	$\text{poly}(C)$	$\text{poly}(y)$	$\text{poly}(C)$		$i\mathcal{O}$
[KNTY19]	circuit	$\text{poly}(C)$	$\text{poly}(y)$	$\text{poly}(C)$	✓	1-key sls FE
[GWZ22]	circuit	$\text{poly}(C)$	$ y + O(1)$	$\text{poly}(C)$		$i\mathcal{O}$
[AS16]	TM	$\text{poly}(f)$	$\text{poly}(y)$	$T \text{poly}(f , y)$	✓	$i\mathcal{O}$
[AJS17]	TM	$c f + O(1)$	$c y + O(1)$	$T \text{poly}(f , y)$	✓	subexp $i\mathcal{O}$
[AM18]	TM	$\text{poly}(f)$	$O(y)$	$T \text{poly}(f , y)$	✓	FE
[KNTY19]	TM	$\text{poly}(f)$	$\text{poly}(y)$	$T \text{poly}(f , y)$		1-key sls FE
[ACFQ22]	RAM	$\text{poly}(f)$	$\text{poly}(y)$	$T \text{poly}(f)$		PK-DE-PIR & FE
Theorem 25	RAM	$O(1)$	$2 y + O(1)$	$O(T + f + x + y)$	✓	FE
Corollary 26	RAM (1-bit output)	$O(1)$	$ y + O(1)$	$O(T + f + x + y)$		FE

Table 2. Comparison among some KP-ABE schemes. Notations shared with Table 1. ABP means arithmetic branching programs (also denoted by C). For assumptions, “ k -Lin” means k -Linear in pairing groups, “LWE” means learning with errors, “GGM” means generic pairing group model, “SNARK” means succinct non-interactive argument of knowledge, and “ $di\mathcal{O}$ ” means differing-input obfuscation.

reference	functionality	$ \text{sk} $	$ \text{ct} $	T_{Dec}	adaptive	assumption
[LL20]	ABP	$O(C \cdot x)$	$O(1)$	$O(C \cdot x)$	✓	k -Lin
[BGG ⁺ 14]	circuit	$\text{poly}(d)$	$ x \text{poly}(d)$	$ C \text{poly}(d)$		LWE
[LLL22]	circuit	$O(1)$	$\text{poly}(d)$	$ C \text{poly}(d)$	✓	GGM & LWE
[GKP ⁺ 13]	RAM	$O(1)$	$\text{poly}(x)$	$O(T + f + x)$		SNARK & $di\mathcal{O}$
Corollary 27 or 28	RAM	$O(1)$	$O(1)$	$O(T + f + x)$	✓	FE
Corollary 28	RAM	$ f + O(1)$	$O(1)$	$O(T + x)$	✓	FE
Corollary 28	RAM	$O(1)$	$ x + O(1)$	$O(T + f)$	✓	FE
Corollary 28	RAM	$ f + O(1)$	$ x + O(1)$	$O(T)$	✓	FE

Corollary 27. *Assuming polynomially secure FE for circuits, there exist adaptively secure key/ciphertext-policy ABE schemes for RAM with*

$$\begin{aligned}
 (\text{KP-ABE}) \quad & |\text{mpk}| = O(1), \quad |\text{sk}_f| = O(1), \quad |\text{ct}_x| = O(1), \quad T_{\text{Dec}} = O(T + |f| + |x|), \\
 (\text{CP-ABE}) \quad & |\text{mpk}| = O(1), \quad |\text{sk}_x| = O(1), \quad |\text{ct}_f| = O(1), \quad T_{\text{Dec}} = O(T + |f| + |x|).
 \end{aligned}$$

The decryption time of our PHFE and ABE *appears* suboptimal. In addition to the necessary linear dependency on T , it also grows linearly with $|f|, |x|, |y|$. It turns out that *ideal succinctness and ideal decryption time are at conflict*. We prove that for PHFE, under *sublinear* succinctness, the linear dependency of T_{Dec} on $|f|, |x|$ is *inherent*. We also show barriers towards removing the dependency of T_{Dec} on $|y|$ or $|f|, |x|$ while maintaining linear succinctness.

Our PHFE scheme matches the lower bounds and the barriers — it is Pareto-optimal with respect to the dependency on $|f|, |x|$. For ABE, our lower bounds and barriers do not apply. Nevertheless, our ABE scheme matches an existing lower bound by [Luo22], which states that any moderately expressive ABE must satisfy $|\text{ct}_x| \cdot T_{\text{Dec}} = \Omega(|x|)$ and $|\text{sk}_f| \cdot T_{\text{Dec}} = \Omega(|f|)$.⁴ Given that our scheme has constant-size keys and ciphertexts, its decryption time matches the lower bound of [Luo22], hence it is thus Pareto-optimal. By tweaking the construction, we obtain several other Pareto-optimal ABE schemes:

Corollary 28. *Assuming polynomially secure FE for circuits, there exist adaptively secure KP-/CP-ABE schemes for RAM with*

$$\begin{aligned}
 (\text{KP-ABE}) \quad & |\text{mpk}| = O(1), \quad |\text{sk}_f| = |f|^A + O(1), \quad |\text{ct}_x| = |x|^B + O(1), \\
 & T_{\text{Dec}} = O(T + |f|^{1-A} + |x|^{1-B}), \\
 (\text{CP-ABE}) \quad & |\text{mpk}| = O(1), \quad |\text{sk}_x| = |x|^B + O(1), \quad |\text{ct}_f| = |f|^A + O(1), \\
 & T_{\text{Dec}} = O(T + |f|^{1-A} + |x|^{1-B}).
 \end{aligned}$$

All of the four combinations of $(A, B) \in \{0, 1\}^2$ are possible for both KP- and CP-ABE.

⁴The lower bounds apply as long as the ABE scheme supports broadcast encryption. Theorem 14 in [Luo22] is the first trade-off between $|\text{ct}_x|$ and T_{Dec} . Essentially the same proof yields the second trade-off between $|\text{sk}_f|$ and T_{Dec} .

Contention Between Succinct Components and Fast Decryption. We now describe our lower bounds in more detail. Consider the efficiency dependency on the lengths of public information f and x . We show that unconditionally, it is impossible to simultaneously achieve key size sublinear in $|f|$ and decryption time sublinear in $|f|$. Similarly, it is impossible to have both ciphertext size and decryption time sublinear in $|x|$. In fact, these trade-offs apply to the weakest secret-key 1-key 1-ciphertext selectively secure PHFE, and the first trade-off with respect to $|f|$ also applies to plain FE.

Theorems 5 and 6. *For a secret-key 1-key 1-ciphertext selectively secure moderately expressive PHFE with*

$$\begin{array}{llll} \text{either} & |\text{sk}| = O(|f|^A) & \text{and} & T_{\text{Dec}} = (T + |f|^B + |y|) \text{poly}(|x|), \\ \text{or} & |\text{ct}| = |x|^A \text{poly}(|y|) & \text{and} & T_{\text{Dec}} = (T + |f| + |x|^B) \text{poly}(|y|), \end{array}$$

it must hold that $A \geq 1$ or $B \geq 1$. In the first case, the same (without x) is true for FE.

Our PHFE scheme achieves one profile of Pareto-optimality, $A = 0$ and $B = 1$.

A natural question is whether the other Pareto-optimal profile, $A = 1$ and $B = 0$ (or even just $B < 1$), is attainable. Another question is whether the decryption time must grow with the length of the private input y .

Barriers to Ideal Decryption Time. We illustrate barriers to positive answers to the above two questions. We show that PHFE with decryption time independent of $|f|$, $|x|$, or $|y|$ implies doubly efficient private information retrieval (DE-PIR) with small preprocessed database.

Theorems 10, 11, and 12. *Suppose a secret-key moderately expressive PHFE with selective security has*

$$\begin{array}{llll} \text{either} & |\text{sk}_f| = O(|f|^A) & \text{and} & T_{\text{Dec}} = |f|^B \text{poly}(T, |x|, |y|), \\ \text{or} & |\text{ct}_x| = |x|^A \text{poly}(|y|) & \text{and} & T_{\text{Dec}} = |x|^B \text{poly}(T, |f|, |y|), \\ \text{or} & |\text{ct}_x| = |y|^A \text{poly}(|x|) & \text{and} & T_{\text{Dec}} = |y|^B \text{poly}(T, |f|, |x|), \end{array}$$

for constants A and $0 \leq B < 1$, then there exists a secret-key DE-PIR with

$$|\tilde{D}| = |D| + O(|D|^A), \quad T_{\text{Resp}} = O(|D|^B), \quad T_{\text{Query}} = O(1), \quad T_{\text{Dec}} = O(1).$$

In the first case, the PHFE only has to be 1-key secure, and if it is public-key, then so can be the resultant DE-PIR.

DE-PIR, introduced by [BIPW17,CHR17], allows a client to privately encode a database D into \tilde{D} while keeping a public or 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, DE-PIR allows the database to be privately encoded in exchange for *double efficiency* – for each query, the complexities of both the client and the server are, ideally, independent of the database size $|D|$, whereas PIR necessarily has server complexity $\Omega(|D|)$. The double efficiency of DE-PIR makes it highly desirable. The initial works [BIPW17,CHR17] presented candidate constructions 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 (though formally unrelated to) those DE-PIR schemes has been broken [BHMW21]. Very recently, in a concurrent and independent work, Lin, Mook, and Wichs [LMW23] constructed DE-PIR with public preprocessing from the ring LWE assumption.

The most important efficiency metrics of DE-PIR are the preprocessed database size and the complexity per query. Our theorem shows that constructing PHFE with short decryption time entails constructing DE-PIR with preprocessed database size inherited from ciphertext/key size. In particular, a PHFE scheme with decryption time independent of $|y|$ and ciphertext size linear in $|y|$ implies a DE-PIR with preprocessed database of length $O(N)$ and constant complexity per query. Such efficiency is currently considered open. (See Sections 1.4 and B.)

Succinct Garbled RAM and Constant-Overhead $i\mathcal{O}$. The main tool in our construction of PHFE for RAM is succinct garbled RAM (GRAM). Initiated by [KLW15, BGL⁺15, CHJV15], a sequence of works have constructed succinct garbled RAM [CH16, CCC⁺16, ACC⁺16, CCHR16] based on subexponentially secure FE for circuits and succinct garbled Turing machines [KLW15, AJS17, AL18, GS18b] based on polynomially secure FE for circuits.

In this work, we formulate a new notion of succinct GRAM geared for building highly efficient PHFE for RAM, and construct it based on polynomially secure FE for circuits. Our construction has two consequences: *i*) we obtain the first succinct GRAM (our or the standard notion) based on polynomial hardness, as opposed to subexponential hardness as in prior constructions, and *ii*) using $i\mathcal{O}$ for circuits, we obtain $i\mathcal{O}$ for RAM with constant overhead — the size of the obfuscated program is $2|M| + \text{poly}(N)$, where M is the RAM to be obfuscated and N is the input length. Previously, constant-overhead $i\mathcal{O}$ was only known for circuits [BV15] and Turing machines [AJS17].

1.2 What’s Next?

Recent developments of PHFE for RAM leave several interesting questions open for future research.

Optimal Decryption Time and Ideally Efficient DE-PIR. We are yet to construct PHFE with optimal decryption time from ideally efficiency DE-PIR. The work of [ACFQ22] provides a partial answer, but a more careful construction is required per definitions in this work. Constructing ideally efficiency DE-PIR is also in itself a goal.

Rate, Security, and Functionality. Our PHFE achieves either rate-2 with adaptive security and long output, or rate-1 with semi-adaptive security and short output. (The rate-1 scheme of [GWZ22] is also semi-adaptively secure and for short output.) The factor of two in our scheme arises from the classic technique of double encryption [NY90, BS18]. Achieving rate-1 with *either* adaptive security *or* long output (or even both) seems to require novel techniques.

Tight Relation Between Optimal Decryption Time and DE-PIR Notion. From [ACFQ22], we know that FE for circuits plus *public-key* DE-PIR implies FE for RAM with y -independent decryption time, yet both [ACFQ22] and our work are only able to show such FE for RAM implies *secret-key* DE-PIR. A natural question is what the tight relation between those notions is, or whether gap between public-/secret-key can be closed. Can

we construct such FE for RAM from secret-key DE-PIR? Does such FE for RAM imply public-key DE-PIR?

Exact Pareto Frontier of PHFE and ABE Efficiency. The efficiency trade-offs characterized by [Luo22] and this work are depicted in Figure 1. For PHFE, studying DE-PIR and solving the first open question could completely resolve the unknown area related to DE-PIR. For ABE, we know neither lower bounds⁵ nor constructions in the unknown area, and it remains to pin down the exact Pareto frontier of its efficiency.

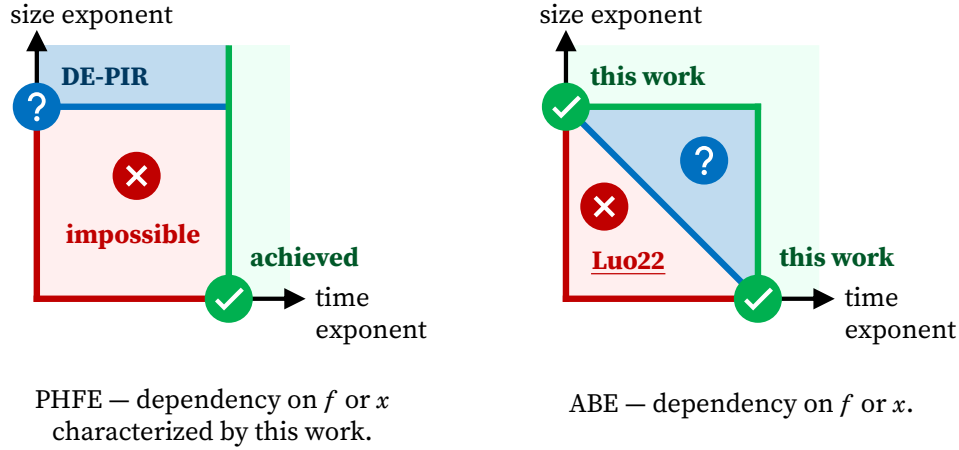


Figure 1. Currently known trade-offs of PHFE and ABE efficiency.

1.3 Technical Overview

We start with an overview of our negative results.

Unconditional Lower Bounds. As introduced earlier, we show that it is impossible for a secure PHFE to enjoy sublinear dependency on $|f|$ [resp. $|x|$] simultaneously for $|\text{sk}_f|$ [resp. $|\text{ct}_x(y)|$] 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

$$|\text{sk}_f| = O(|f|^A) \quad \text{and} \quad T_{\text{Dec}} = O(T + |f|^B + |x| + |y|) \quad \text{for } A < 1 \text{ and } B < 1$$

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 indices and w is a one-time pad.

⁵The existing lower bound [Luo22] only uses broadcast functionality, a rather simple ABE. It is well possible that a stronger lower bound applies to ABE for RAM.

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,

$$\begin{aligned} |f| &= O(N), & |x| &= O(1), & |y| &= O(n), & T &= O(n), \\ |\text{sk}| &= O(N^A), & & & & & T_{\text{Dec}} &= O(n + N^B). \end{aligned}$$

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

The adversary chooses

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

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 them with random access to the function description R in the clear. It notes down the set $L \subseteq [N]$ of indices in R that are read during decryption, i.e.,

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

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 accessed when decrypting ct encrypting y_b .

- When ct encrypts y_0 , the decryption algorithm can be used to recover $R[I]$ (as the adversary knows w). It can only obtain information of $R[I]$ from sk and R . Since $R[I]$ contains n bits of entropy, by setting $|\text{sk}| = O(N^A) \ll n$, decryption must read a *large* portion of $R[I]$, implying that $|L \cap I|$ is *large*, namely, $\Omega(n)$.
- In contrast, when ct encrypts y_1 , observe that the joint distribution of $(R, \text{ct}, \text{sk})$ is independent of I (w is a one-time pad and completely hides I in $y_1 = R[I] \oplus w$). Therefore, the behavior of Dec is independent of I . By tuning the parameters, we make Dec run in time $O(n + N^B) \ll N$, so short that with I unknown, the locations Dec reads in R only have a small overlap with I , i.e., $|L \cap I|$ is likely to be *small*.

It remains to analyze how large and small $|L \cap I|$ is in the above two cases. Let's first consider a toy proof, under the hypothesis that sk contains no information about $R[I]$ at all. We will remove this hypothesis later. When ct encrypts y_0 , the decryption algorithm must read the entire $R[I]$ from R itself, i.e., $I \subseteq L_0$, so $|L_0 \cap I| = |I| = n$. When

ct encrypts y_1 , since the indices L_1 read from R are independent of I , the intersection size $|L_1 \cap I|$ follows a hypergeometric distribution and

$$\mathbb{E}[|L_1 \cap I|] \leq \frac{T_{\text{Dec}} \cdot n}{N} \ll n.$$

This means the adversary can distinguish when ct encrypts y_0 or y_1 with good advantage, 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 decryption algorithm may obtain some information of $R[I]$ from sk . Our intuition is $|L_0 \cap I| \geq |I| - |\text{sk}|$, but proving this formally is not trivial as sk could depend arbitrarily on $R[I]$.

We employ a compression argument. The basic idea behind a compression argument is that no pair of encoding and decoding algorithms (E, D) , sharing arbitrarily long randomness s , can transmit an n -bit random string u (independent of s) from one end to the other via an encoding v of less than n bits, i.e.,

$$\Pr \left[\begin{array}{l} s \xleftarrow{\$} \mathcal{S} \\ u \xleftarrow{\$} \{0, 1\}^n : D(s, v) = u \\ v \leftarrow E(s, u) \end{array} \right] = 1 \quad \implies \quad |v| \geq |u|.$$

We show that if $|L_0 \cap I| < |I| - |\text{sk}|$, then there would exist (E, D) violating the above information-theoretic bound:

- The shared randomness s consists of PHFE randomness and $I, w, R[[N] \setminus I]$.
- To encode $u \in \{0, 1\}^n$, the procedure E first fills $R[I] = u$. Using s , it generates a PHFE key sk for R and a ciphertext ct encrypting $y_0 = (I, w)$, and then runs Dec to obtain the indices L_0 read into R . The codeword is $v = (\text{sk}, R[L_0 \cap I])$.
- To decode, D regenerates ct using s , runs Dec to obtain $z = R[I] \oplus w$, and recovers $u = z \oplus w$. Note that every query by Dec into R is in either $R[[N] \setminus I]$ (found in s) or $R[L_0 \cap I]$ (found in v).

Suppose $|L_0 \cap I| < |I| - |\text{sk}|$, then $|v| = |\text{sk}| + |L_0 \cap I|$ would be less than $|u| = |I|$, contradicting the incompressibility of u .⁶

The toy analysis of ct encrypting y_1 holds, for which $|L_1 \cap I| \leq n/2$ with high probability by Markov's inequality. To make ciphertexts of y_0 and y_1 easily distinguishable, we can set,⁷ e.g., $n = N^{(A+1)/2}$, so that $|\text{sk}| = O(N^A) \ll n$ hence $|L_0 \cap I| \geq |I| - |\text{sk}| \geq 2n/3$. In summary, any PHFE scheme with both $|\text{sk}|$ and T_{Dec} sublinear in $|f|$ (and linear in $T, |x|, |y|$) must be insecure.

⁶We have implicitly assumed that $|L_0 \cap I|$ has a fixed size. The formal proof handles potentially random $|L_0 \cap I|$ by truncation. It suffers from incorrect decoding hence the statements are probabilistic.

⁷In the formal proof, N itself is a large $\text{poly}(\lambda)$ to overwhelm the hidden $\text{poly}(\lambda)$ factors in the efficiency parameters, which are ignored in this overview.

Technical Barrier Towards Fast Decryption. We demonstrate barriers in current techniques against constructing a PHFE scheme with fast decryption. Consider a PHFE scheme whose decryption time is

$$|f|^B \text{poly}(T, |x|, |y|) \quad \text{or} \quad |x|^B \text{poly}(T, |f|, |y|) \quad \text{or} \quad |y|^B \text{poly}(T, |f|, |x|)$$

for constant $B < 1$, with linear-size components. We show that any of those schemes implies doubly efficient private information retrieval (DE-PIR).

To illustrate our main idea, we present this transformation for the case when decryption takes time $\text{poly}(T, |x|, |y|)$, independent of $|f|$. In this case, we obtain *public-key* DE-PIR. The ideas naturally extend to the other cases, but the resultant DE-PIR schemes are only secret-key.

Since decryption is efficient in $|f|$, it is natural to set f as the database $D \in \{0, 1\}^B$. For DE-PIR security, we set y , the only component with privacy in PHFE, to be the index $i \in [N]$ being queried. The client processes D by sampling (mpk, msk) of the PHFE and sending

$$\tilde{D} = (D, \text{sk}), \text{ where } \text{sk} \stackrel{\$}{\leftarrow} \text{KeyGen}(\text{mpk}, D)$$

to the server. It keeps mpk locally. To look up $D[i]$, the client sends a fresh PHFE ciphertext

$$\text{ct} \stackrel{\$}{\leftarrow} \text{Enc}(\text{mpk}, \overbrace{\perp, i}^{x,y})$$

to the server, which responds by running $D[i] \leftarrow \text{Dec}^{D, \perp, \text{sk}, \text{ct}}(\text{mpk})$. Note that double efficiency is already satisfied. The client prepares a query in time polynomial in the bit-length of y , i.e., $\text{poly}(\log N)$. Due to the supposed efficiency of decryption, the server responds within time $\text{poly}(T, |x|, |y|) = \text{poly}(\log N)$.

While this idea solves the core issue, we have missed an important aspect. The scheme does not fully hide the queried indices, only so when $D[i_0] = D[i_1]$. Fortunately, full hiding of DE-PIR can be obtained via generic transformations. When processing $D \in \{0, 1\}^N$, we let D' be D concatenated with its bitwise negation, and process D' using the weak scheme. Each time some $D[i]$ needs to be looked up, the client randomly issues a query (in the weak scheme) to either $D'[i]$ or $D'[N+i]$ and notes down its choice. When the server responds, the client obtains the correct result from the remembered choice. The transformed DE-PIR fully hides the query indices while preserving the efficiency of the underlying scheme.

Overview of Our PHFE for RAM. At a very high level, we use a *succinct* garbled RAM (GRAM) scheme to lift an FE for circuits to a PHFE for RAM. The former can be viewed as a 1-key, 1-ciphertext, secret-key FE for RAM, where succinctness means that the running time of key generation and encryption is independent of the running time of the RAM computation. A (collusion-resistant) FE for circuits then lifts one-time security to many-time. This high-level approach first appeared in [AS16] for building FE for TM. In this work, towards nearly optimally efficient FE for RAM, we first observe that existing definitions and constructions of succinct GRAM [BGL⁺15, CHJV15, CH16, CCC⁺16, ACC⁺16, CCHR16] are insufficient. Therefore, we formulate a new variant of succinct GRAM, termed *laconic GRAM*, then construct it. Along the way, we also weaken the assumption underlying succinct/laconic GRAM schemes from $i\mathcal{O}$, which is considered an inherently subexponential assumption, to polynomially secure FE for circuits.

Let's first review the syntax and security of standard GRAM schemes. They consist of the following algorithms. The encoding algorithm encodes a database D into \widehat{D} and outputs a private state k . The garbling algorithm uses k to garble a RAM M into \widehat{M} and outputs a collection of input labels $\{L_{i,b}\}_{i,b}$. The evaluation algorithm, given the garbled RAM \widehat{M} , one set of labels corresponding to an input w , and random access to \widehat{D} , returns the output $M^D(w)$ of the RAM computation. Simulation-based security ensures that $\widehat{D}, \widehat{M}, \{L_{i,w[i]}\}_i$ can be simulated using only the output $M^D(w)$. The efficiency requirements of those algorithms are

$$\begin{aligned} (\widehat{D}, k) &\stackrel{\$}{\leftarrow} \text{Encode}(D), & (\widehat{M}, \{L_{i,b}\}_{i,b}) &\stackrel{\$}{\leftarrow} \text{Garble}(M, k), & M^D(w) &\leftarrow \text{Eval}^{\widehat{D}}(\widehat{M}, \{L_{i,w[i]}\}_i), \\ |\widehat{D}| &= O(|D|), & |\widehat{M}| &= \text{poly}(|M|), & T_{\text{Eval}} &= T \text{ poly}(|M|). \end{aligned}$$

Unfortunately, the standard notion falls short for our purpose of building highly efficient FE for RAM. We elaborate and explain how to address those issues.

- multi-tape instead of single-tape. To begin, we consider RAM computation with multiple tapes, $M^{f,x,y}()$, instead of $M^D(w)$ with a single tape. GRAM schemes directly constructed often have (inevitably bad) polynomial efficiency dependency on $|M|$, which is inherited by the PHFE. By making M a universal machine and letting f, x, y be the tape contents, it is easier to characterize and achieve the desired efficiency.
- public tape instead of private tape. The tape contents D or parts of them (such as f, x) are public. The standard GRAM is defined only for private tapes, making the encoding size at least $|D|$. In PHFE, we provide verbatim copies of f, x for free and only count the *overhead* in sk, ct, so we must not use standard GRAM when aiming for $|\text{sk}|, |\text{ct}|$ independent of $|f|, |x|$.
- compressing instead of encoding. Our notion requires a compression algorithm that hashes public tapes down to short digests, and the garbling algorithm “binds” the hashes to the garbled program. Lastly, the evaluation algorithm gets random access to the tape contents in the clear, like PHFE decryption. Our GRAM only handles public tapes and the hiding of y is deferred to the construction of PHFE, where we achieve rate-2 thanks to the digests being short.
- reusable digests instead of one-time encoding. The encoding \widehat{D} in standard GRAM can only be used once for a single garbled program generated using the secret k .⁸ When lifted to multi-time security, each decryption of PHFE for RAM requires generating a fresh \widehat{D} inside the FE for circuits, inheriting its bad efficiency. To get rid of it, we require the digests be reusable, so that decryption does not have to recompute them.
- relaxed evaluation time. Reusability comes at a cost. In our notion, the evaluation time is $(T + |f| + |x| + |y|) \text{ poly}(|M|)$, whereas in the standard GRAM it is independent of the tape lengths $|f|, |x|, |y|$. Nevertheless, our negative results imply that the dependencies are either impossible (subject to the laconic requirement) or difficult (due to DE-PIR) to get around.

⁸We remark that \widehat{D} is often sequentially reusable, meaning that it can be updated by one execution, from where the next execution resumes. However, in FE, we need \widehat{D} to be parallel reusable because multiple decryptions start from the same tape contents and can be done in no particular order.

- *long outputs instead of single-bit output.* The standard GRAM notions deal with RAM with single-bit (or fixed-length) output, and longer outputs are handled by running multiple instances, one for each bit. For simulation security, the garbling size grows at least linearly with the output length, not to mention that the output length could be the running time, with no *a priori* polynomial upper bound. We consider garbling with arbitrarily long outputs without compromising its efficiency, for which we consider only indistinguishability-based security.

Putting the above pieces together, we formulate our laconic GRAM as in Figure 2.

- $\text{Compress}(\tau, D_\tau)$ compresses the τ^{th} public tape D_τ into a short hash digest_τ of constant length. It runs in time $O(|D_\tau|)$.
- $\text{Garble}(M, \{\text{digest}_\tau\}_\tau)$ outputs a garbled program \widehat{M} bound to the public tapes via their hashes, and pairs of labels $\{L_{i,b}\}_{i,b}$. It runs in time $\text{poly}(|M|)$.
- $\text{Eval}^{D_1, \dots, D_\tau}(M, \{\text{digest}_\tau\}_\tau, \widehat{M}, \{L_{i,w[i]}\}_i)$ returns the (arbitrarily long) output of RAM computation $M^{D_1, \dots, D_\tau}(w)$. It runs in time $O(T + \sum_\tau |D_\tau|) \text{poly}(|M|)$.
- Security guarantees that if $M^{D_1, \dots, D_\tau}(w_0)$ and $M^{D_1, \dots, D_\tau}(w_1)$ have identical outputs and running time, the distributions of $(\widehat{M}, \{\text{digest}_\tau\}_\tau, \{L_{i,w_0[i]}\}_i)$ and $(\widehat{M}, \{\text{digest}_\tau\}_\tau, \{L_{i,w_1[i]}\}_i)$ are indistinguishable. This holds when the tape contents $\{D_\tau\}_\tau$ are chosen adaptively, dependent on the hashes of previously chosen tape contents, before the program M and inputs w_0, w_1 are chosen.

Figure 2. An overview of the notion of laconic GRAM.

Our Construction of Laconic GRAM. One approach towards constructing laconic GRAM for RAM is to first obtain a non-succinct GRAM for RAM (with garbling size proportional to the worst-case running time) with tape compression from laconic OT techniques, then further compress the GRAM. First introduced in [BGL⁺15], the second step uses $i\mathcal{O}$ to obfuscate a circuit that on input an index t outputs the t^{th} component of the non-succinct GRAM. If each component can be locally generated using a small circuit of size $\text{poly}(|M|)$, then the obfuscated circuit is also of size $\text{poly}(|M|)$ and can be used as the succinct garbled program. To prove security, [BGL⁺15] identified that the non-succinct garbling scheme must satisfy another property, articulated later by [AL18], called *local simulation*. Informally, the non-succinct garbling must be proven secure via a sequence of hybrids where the components of every hybrid garbled program can be locally generated using a small circuit, and in neighboring hybrids, only a few components change. Beyond succinct garbling, local simulation has also found applications in achieving adaptive security [BHR12] of garbling schemes. A sequence of works developed local simulation strategies for garbled circuits [HJO⁺16, GS18a], Turing machines [GS18b, AL18], and RAM [GOS18]. Most notably, the work of [GS18a] took advantage of a clever pebbling technique.

Our construction of laconic GRAM proceeds in multiple steps. First, we use the techniques of [GS18a] to obtain a non-succinct GRAM with local simulation proof for a weak 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. Next, by the same approach of [BGL⁺15, GS18b, AL18], we turn it into a succinct one, still with only *fixed-memory security*, relying on $i\mathcal{O}$ for polynomial-size domain, which is implied by polynomially secure FE for circuits.

Many details need to be ironed out in order to implement our notion of laconic GRAM. For example, prior works [GS18a, GS18b, AL18] deal with single-bit output and the intermediate hybrids have the output hardwired into the garbled program. In contrast, we aim for RAM with arbitrarily long outputs. Hardwiring the long output would compromise the local simulation property since the hybrid garbled program can no longer be locally generated. To avoid this, we build a hybrid GRAM running with one input k_0 in the prefix of the computation and with the other input k_1 in the suffix. This ensures that the output can always be correctly computed while maintaining local simulation. Similar techniques appeared in [GOS18] for different reasons.

Lastly, we lift *fixed-memory security* to full security using punctured PRF and ORAM to protect memory content and access pattern, respectively. One issue in hiding the access pattern is that in our laconic GRAM, the public input tapes D_1, \dots, D_τ are not encoded using ORAM prior to garbling and evaluation. Yet, to ensure security, evaluation must access them in an oblivious way, independent of the input w_0 or w_1 . To solve this issue, we use a modified RAM M' that first copies D_1, \dots, D_τ into a freshly initialized, empty ORAM and then runs M , with accesses to D 's redirected to the ORAM. Now that the access pattern of M' is independent of the input, it suffices to garble it using GRAM with weaker security. The running time of M' scales linearly with the total length of all tapes $\sum_\tau |D_\tau|$, leading to such linear dependency in our laconic GRAM evaluation time. Our lower bound shows that this dependency cannot be removed. As a final point, to prove security, we must ensure that the use of ORAM does not hurt local simulation. Fortunately, the work of [CH16] provides a solution.

1.4 Related Works

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

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

FE for Turing Machines. Several works constructed FE for Turing machines with arbitrary-length inputs, first from $i\mathcal{O}$ [AS16], then from FE for circuits [AM18], and more recently from 1-key sublinearly compact FE [KNTY19]. The scheme of [AS16] relies on a 1-key 1-ciphertext secret-key FE for TM, which is essentially a succinct garbling scheme

for TM with indistinguishability-based security. They constructed it by modifying the succinct garbling for TM of [KLW15]. Later, the works of [AL18,GS18b] improved and simplified the garbling construction. The work of [KNTY19] improved the assumption to 1-key sublinearly succinct FE, and showed that their garbling scheme can be combined with [AS16] to obtain FE for TM. On the other hand, the work of [AM18] presented an alternative direct approach to FE for TM from FE for circuits without going through succinct garbling for TM. Prior constructions of FE for TM [AS16,AM18,KNTY19] put more focus on weakening the underlying assumptions, and only show polynomial efficiency. Examining their schemes, we conclude that they achieve efficiency listed in Table 1.

FE for Bounded-Input RAM. A line of research obtained *bounded-input* $i\mathcal{O}$ for Turing machines [KLW15,AJS17,GS18b] and RAM [BGL⁺15,CHJV15,CH16,CCC⁺16,ACC⁺16,CCHR16]. Plugging these $i\mathcal{O}$ schemes into [GGH⁺13] yields *bounded-input* FE for TM or RAM — existing $i\mathcal{O}$ only handles bounded-input TM or RAM in the sense that the obfuscator needs to know the maximum input length $\max |y|$ to the machine being obfuscated, and constructing $i\mathcal{O}$ for unbounded-input TM/RAM remains a major open question.⁹ Plugging them into [GGH⁺13] yields schemes where the key generation algorithm depends on the maximum input length $\max |y|$, despite that the machine f could process arbitrarily long inputs. Such FE is said to have *bounded input length*. In terms of efficiency, the secret key contains an obfuscated program of size $\text{poly}(|f|, \max |y|)$ when using the $i\mathcal{O}$ for RAM of [CHJV15,CH16], and $\text{poly}(|f|, \max |y|, S)$ with S being the space complexity of f when using the $i\mathcal{O}$ for RAM of [BGL⁺15].

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

ABE for Circuits and Turing Machines. Since FE implies ABE, the aforementioned FE schemes immediately imply ABE with the same level of efficiency. We can also tweak our construction to move the linear dependency between component size and decryption time.

The literature on ABE focuses on constructing ABE from weaker assumptions and achieving better efficiency, among others. The celebrated works of [GVW13,BGG⁺14] showed that ABE for *bounded-depth* circuits can be constructed from the learning with errors (LWE) assumption. Parameters of these schemes, however, depend polynomially on the maximum depth d of the supported circuits, namely, $|\text{mpk}| = \text{poly}(d)$, $|\text{sk}_f| = \text{poly}(d)$, $|\text{ct}_x| = |x| \text{poly}(d)$, and the decryption time is $T_{\text{dec}} = |f| \text{poly}(d)$. A recent work [LLL22] improved it to obtain constant-size keys while keeping the sizes of master public key and ciphertext intact, but at the cost of additionally relying on the generic (pairing) 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 RAM with constant-size secret keys $|\text{sk}_f| = O(1)$, yet large ciphertexts $|\text{ct}_x| = \text{poly}(|x|)$. Their scheme uses SNARK and differing-input obfuscation, which cannot be based on falsifiable assumptions. Another work [AFS19] tries to construct ABE for RAM from LWE, at the cost of

⁹The only known obfuscation for unbounded-input Turing machines [JJ22] requires polynomial-size proof of equivalence in Cook’s theory PV for security to hold.

making the master public key, secret keys, and ciphertexts all grow polynomially with the maximum running time, i.e., it is an ABE for bounded-time RAM.

In summary, we give the first ABE schemes for RAM from falsifiable assumptions, simultaneously having constant- or linear-size secret keys, constant- or linear-size ciphertexts, and the best-possible decryption times matching the known lower bound [Luo22] under the constraint of those key and ciphertext sizes, as shown in Table 2.

Concurrent and Independent Work on FE for RAM. Concurrently and independently of our work, the recent work by Ananth, Chung, Fan, and Qian [ACFQ22] also considers the question of FE for RAM. Despite an apparent overlap between the two works, there are many differences. The two works start with different motivations. Our goal is to understand the optimal succinctness and efficiency of PHFE, both constructively and from a lower-bound perspective, whereas [ACFQ22] aims to construct FE for RAM with optimal decryption time $T_{\text{dec}} = O(T)$. Consequently, the two works obtain mostly complementary results.

First, we prove unconditional trade-offs between the sizes of keys/ciphertexts and decryption time. It shows that no PHFE can have both optimal succinctness and optimal decryption time. We then construct PHFE for RAM with (nearly) optimal succinctness, while minimizing the decryption time to the best-possible, matching our lower bounds. The work of [ACFQ22], on the other hand, constructs FE for RAM with optimal decryption time. They did not attempt to simultaneously minimize the sizes of secret keys and ciphertexts.

On the common front, both works show that any (PH-)FE scheme for RAM with optimal decryption time implies DE-PIR. We regarded this as a barrier to optimal efficiency due to lack of DE-PIR schemes from well-studied assumptions, whereas in [ACFQ22], public-key DE-PIR (PK-DE-PIR) is used as a building block to implement such PHFE. As a result, their scheme relied on ideal obfuscation and a new assumption of permuted puzzles inherited from the *then*-current candidate PK-DE-PIR, whereas our storage-optimal PHFE scheme is based on FE for circuits, which is necessary and can in turn be based on well-studied assumptions. Our results about DE-PIR implication is finer, in that we obtain PK-DE-PIR from f -fast decryption. (See below for discussion about the current status of DE-PIR.)

There are two other major differences in the schemes. Our scheme handles arbitrarily long output, whereas [ACFQ22] considers single-bit output. To handle long output, they propose to generate a separate key to compute each output bit, meaning that the key size grows linearly with the output length, which could be as long as the running time in many scenarios. Moreover, our scheme achieves adaptive security, whereas [ACFQ22] only considers selective security.

In terms of techniques, both works demonstrate that the main bottleneck towards (PH-)FE for RAM is the insufficiency of existing notions of succinct GRAM — GRAM with reusable tape encoding is needed. The two works develop different techniques to achieve this. Our construction makes the garblings build fresh ORAM storage at the beginning of every evaluation and hence ORAM is never reused, whereas [ACFQ22] uses PK-DE-PIR to implement a “resettable” ORAM.

Doubly Efficient Private Information Retrieval. After the initial write-up of this work, Lin, Mook, and Wichs [LMW23] presented a novel and beautiful construction of DE-PIR scheme based on the ring LWE assumption. In their scheme, the preprocessing

of the database is a deterministic procedure that does not involve any secrets. Given a database D , the preprocessed database \tilde{D} has size $|D|^{1+\varepsilon}$, where ε can be any positive constant, while the client/server complexities are $\text{poly}(\log |D|, \lambda)$. A variant of the construction achieves a different trade-off, where the preprocessed database has size $|\tilde{D}| = |D| \cdot 2^{\tilde{O}(\sqrt{\log |D|})}$ and the client/server complexities are $2^{O(\sqrt{\log |D|})}$. While it significantly lowers the difficulty implied by our technical barrier theorems, they still provide interesting insight about the connection between PHFE efficiency and DE-PIR. Furthermore, the efficiency parameters of [LMW23] is slightly off from the ideal, linear version, yet the issue is more delicate than it appears. We defer further discussion on efficiency parameters to Section B.

2 Preliminaries

Throughout the paper, we denote the security parameter by λ , which is omitted except in definitions. Let Σ be a set and n a natural number, $\Sigma^{\leq n}$ is the set of *non-empty* strings of length at most n over the alphabet Σ . 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 $C[x_1](x_2) = C(x_1, x_2)$.

Symbols. Table 3 is a cheat sheet of select symbols used in this work.

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.¹⁰ The behavior of a RAM is described by its step circuit

$$(\ell_1, \dots, \ell_{\mathcal{T}}, w, \text{oldst}, \text{rdata}) \xrightarrow{\text{CPU}} (\text{done}, \text{newst}, \tau, i, \text{wdata}, \text{out}).$$

The step circuit takes as input *i*) the input tape lengths $\ell_1, \dots, \ell_{\mathcal{T}}$ 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 , and 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 $D_1, \dots, D_{\mathcal{T}}$ on the input tapes and w as the short input begins by zero-initializing¹¹ the working tape, the state, and the last-read string. Throughout the process, the short input stays unchanged and the state is updated by the machine. At each step, the requested location (determined by the output of the previous step) is read, whose value is passed into the last-read string of the current step, before that location is overwritten. For simplicity, we insist that overwriting happen if and only

¹⁰The distinction between input and state is arbitrary, and many works formulate them as a single entity. We choose to separate them for clearer semantics.

¹¹We do not consider persistent memory in this work.

Table 3. Cheat sheet of select symbols.

context	symbol	meaning	
general	$\lambda, \mathcal{A}, \beta', q$	security parameter, adversary, output of adversary, query index	
	β, β'	challenge bit, output of adversary	
	B, β', b	choice bit (parameter, name), choice bit (argument, value), choice bit	
	ℓ, r	length, randomness	
	J, J'	object in constructed scheme, object J in underlying scheme	
RAM	\mathcal{T}, τ, D	tape count, tape index, tape content	
	i, j	address (index into tape content / of cell), index into cell content	
	M, w, t, T	RAM, short input, time step, instance running time	
	\bar{T}	instance running time bound (for security)	
	T_{\max}	time / time and space bound (for correctness, $T \leq \bar{T} \leq T_{\max}$)	
LGRAM	M, \widehat{M}, L	program, garbled program, label	
	b, i	short input bit (choosing L), index into short input	
	t, s	hybrid index (time step), hybrid index (pebbles)	
	t^*	pebble to be changed (time step, $t^* < t$)	
	B, K, W	choice bit, PPRF key, short input (parameters, names)	
	β, k, w	choice bit, PPRF key, short input (arguments, values)	
	\widetilde{st}	state of simulated execution of underlying machine	
	\widetilde{rdata}	last read string of simulated execution of underlying machine	
	\widetilde{wdata}	overwriting string of simulated execution of underlying machine	
	$\widetilde{\ell}$	time bound for copying input tape contents	
PHFE	φ, Φ	functionality, functionality family	
	f, F, z, Z	function description, function description space, output, output space	
	x, X, y, Y	public input, public input space, private input, private input space	
	T, \bar{T}	baseline of decryption efficiency, upper bound of T (for security)	
	idx, idx'	index, encrypted index (proof progress)	
	k, s, k', q	PPRF key, PPRF input, SKE key, hybrid index (PHFE key being operated)	
	β, w, k	choice bit, double encryption, PRF key	
	iO	C, \widetilde{C} circuit, obfuscated circuit	
	LOT	D, \widehat{D}, h	database, processed database, hash
		m, b	message, cell content / hash bit (choosing m)
$\widetilde{rct}, \widetilde{wct}$		simulated rct, simulated wct	
GC	C, \widehat{C}, L	circuit, garbled circuit, label	
	b, i	input bit (choosing L), index into input	
	π, \widehat{x}	point-and-permute string, permuted input	
PPRF	$k, \mathring{k}, \mathring{k}_p$	key, punctured key, key punctured over p	
SKE	k	key	
ORAM	J'	physical object for logical object J	
	S'_{\max}	physical address bound (running space)	
	T_0, t_0	physical step count for one logical access, index	
DE-PIR	$D, \widehat{D}, \sigma, \rho$	database, processed database, query-specific secret, response	

if the requested tape is the working tape (the input tapes are read-only). The whole sequence of out's, including their timing (at which step each output bit is produced), is the output of the execution. This process is denoted by $M^{D_1, \dots, D_{\mathcal{T}}}(w)$. We now give the formal definition.

Definition 1 (multi-tape RAM). Let $\mathcal{T} \in \mathbb{N}$. A \mathcal{T} -tape RAM M is specified by its step circuit

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

which is subject to the constraints below.

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} , from 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 the following.

- $\ell_{\tau} \in [2^{\ell_{\text{addr}}}]$ 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} .
- $\text{oldst} \in \{0, 1\}^{\ell_{\text{st}}}$ is the state of length ℓ_{st} .
- $\text{rdata} \in \{0, 1\}^{\ell_{\text{CELL}}}$ 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, they are as follows.

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

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

1. Let $\text{st}_0 \leftarrow 0^{\ell_{\text{st}}}$, $\text{rdata}_0 \leftarrow 0^{\ell_{\text{CELL}}}$, $D_{\text{work},0} \leftarrow (0^{\ell_{\text{CELL}}})^{2^{\ell_{\text{ADDR}}}}$, $t \leftarrow 1$.
2. Let $(\text{done}_t, \text{st}_t, \tau_t, i_t, \text{wdata}_t, \text{out}_t) \leftarrow \text{CPU}(|D_1|, \dots, |D_{\mathcal{T}}|, w, \text{st}_{t-1}, \text{rdata}_{t-1})$.
3. If done_t is set, the execution halts.
4. If $\tau_t \in [\mathcal{T}]$, let $\text{rdata}_t \leftarrow D_{\tau_t}[i_t] \parallel 0^{\ell_{\text{CELL}} - \ell_{\text{cell}}}$ and $D_{\text{work},t} \leftarrow D_{\text{work},t-1}$.

5. Otherwise, $\tau_t = \text{work}$, let $\text{rdata}_t \leftarrow D_{\text{work},t-1}[i_t]$ and $D_{\text{work},t}$ be $D_{\text{work},t-1}$ with $D_{\text{work},t}[i_t]$ replaced by wdata_t .
6. Let $t \leftarrow t + 1$ and go back to Step 2.

We assume, without loss of generality, that all those constraints are respected during all executions.¹² For a halting execution,

- its *running time* $T = \text{time}(M, D_1, \dots, D_T, w)$ is the value of t when it halts;¹³
- its *running space* is $\text{space}(\dots) = \max_{t < T, \tau_t = \text{work}} i_t$;¹⁴
- its *state sequence* is $\text{stS}(\dots) = (\text{st}_1, \dots, \text{st}_{T-1})$;
- its *address sequence* is $\text{addrS}(\dots) = (\tau_1, i_1, \dots, \tau_{T-1}, i_{T-1})$;
- its *write sequence* is $\text{writeS}(\dots) = (\text{wdata}_1, \dots, \text{wdata}_{T-1})$;
- its *output sequence* is $\text{outS}(\dots) = (\text{out}_1, \dots, \text{out}_{T-1})$.

Remark 1 (output sequence). The machine does *not* necessarily produce non- \perp outputs at the end or even consecutively. In 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 timing of each non- \perp output are not supposed to be hidden as such information is incorporated in the output sequence.

Remark 2 (running space). A better name for $\text{space}(\dots)$ is *address bound*. Jumping ahead, the running space must be polynomially bounded in bounded laconic garbled RAM schemes, but can be exponentially large (and is hidden) in (full-fledged) LGRAM.

2.2 Laconic Garbled RAM

Our notion of garbling RAM laconically involves two steps. First, a reusable short digest is created for each input tape. The digest has length independent of that of the tape content and 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 three algorithms:

¹²For example, 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).

¹³If the execution does not halt, we say $\text{time}(\dots) = +\infty$.

¹⁴To be complete, $\text{space}(\dots)$ is defined to be 1 if the working tape is never accessed.

- $\text{Compress}(1^\lambda, 1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, \tau, D_\tau)$ takes as input a cell length ℓ_{cell} , an address length ℓ_{addr} , an input tape index $\tau \in [\mathcal{T}]$, and its content $D_\tau \in (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$. It outputs a short digest digest_τ . The algorithm runs in time $|D_\tau| \text{poly}(\lambda, \ell_{\text{cell}}, \ell_{\text{addr}})$ and its output length is $\text{poly}(\lambda, \ell_{\text{cell}}, \ell_{\text{addr}})$.
- $\text{Garble}(1^\lambda, T_{\text{max}}, M, \{\text{digest}_\tau\}_{\tau \in [\mathcal{T}]})$ takes as input a time bound $T_{\text{max}} \in \mathbb{N}_+$, 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 $\{L_{i,b}\}_{i \in [\ell_{\text{in}}], b \in \{0,1\}}$. The algorithm runs in polynomial time.
- $\text{Eval}^{D_1, \dots, D_\mathcal{T}}(1^\lambda, T_{\text{max}}, M, \{\text{digest}_\tau\}_{\tau \in [\mathcal{T}]}, \widehat{M}, \{L_i\}_{i \in [\ell_{\text{in}}]})$ takes as input T_{max} , M , the input tape digests, \widehat{M} , and one set of labels. Given random access to the input tapes, it is supposed to compute the output sequence. The algorithm runs in time

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

where w is the short input corresponding to the labels.

The scheme must be *correct*, i.e., for all $\lambda \in \mathbb{N}$, $\ell_{\text{cell}}, \ell_{\text{addr}} \in \mathbb{N}_+$, $T_{\text{max}} \in \mathbb{N}_+$, $\ell_{\text{in}} \in \mathbb{N}$, \mathcal{T} -tape RAM M , input tape contents $D_1, \dots, D_\mathcal{T} \in (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$, short input $w \in \{0, 1\}^{\ell_{\text{in}}}$ such that $M^{D_1, \dots, D_\mathcal{T}}(w)$ halts in time at most T_{max} , it holds that

$$\Pr \left[\begin{array}{l} \text{digest}_\tau \stackrel{\$}{\leftarrow} \text{Compress}(1^\lambda, 1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, \tau, D_\tau) \quad \text{for all } \tau \in [\mathcal{T}] \\ (\widehat{M}, \{L_{i,b}\}_{i \in [\ell_{\text{in}}], b \in \{0,1\}}) \stackrel{\$}{\leftarrow} \text{Garble}(1^\lambda, T_{\text{max}}, M, \{\text{digest}_\tau\}_{\tau \in [\mathcal{T}]}) \\ \vdots \\ \text{Eval}^{D_1, \dots, D_\mathcal{T}}(1^\lambda, T_{\text{max}}, M, \{\text{digest}_\tau\}_{\tau \in [\mathcal{T}]}, \widehat{M}, \{L_{i,w[i]}\}_{i \in [\ell_{\text{in}}]}) \\ = \text{outS}(M, D_1, \dots, D_\mathcal{T}, w) \end{array} \right] = 1.$$

Remark 3 (unboundedness). Our notion of LGRAM is *unbounded*, i.e., it is not necessary to know a polynomial upper bound of the instance running time upon garbling. By choosing an exponentially large T_{max} , one garbling works for all polynomial-time computation. In contrast is a *bounded* scheme for all polynomial-time computation, where T_{max} can be *any* polynomial, but it *must* be a polynomial, hence every garbling is restricted to *some* polynomial time bound upon creation. Unboundedness is reflected in both efficiency and security (below), where T_{max} is written in binary.

See also Remark 2 on RAM running space.

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

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive from it $1^{\ell_{\text{cell}}}$ and $1^{\ell_{\text{addr}}}$.
- **Tape Choices.** Repeat the following for \mathcal{T} rounds. In each round, \mathcal{A} chooses $\tau \in [\mathcal{T}]$ and $D_\tau \in (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$. Upon receiving such choice, run

$$\text{digest}_\tau \stackrel{\$}{\leftarrow} \text{Compress}(1^\lambda, 1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, \tau, D_\tau)$$

and send digest_τ to \mathcal{A} .

- **Challenge.** \mathcal{A} chooses an instance running time bound $1^{\bar{T}}$ (in unary), a time bound T_{\max} (in binary), a \mathcal{T} -tape RAM M , and two inputs (w_0, w_1) . Run

$$(\widehat{M}, \{L_{i,b}\}_{i \in [\ell_{\text{in}}], b \in \{0,1\}}) \stackrel{\$}{\leftarrow} \text{Garble}(1^\lambda, T_{\max}, M, \{\text{digest}_\tau\}_{\tau \in [\mathcal{T}]})$$

and send $(\widehat{M}, \{L_{i,w_\beta[i]}\}_{i \in [\ell_{\text{in}}]})$ to \mathcal{A} .

- **Guess.** \mathcal{A} outputs a bit β' . The output of the experiment is β' if all of the following conditions hold.
 - The τ 's in all rounds of **Tape Choices** are distinct.
 - Both $M^{D_1, \dots, D_\tau}(w_0)$ and $M^{D_1, \dots, D_\tau}(w_1)$ halt in time $T \leq \bar{T} \leq T_{\max}$ with identical output sequences $\text{outS}(\dots)$.

Otherwise, the output is set to 0.

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

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

Remark 4 (polynomial security). Although T_{\max} can be exponentially large, we only require security for polynomially large *instance* running time, which is captured by the requirement that the adversary must produce $1^{\bar{T}}$, 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 *bounded LGRAM* is obtained by modifying Definition 2 as follows:

- The evaluation procedure runs in time

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

- Correctness is required only if $M^{D_1, \dots, D_\tau}(w)$ 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 $1^{\bar{T}}$ and T_{\max}).
- In **Guess**, the second condition is strengthened to "... halt using space at most T_{\max} in time $T \leq T_{\max}$ with identical..."

Remark 5 (efficiency and security). Although evaluation efficiency is relaxed and security is restricted to polynomially large T_{\max} , the garbling procedure of a bounded LGRAM scheme still runs in time poly-logarithmic in T_{\max} . This is important for its bootstrapping to (unbounded) LGRAM.

2.3 Partially Hiding Functional Encryption and FE for Circuits

We define partially hiding functional encryption with respect to functionality

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

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 $y \in Y$ and is tied to some public $x \in X$. 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 $\varphi(f, x, y) = \perp$, 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 with

$$\varphi : F_\varphi \times X_\varphi \times Y_\varphi \rightarrow \{\perp\} \cup (\mathbb{N}_+ \times Z_\varphi) \quad \text{for each } \varphi \in \Phi_\lambda.$$

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

- $\text{Setup}(1^\lambda, \varphi)$ takes a functionality $\varphi \in \Phi_\lambda$ as input, and outputs a pair of master public/secret keys (mpk, msk) .
- $\text{KeyGen}(1^\lambda, \text{msk}, f)$ takes as input msk and a function description $f \in F_\varphi$. It outputs a secret key sk_f for f .
- $\text{Enc}(1^\lambda, \text{mpk}, x, y)$ takes as input mpk , a public input $x \in X_\varphi$, and a private input $y \in Y_\varphi$. It outputs a ciphertext ct_x of y tied to x .
- $\text{Dec}^{f, x, \text{sk}_f, \text{ct}_x}(1^\lambda, \text{mpk})$ takes mpk as input and is given random access to $f, x, \text{sk}_f, \text{ct}_x$. It is supposed to compute z in $\varphi(f, x, y) = (T, z)$ efficiently.

The scheme must be *correct*, i.e., 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

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

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

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

The following time-efficiency properties are considered:

- It has *linear-time* KeyGen [resp. Enc, Dec] if KeyGen [resp. Enc, Dec] runs in time $|f| \text{poly}(\lambda, |\varphi|)$ [resp. $(|x| + |y|) \text{poly}(\lambda, |\varphi|)$, $(T + |f| + |x| + |y|) \text{poly}(\lambda, |\varphi|)$].

- It has f -fast [resp. x -fast, y -fast] Dec if Dec runs in time $(T + |x| + |y|) \text{poly}(\lambda, |\varphi|)$ [resp. $(T + |f| + |y|) \text{poly}(\lambda, |\varphi|)$, $(T + |f| + |x|) \text{poly}(\lambda, |\varphi|)$].

The following size-efficiency properties are considered:

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

Security. We consider adaptive IND-CPA for polynomially bounded T .

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

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive from it some $\varphi \in \Phi_\lambda$ and $1^{\bar{T}}$. Run

$$(\text{mpk}, \text{msk}) \stackrel{\$}{\leftarrow} \text{Setup}(1^\lambda, \varphi)$$

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 some $f_q \in F_\varphi$. Upon receiving such query, run

$$\text{sk}_q \stackrel{\$}{\leftarrow} \text{KeyGen}(1^\lambda, \text{msk}, f_q)$$

and send sk_q to \mathcal{A} .

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

$$\text{ct} \stackrel{\$}{\leftarrow} \text{Enc}(1^\lambda, \text{mpk}, x, y_\beta)$$

and send ct to \mathcal{A} .

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

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

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

Otherwise, the outcome is set to 0.

FE for Circuits. We will use FE for circuits as a building block:

Definition 8 (FE for circuits). A *functional encryption scheme for circuits* is a PHFE scheme (Definition 5) for

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

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

Remark 6. The first output of $\varphi_{\ell,s}$ is just a placeholder 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.

Thanks to a long line of beautiful prior works, we can weaken the assumption of FE for circuits all the way down to an “obfuscation-minimum FE” with only polynomial security, summarized in the lemma below.

Lemma 1 ([KNTY19]). *Assuming the existence of “weakly selectively secure, 1-key, sublinearly succinct FE for circuits” (per definitions in [KNTY19]), i.e., a so-called “obfuscation-minimum FE with 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 behind certain subtleties in our formulation.

To obtain PHFE for RAM, we will employ the standard transformation [QWW18] of using FE for circuits to compute LGRAM. However, in LGRAM (Definition 2), the running time of Garble depends on the machine size. This dependency is inherited by all the efficiency parameters of the resultant PHFE for RAM if we associate each key with a RAM. To get rid of this dependency, we fix some universal RAM U of size $\text{poly}(\lambda)$ ¹⁵ upon setting up the scheme, and associate with each key a piece of code interpreted by U .

The other issue is that LGRAM puts an upper bound on the running time and its incorrectness in case of exceeding the time limit is propagated to the PHFE scheme. We avoid it¹⁶ by defining $\varphi = \perp$ if the running time exceeds some super-polynomial value prescribed upon set-up.

The above explains the intended usage of PHFE for RAM, yet we define it for general machines. 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 machine.

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*

$$\begin{aligned} \Phi &= \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}, & \Phi_\lambda &= \{\varphi_{M, T_{\max}}\}_M \text{ is a 2-tape RAM and } T_{\max} \in \mathbb{N}_+, \\ \varphi_{M, T_{\max}} &: F_M \times X_M \times Y_M \rightarrow \{\perp\} \cup (\mathbb{N}_+ \times Z), \\ F_M = X_M &= (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}, & Y_M &= \{0, 1\}^{\ell_{\text{in}}}, & Z &= \{\perp, 0, 1\}^*, \\ \varphi_{M, T_{\max}}(f, x, y) &= \begin{cases} (T, \text{outS}(M, f, x, y)), & \text{if } \text{time}(M, f, x, y) = T \leq T_{\max}; \\ \perp, & \text{otherwise;} \end{cases} \end{aligned}$$

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

¹⁵ U is not the same RAM across different values of λ – its input address length should be $\omega(\log \lambda)$ to accommodate all polynomially long input.

¹⁶An alternative solution is to blatantly reveal everything when the running time is too large so that correctness in that case can be fulfilled by executing the machine in the clear. Security is not affected because the adversary is not allowed to choose keys and ciphertexts exhibiting super-polynomial instance running time. However, non-halting computation still needs to be handled separately.

In a full-fledged PHFE for RAM, the machine has no 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

$$\begin{aligned} \Phi &= \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}, & \Phi_\lambda &= \{\varphi_{M, T_{\max}}\}_M \text{ is a 2-tape RAM with } \ell_{\text{in}}=0, \text{ and } T_{\max} \in \mathbb{N}_+, \\ \varphi_{M, T_{\max}} &: F_M \times X_M \times Y_M \rightarrow \{\perp\} \cup (\mathbb{N}_+ \times Z), \\ F_M = X_M = Y_M &= (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}, & Z &= \{\perp, 0, 1\}^*, \\ \varphi_{M, T_{\max}}(f, x, y) &= \begin{cases} (T, \text{outS}(M, f, x \| y, \varepsilon)), & \text{if } |x| + |y| \leq 2^{\ell_{\text{addr}}} \text{ and } \text{time}(M, f, x \| y, \varepsilon) = T \leq T_{\max}; \\ \perp, & \text{otherwise;} \end{cases} \end{aligned}$$

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

Remark 7 (unbounded scheme and polynomial security). When Definitions 5 and 7 are instantiated into PHFE for RAM (Definitions 9 and 10), the scheme is *unbounded*, meaning that T_{\max} can be exponentially large, yet security only holds for polynomially bounded instance running time. See also Remarks 3 and 4.

2.5 Indistinguishability Obfuscation

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

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

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

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

$$\{(1^\lambda, C_0, C_1, i\mathcal{O}(1^\lambda, C_0))\}_{\lambda \in \mathbb{N}} \approx \{(1^\lambda, C_0, C_1, i\mathcal{O}(1^\lambda, C_1))\}_{\lambda \in \mathbb{N}}.$$

Remark 8. The above definition does not allow the obfuscator to work by simply outputting the truth table, as the constraint of having polynomial-size domains only applies to security, not efficiency nor correctness. See also Remarks 3, 4, and 7.

Secure $i\mathcal{O}$ for polynomial-size domains is implied by polynomially secure FE for circuits, via either a tight security reduction [LT17] of FE-to- $i\mathcal{O}$ transformation or decomposable obfuscation [LZ17].

Lemma 2 ([LT17, LZ17]). *Assuming the existence of secure FE for circuits (Definition 8), there exists an indistinguishability obfuscator for circuits that is secure for polynomial-size domains.*

2.6 Laconic Oblivious Transfer

We will use selectively secure laconic oblivious transfer as a building block.

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

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

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

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

An *updatable* LOT has two additional algorithms:

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

Correctness requires that for all $\lambda \in \mathbb{N}$, $\ell_{\text{cell}}, \ell_{\text{addr}} \in \mathbb{N}_+$, $D \in (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$, $i \in [|D|]$, $\text{wdata} \in \{0, 1\}^{\ell_{\text{cell}}}$, $\{m_j^b\}_{j \in [\ell_{\text{hash}}]}^{b \in \{0,1\}}$ of identical length,

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

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 $\text{Hash0s}(1^\lambda, \text{hk}, S)$ computing the hash of $(0^{\ell_{\text{cell}}})^S$. More precisely, for all $\lambda \in \mathbb{N}$, $\ell_{\text{cell}}, \ell_{\text{addr}} \in \mathbb{N}_+$, $S \in [2^{\ell_{\text{addr}}}]$, it holds that

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

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 can be computed in time $\text{poly}(\lambda, \ell_{\text{cell}}, \ell_{\text{addr}})$.

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 SimRead such that for all polynomial-size $1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}$, $D \in (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$, $i \in [|D|]$, $\{m_j^b\}_{j \in [\ell_{\text{cell}}]}^{b \in \{0,1\}}$ of identical length, it holds that

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

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-size $1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}$, $D \in (\{0, 1\}^{\ell_{\text{cell}}})^{\leq 2^{\ell_{\text{addr}}}}$, $i \in [|D|]$, $\text{wdata} \in \{0, 1\}^{\ell_{\text{cell}}}$, $\{m_j^b\}_{j \in [\ell_{\text{hash}}]}^{b \in \{0,1\}}$ of identical length, letting D' be D with $D'[i]$ replaced by wdata , it holds that

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

Remark 9 (partially adaptive security). The above definitions are index- and message-selective. Yet, without loss of generality, we may assume that security holds even when the adversary is allowed to choose i and m_j^b 's adaptively (dependent 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 secure FE for circuits (Definition 8), there exists a read- and write-secure updatable LOT with fast initialization.*

2.7 Garbled Circuits

Following the formulation in [GS18a], we make the garbling procedure take the labels as input instead of letting it generate them. However, their formulation cannot be perfectly correct,¹⁷ which we fix by incorporating the point-and-permute [BMR90] technique.

Definition 16 (garbled circuits [Yao82]). A *circuit garbling scheme* with label length $\ell_L(\lambda) \leq \text{poly}(\lambda)$ consists of two efficient algorithms:

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

The scheme must be *correct*, i.e., for all $\lambda \in \mathbb{N}$, $C : \{0,1\}^n \rightarrow \{0,1\}^*$, $x \in \{0,1\}^n$,

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

The scheme is *secure* if there exists an efficient simulator $\widetilde{\text{Garble}}$ such that for all polynomial-size (C, x) ,

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

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

2.8 Puncturable Pseudorandom Function

We need puncturable pseudorandom functions as a building block.¹⁸

Definition 17 (PPRF [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 $\text{poly}(\lambda)$ -bounded] consists of two efficient algorithms:

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

¹⁷The inevitable correctness error of [GS18a] arises from the absence of point-and-permute string and the (albeit unlikely) possibility of two labels for one input bit being the same. See also Footnote 20.

¹⁸We 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 $\lambda \in \mathbb{N}$, $k \in \{0, 1\}^{\ell_{\text{key}}(\lambda)}$, $\mathfrak{p} \subseteq \{0, 1\}^{\ell_{\text{in}}(\lambda)}$, and input $x \in \{0, 1\}^{\ell_{\text{in}}(\lambda)} \setminus \mathfrak{p}$, it holds that

$$\Pr[\text{Eval}(1^\lambda, \text{Puncture}(1^\lambda, k, \mathfrak{p}), x) = \text{Eval}(1^\lambda, k, x)] = 1.$$

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

$$\{(1^\lambda, \mathfrak{p}, \mathring{k}_{\mathfrak{p}}, \{\text{Eval}(1^\lambda, k, x)\}_{x \in \mathfrak{p}})\}_{\lambda \in \mathbb{N}} \approx \{(1^\lambda, \mathfrak{p}, \mathring{k}_{\mathfrak{p}}, \{r_x\}_{x \in \mathfrak{p}})\}_{\lambda \in \mathbb{N}},$$

where $k \stackrel{\$}{\leftarrow} \{0, 1\}^{\ell_{\text{key}}(\lambda)}$, $\mathring{k}_{\mathfrak{p}} \stackrel{\$}{\leftarrow} \text{Puncture}(1^\lambda, k, \mathfrak{p})$, and $r_x \stackrel{\$}{\leftarrow} \{0, 1\}^{\ell_{\text{out}}(\lambda)}$ for all $x \in \mathfrak{p}$.

2.9 Secret-Key Encryption

We will use secret-key encryption as a building block.

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

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

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

$$\Pr[k \stackrel{\$}{\leftarrow} \text{Gen}(1^\lambda) : \text{Dec}(1^\lambda, \text{Enc}(1^\lambda, k, m)) = m] = 1.$$

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 bounded $\{m_q\}_{q \in [Q]}$, it holds that

$$\{(1^\lambda, 1^Q, \{m_q, c_q\}_{q \in [Q]})\}_{\lambda \in \mathbb{N}} \approx \{(1^\lambda, 1^Q, \{m_q, r_q\}_{q \in [Q]})\}_{\lambda \in \mathbb{N}},$$

where $k \stackrel{\$}{\leftarrow} \text{Gen}(1^\lambda)$, and $c_q \stackrel{\$}{\leftarrow} \text{Enc}(1^\lambda, k, m_q)$, $r_q \stackrel{\$}{\leftarrow} \{0, 1\}^{|c_q|}$ 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'd machine, we avoid having to incorporate randomness¹⁹ into the definition of RAM. We also include error (overflow) checking in our definition so that our LGRAM and PHFE can be made perfectly correct.²⁰

¹⁹Our LGRAM only works with deterministic RAM and dealing with garbling of randomized computation is cumbersome. An ORAM'd machine is probabilistic and cannot be directly fed into LGRAM with lesser security to obtain LGRAM with stronger security, so the notion of ORAM'd 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 it.

²⁰Overwhelming correctness is subtle to define, and it has multiple acceptable yet different definitions. Perfect correctness is also crucial [GKVW20] to some applications such as obfuscation. Therefore, we insist on perfect correctness in this work.

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

$$\text{MakeORAM} : (1^\lambda, 1^{\ell_{\text{CELL}}}, 1^{\ell_{\text{ADDR}}}, T_{\text{max}}) \mapsto (S'_{\text{max}}, 1^{\ell'_{\text{CELL}}}, 1^{\ell_r}, 1^{\ell_{\text{ost}}}, \{\text{ORW}_{t_0}\}_{t_0 \in [T_0]})$$

taking as input a logical cell length ℓ_{CELL} , a logical address length ℓ_{ADDR} , and a time bound $T_{\text{max}} \in \mathbb{N}_+$. It outputs a physical space bound S'_{max} , a physical cell length ℓ'_{CELL} , an ORAM randomness length ℓ_r , an ORAM state length ℓ_{ost} , and a sequence of circuits $\{\text{ORW}_{t_0}\}_{t_0 \in [T_0]}$ satisfying the following conditions:

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

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

Here,

- $i \in [2^{\ell_{\text{ADDR}}}]$ is the logical address to read from and write to,
- $\text{rdata} \in \{0, 1\}^{\ell_{\text{CELL}}}$ is the logical string that was read,
- $\text{wdata} \in \{0, 1\}^{\ell_{\text{CELL}}}$ is the logical string to write,
- $r_1, \dots, r_{T_0} \in \{0, 1\}^{\ell_r}$ are the ORAM randomness,
- $\text{ost}_1, \dots, \text{ost}_{T_0-1} \in \{0, 1\}^{\ell_{\text{ost}}}$ are the ORAM states,
- $i'_1, \dots, i'_{T_0} \in [S'_{\text{max}}]$ are the physical addresses to read from and write to,
- $\text{wdata}'_1, \dots, \text{wdata}'_{T_0} \in \{0, 1\}^{\ell'_{\text{CELL}}}$ are the physical string to write.

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

$$\begin{aligned} (S'_{\text{max}}, 1^{\ell'_{\text{CELL}}}, 1^{\ell_r}, 1^{\ell_{\text{ost}}}, \{\text{ORW}_{t_0}\}_{t_0 \in [T_0]}) & \leftarrow \text{MakeORAM}(1^\lambda, 1^{\ell_{\text{CELL}}}, 1^{\ell_{\text{ADDR}}}, T_{\text{max}}), \\ r_{t,t_0} & \stackrel{\$}{\leftarrow} \{0, 1\}^{\ell_r} \quad \text{for } t \in [T], t_0 \in [T_0], \end{aligned}$$

and consider the following process:

- Let $D_0 \leftarrow (0^{\ell_{\text{CELL}}})^{2^{\ell_{\text{ADDR}}}}$ and $D'_{0,T_0} \leftarrow (0^{\ell'_{\text{CELL}}})^{S'_{\text{max}}}$.
- For $t = 1, \dots, T$, do the following.
 - *Logical Access*. Let

$$\text{rdata}_t \leftarrow D_{t-1}[i_t], \quad D_t \leftarrow D_{t-1} \text{ with } D_t[i_t] \text{ replaced by } \text{wdata}_t.$$

– *Physical Accesses*. Perform T_0 physical steps by

$$\begin{aligned} (\text{ost}_{t,1}, i'_{t,1}, \text{wdata}'_{t,1}) &\leftarrow \text{orW}_1(i_t, \text{wdata}_t, r_{t,1}), \\ (\text{ost}_{t,t_0}, i'_{t,t_0}, \text{wdata}'_{t,t_0}) &\leftarrow \text{orW}_{t_0}(\text{ost}_{t,t_0-1}, \text{rdata}'_{t,t_0-1}, r_{t,t_0}) \quad \text{for all } 1 < t_0 < T_0, \\ (i'_{t,T_0}, \text{wdata}'_{t,T_0}, \widetilde{\text{rdata}}_t) &\leftarrow \text{orW}_{T_0}(\text{ost}_{t,T_0-1}, \text{rdata}'_{t,T_0-1}, r_{t,T_0}), \end{aligned}$$

where $D'_{t,0} \leftarrow D'_{t-1,T_0}$ and for all $t_0 \in [T_0]$,

$$\begin{aligned} \text{rdata}'_{t,t_0} &\leftarrow D'_{t,t_0-1}[i'_{t,t_0}] \quad (\text{rdata}'_{t,T_0} \text{ is never used}), \\ D'_{t,t_0} &\leftarrow D'_{t,t_0-1} \text{ with } D'_{t,t_0}[i'_{t,t_0}] \text{ replaced by } \text{wdata}'_{t,t_0}. \end{aligned}$$

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

$$\begin{aligned} \Pr[\text{any of } \text{orW}_{t,t_0} \text{ outputs } \perp] &\leq 2^{-\lambda} \quad \text{and} \\ \Pr[(\text{some of } \text{orW}_{t,t_0} \text{ outputs } \perp) \vee (\widetilde{\text{rdata}}_t = \text{rdata}_t \text{ for all } t \in [T])] &= 1. \end{aligned}$$

Remark 10 (error checking). In the definition above, error is indicated by \perp from orW_{t,t_0} 's. The correctness requirement is two-fold. First, it is negligibly likely that any error is reported. Second, if no error is reported, then all the logical accesses are perfectly fulfilled. Quantifying over all $T \leq T_{\max}$ allows us to further deduce that all the logical accesses are perfectly fulfilled *until* the first time an error is reported, ensuring 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 PartRnd and SimORAM such that for all $\lambda \in \mathbb{N}$, $\ell_{\text{CELL}}, \ell_{\text{ADDR}}, T_{\max} \in \mathbb{N}$, and all sequence $\{(i_t, \text{wdata}_t)\}_{t \in [T]}$ of logical accesses with $T \in [T_{\max}]$,

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

satisfies

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

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

The ORAM scheme in [CP13] has localized randomness, as observed in [CH16].

Lemma 4 ([CP13, CH16]). *There exists an ORAM with localized randomness.*

It should be noted that in our formulation, the ORAM need *not* encrypt the physical database, whose content is protected using a separate mechanism.

2.11 Primitives Related to Lower Bounds

Secret-Key PHFE. We consider secret-key PHFE with weak security in our lower bounds.

Definition 21 (secret-key PHFE and security). The definition of a *secret-key PHFE scheme* is obtained by modifying Definition 5 as follows:

- Setup only outputs msk, without an mpk.
- Enc takes msk as input instead of mpk.

The definition of *1-key very selective security* for public- or secret-key PHFE is obtained by modifying Definition 7 as follows:

- The adversary chooses $f_1, \{x_q, y_{0,q}, y_{1,q}\}_{q \in [Q]}$ together with $\varphi, 1^{\bar{T}}$ during **Setup**, after which it receives mpk, $sk_1, \{ct_q\}_{q \in [Q]}$ (but no mpk for secret-key PHFE), where

$$ct_q \stackrel{s}{\leftarrow} \begin{cases} \text{Enc}(1^\lambda, \text{mpk}, x_q, y_{\beta,q}), & \text{for public-key PHFE in } \text{Exp}_{\text{PHFE}}^\beta; \\ \text{Enc}(1^\lambda, \text{msk}, x_q, y_{\beta,q}), & \text{for secret-key PHFE in } \text{Exp}_{\text{PHFE}}^\beta. \end{cases}$$

- There is no **Query I, Challenge, Query II** phases after **Setup**.
- The constraint of **Guess** is

$$\begin{aligned} & |y_{0,q}| = |y_{1,q}|, \\ \text{and } & \varphi(f_1, x_q, y_{0,q}) = \varphi(f_1, x_q, y_{1,q}) = (T_q, z_q) \neq \perp \quad \text{for all } q, \\ \text{and } & T_q \leq \bar{T} \quad \text{for all } q. \end{aligned}$$

The definition of *1-ciphertext very selective security* for public- or secret-key PHFE is obtained by modifying Definition 7 as follows:

- The adversary chooses $\{f_q\}_{q \in [Q]}, x, y_0, y_1$ together with $\varphi, 1^{\bar{T}}$ during **Setup**, after which it receives mpk, $\{sk_q\}_{q \in [Q]}, ct$ (but no mpk for secret-key PHFE).
- There is no **Query I, Challenge, Query II** phases after **Setup**.

The definition of *1-key 1-ciphertext very selective security* for public- or secret-key PHFE is obtained by modifying Definition 7 as follows:

- The adversary chooses f_1, x, y_0, y_1 together with $\varphi, 1^{\bar{T}}$ during **Setup**, after which it receives sk_1, ct (without mpk).
- There is no **Query I, Challenge, Query II** phases after **Setup**.

Given a secure full-fledged PHFE for RAM, it is straight-forward to obtain a secure full-fledged secret-key PHFE for RAM by putting mpk into msk, sk , which does not degrade any efficiency parameter.

DE-PIR. We consider public- or secret-key doubly efficient private information retrieval.

Definition 22 (DE-PIR [BIPW17,CHR17] and security). A *public-key doubly efficient private information retrieval (PK-DE-PIR) protocol* consists of four algorithms:

- $\text{Process}(1^\lambda, D)$ takes the database $D \in \{0, 1\}^*$ as input. It outputs a public key pk and a processed database \tilde{D} .
- $\text{Query}(1^\lambda, \text{pk}, i)$ takes as input pk and an index $i \in [|D|]$. It outputs an encrypted query ct and a query-specific secret σ .
- $\text{Resp}^{\tilde{D}}(1^\lambda, \text{ct})$ takes ct as input. Given random access to \tilde{D} , it outputs a response ρ .
- $\text{Dec}(1^\lambda, \sigma, \rho)$ takes σ, ρ as input. It is supposed to recover $D[i]$.

The algorithm Process runs in polynomial time and $|\text{pk}| \leq \text{poly}(\lambda, \log |D|)$. The other algorithms run in time $|D|^{1-\varepsilon} \text{poly}(\lambda, \log |D|)$ for some constant $0 < \varepsilon \leq 1$. The protocol must be *correct*, i.e., for all $\lambda \in \mathbb{N}$, $D \in \{0, 1\}^*$, $i \in [|D|]$, it holds that

$$\Pr \left[\begin{array}{l} (\text{pk}, \tilde{D}) \stackrel{\$}{\leftarrow} \text{Process}(1^\lambda, D) \\ (\text{ct}, \sigma) \stackrel{\$}{\leftarrow} \text{Query}(1^\lambda, \text{pk}, i) : \text{Dec}(1^\lambda, \sigma, \rho) = D[i] \\ \rho \stackrel{\$}{\leftarrow} \text{Resp}^{\tilde{D}}(1^\lambda, \text{ct}) \end{array} \right] = 1.$$

The protocol is *secure* if for all polynomial-size D , $\{i_{\beta,q}\}_{\beta \in \{0,1\}, q \in [Q]}$ with $D[i_{0,q}] = D[i_{1,q}]$ for all $q \in [Q]$, it holds that

$$\{(1^\lambda, D, \text{pk}, \tilde{D}, \{i_{0,q}, i_{1,q}, \text{ct}_{0,q}\}_{q \in [Q]})\}_{\lambda \in \mathbb{N}} \approx \{(1^\lambda, D, \text{pk}, \tilde{D}, \{i_{0,q}, i_{1,q}, \text{ct}_{1,q}\}_{q \in [Q]})\}_{\lambda \in \mathbb{N}}.$$

The definitions of a *secret-key DE-PIR (SK-DE-PIR) protocol* and its *security* are obtained with the following modifications:

- Process outputs a secret key sk instead of pk .
- Query takes sk as input instead of pk .
- For security, sk does not appear in the distributions (there is also no pk).

The security notion we consider is weaker than those defined in [BIPW17,CHR17,LMW23] in that it is indistinguishability-based, very selective, and does not hide the output nor the database (for secret-key schemes). Nevertheless, standard techniques can be used to lift our security notion to the strongest (simulation-based, adaptive, output-hiding, and for secret-key schemes, database-hiding) at no cost of asymptotic efficiency, i.e., they are existentially equivalent. (For completeness, we present the proofs in Section A.) In our proof of technical barriers, the direct construction only achieves the weak security notion, minimizing the security and functionality required from the underlying PHFE thus making the barrier stronger, with the additional benefit of simplifying the proof.

3 Efficiency Trade-Offs of PHFE for RAM

Before presenting the trade-offs, we remark that one cannot expect Dec for a full-fledged PHFE for RAM to run in time $T^{1-\varepsilon} \text{poly}(\lambda, |\varphi|, |f|, |x|, |y|, \log T)$ for any constant $0 < \varepsilon \leq 1$, due to the time-hierarchy theorems for RAM [CR73,Iva82] (i.e., certain computations cannot be sped up).

3.1 Contention Between Storage Overhead and Decryption Time

In this section, we show that when $A, B < 1$, it is impossible to achieve

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

simultaneously for a secure PHFE for RAM, where polynomial factors in the security parameter are ignored. This leaves us with two candidate optima:

- $A = 0$ and $B = 1$ for succinct keys; or
- $A = 1$ and $B = 0$ for f -fast decryption.

Similarly, if $A, B < 1$, it is impossible to achieve

$$|\text{ct}| = O(|x|^A) \text{ poly}(|y|) \quad \text{and} \quad T_{\text{Dec}} = O(T + |f| + |x|^B + |y|)$$

simultaneously, which implies a contention between succinct ciphertexts and x -fast decryption.

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

Theorem 5 (contention of $|f|$ -dependency between $|\text{sk}|$ and T_{Dec} ; \spadesuit). *For a secret-key 1-key 1-ciphertext very selectively secure full-fledged PHFE for RAM (Definitions 10 and 21),²¹ if*

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

for infinitely many λ , where A, B, C are constants, then $A \geq 1$ or $B \geq 1$.

Theorem 6 (contention of $|x|$ -dependency between $|\text{ct}|$ and T_{Dec}). *For a secret-key 1-key 1-ciphertext very selectively secure full-fledged PHFE for RAM (Definitions 10 and 21), if*

$$|\text{ct}| \leq |x|^A (\lambda + |\varphi| + |y|)^C \quad \text{and} \quad T_{\text{Dec}} \leq (T + |f| + |x|^B) (\lambda + |\varphi| + |y|)^C$$

for infinitely many λ , where A, B, C are constants, then $A \geq 1$ or $B \geq 1$.

Theorem 6 reduces to Theorem 5 by first switching the roles of sk, ct using the double encryption method [NY90,BS18].²² Therefore, we only prove Theorem 5.

Proof (Theorem 5). Let (Setup, KeyGen, Enc, Dec) be a secure full-fledged secret-key PHFE for RAM. Suppose for contradiction that $A, B < 1 - 5\varepsilon$ for some $0 < \varepsilon < \frac{1}{5}$. By enlarging C as needed, we could assume $|\varphi| \leq \lambda^C - \lambda - 1$ for all sufficiently large λ , where

$$\begin{aligned} \varphi &= (M_\lambda, 2^\lambda), \quad f = R \in \{0, 1\}^{\leq 2^\lambda}, \quad x = \perp, \\ y &= \begin{cases} (I, w) = (i_1, w[1], \dots, i_n, w[n]) \in ([2^\lambda] \times \{0, 1\})^{\leq 2^\lambda}; \\ z = (\perp, z[1], \dots, \perp, z[n]) \in (\{\perp\} \times \{0, 1\})^{\leq 2^\lambda}; \end{cases} \end{aligned}$$

²¹The proof only requires the PHFE scheme to be mildly expressive.

²²The implementation is similar to Construction 6. A new secret key for f is an underlying ciphertext tied to $x' = f$ encrypting $y' = (\beta', k_{\beta'})$, where β' is a choice bit and $k_{\beta'}$ is a PRF (SKE) key, both stored in the new msk . A new ciphertext tied to x of y is w together with an underlying secret key tied to $f' = x \| w$, where w contains $w_{\beta'}$ (an encryption of y under $k_{\beta'}$) and $w_{1-\beta'}$ (used in the security proof). This transformation ensures that the efficiency in Theorem 6 is translated to that in Theorem 5.

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

Under appropriate encoding and step circuit design, y has exactly n cells and M halts in exactly $(2n + 1)$ steps.

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

$$|\text{sk}| \leq |f|^A (\lambda + |\varphi|)^C \quad \text{and} \quad T_{\text{dec}} \leq (T + |f|^B + |y|)(\lambda + |\varphi| + |x|)^C$$

By setting

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

we would have $n < N < 2^\lambda$ for sufficiently large λ . Consider the following adversary \mathcal{A} (Definitions 21):

- Upon launching, it computes φ, N, n defined above, sets up the PHFE scheme for φ , and submits 1^{2n+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

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

It challenges with

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

and obtains a ciphertext ct encrypting either y_0 or y_1 .

- It runs $\text{Dec}^{f,x,\text{sk},\text{ct}}()$ 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 make the following claims.

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

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

Claim 8 (♣). For sufficiently large λ with efficiency,

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

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, $A \geq 1$ or $B \geq 1$. \square

To prove Claim 7, we need the following lemma about incompressibility of information:

Lemma 9 ([DTT10]). *Suppose $E : S \times U \rightarrow V$ and $D : S \times V \rightarrow U$ are functions and S is a distribution over S , then*

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

Proof (Claim 7). 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 9 will generate the following inequality equivalent to the desired one:

$$\Pr[|L \cap I| \leq \lfloor N^{1-4\epsilon} \rfloor \text{ in } \text{Exp}_{\text{PHFE}}^0] \leq \frac{1}{4}.$$

Formally, let

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

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

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

- Parse $I = (i_1, \dots, i_n)$ 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

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

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

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

$$\begin{aligned} v_1 &= 0^{\lfloor N^{1-4\epsilon} \rfloor - |\text{sk}| - 1} \mathbf{1} \parallel \text{sk}, \\ v_2[i] &= \begin{cases} R[\ell_j], & \text{if } |\{\ell_1, \dots, \ell_{j-1}\} \cap I| = i - 1 \text{ and } |\{\ell_1, \dots, \ell_{j-1}, \ell_j\} \cap I| = i; \\ 0, & \text{if no such } j \text{ exists.} \end{cases} \end{aligned}$$

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

$$|\text{sk}| \leq |f|^A (\lambda + |\varphi|)^C \leq N^{1-5\epsilon} (\lambda + (\lambda^C - \lambda - 1))^C \leq N^{1-5\epsilon} \lambda^{C^2} < \lfloor N^{1-4\epsilon} \rfloor - 1$$

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 $\lfloor N^{1-4\epsilon} \rfloor$ bits.

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

- Run $\text{ct} \leftarrow \text{Enc}(\text{msk}, \perp, (I, w); r_{\text{Enc}})$.
- 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

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

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

- if $R[i] = \perp$ and $j < \lfloor N^{1-4\epsilon} \rfloor$, then let $j \leftarrow j + 1$ and set $R[i] \leftarrow v_2[j]$;
- if $R[i] = \perp$ and $j = \lfloor N^{1-4\epsilon} \rfloor$, then abort by outputting 0^n ;
- otherwise, $R[i] \neq \perp$, 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 the PHFE algorithms 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 $\text{Exp}_{\text{PHFE}}^0$. If s and u are such that $|L \cap I| \leq \lfloor N^{1-4\epsilon} \rfloor$ in $E(s, u)$, then D will successfully recover u . By Lemma 9,

$$\begin{aligned} \Pr[|L \cap I| \leq \lfloor N^{1-4\epsilon} \rfloor \text{ in } \text{Exp}_{\text{PHFE}}^0] &= \Pr[|L \cap I| \leq \lfloor N^{1-4\epsilon} \rfloor \text{ in } E(s, u)] \\ &\stackrel{s \leftarrow \mathcal{S}}{u \leftarrow \mathcal{U}} \leq \Pr[D(s, E(s, u)) = u] \\ &\stackrel{s \leftarrow \mathcal{S}}{u \leftarrow \mathcal{U}} \leq \frac{|V|}{|U|} = \frac{2^{2\lfloor N^{1-4\epsilon} \rfloor}}{2^n} = 2^{2\lfloor N^{1-4\epsilon} \rfloor - \lfloor N^{1-3\epsilon} \rfloor} \leq \frac{1}{4} \end{aligned}$$

for sufficiently large λ with efficiency. □

Proof (Claim 8). For sufficiently large λ with efficiency,

$$\begin{aligned} |L| \leq T_{\text{Dec}} &\leq (T + |f|^B + |y|)(\lambda + |\varphi| + |x|)^C \\ &\leq ((2n + 1) + N^{1-5\epsilon} + n)(\lambda + (\lambda^C - \lambda - 1) + 1)^C \\ &\leq (3N^{1-3\epsilon} + N^{1-5\epsilon} + 1)\lambda^{C^2} \leq N^{1-2\epsilon}. \end{aligned}$$

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

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

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,

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

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

$$\Pr[|L \cap I| > N^{1-4\epsilon} \text{ in } \text{Exp}_{\text{PHFE}}^1] \leq \frac{\mathbb{E}[|L \cap I|]}{N^{1-4\epsilon}} \leq \frac{N^{1-5\epsilon}}{N^{1-4\epsilon}} = N^{-\epsilon} \leq \frac{1}{4}. \quad \square$$

3.2 Barrier to Fast Decryption

In this section, we show that PHFE schemes with f -/ x -/ y -fast Dec imply doubly efficient private information retrieval (DE-PIR) protocols of various flavors. In particular, such schemes with linear-size keys or ciphertexts imply optimal DE-PIR with

$$|\tilde{D}| \leq |D| \text{poly}(\lambda, \log |D|), \quad T_{\text{Query}}, T_{\text{Resp}}, T_{\text{Dec}} \leq \text{poly}(\lambda, \log |D|),$$

which remains unknown from any assumptions. The latest development in DE-PIR, due to [LMW23], achieves

$$|\tilde{D}| \leq |D|^{1+\epsilon} \text{poly}(\lambda, \log |D|), \quad T_{\text{Query}}, T_{\text{Resp}}, T_{\text{Dec}} \leq \text{poly}(\lambda, \log |D|)$$

for any constant $\epsilon > 0$.

The main idea underlying our proof of technical barrier is as follows. Suppose decryption is fast with respect to a certain input, then the component (key or ciphertext) associated with that input is used to encode the database, which, together with the database in the clear, stored on the server as the processed database. The other component is used to encode a query index. To respond to a query, the server simply performs PHFE decryption. Our idea is straight-forward to implement when Dec is f -fast and it requires the double encryption method [NY90, BS18] when Dec is x -fast or y -fast.

The formal statements do not require full fastness.

Theorem 10 (DE-PIR from f -fast PHFE; \spadesuit). *If there exists a 1-key selectively secure full-fledged PHFE for RAM with*

$$|\text{sk}_f| = |f|^A \text{poly}(\lambda, |\varphi|), \quad T_{\text{Dec}} = |f|^B \text{poly}(\lambda, |\varphi|, T, |x|, |y|),$$

for constants A and $0 \leq B < 1$,²³ then there exists a DE-PIR with

$$\begin{aligned} |\tilde{D}| &= |D| + |D|^A \text{poly}(\lambda, \log |D|), & T_{\text{Resp}} &= |D|^B \text{poly}(\lambda, \log |D|), \\ T_{\text{Query}} &= \text{poly}(\lambda, \log |D|), & T_{\text{Dec}} &= \text{O}(1). \end{aligned}$$

If the PHFE is public-key [resp. secret-key], then so is the DE-PIR.

²³The proof requires lesser expressiveness from the PHFE scheme than that of Theorem 5, so the lower bound $A \geq 1$ might not apply. Note that for DE-PIR, $|\tilde{D}| = \Omega(|D|)$ is necessary.

Theorem 11 (SK-DE-PIR from x -fast PHFE; \spadesuit). *If there exists a secret-key 1-ciphertext selectively secure full-fledged PHFE for RAM with*

$$|\text{ct}_x| = |x|^A \text{poly}(\lambda, |\varphi|, |y|), \quad T_{\text{Dec}} = |x|^B \text{poly}(\lambda, |\varphi|, T, |f|, |y|),$$

for constants A and $0 \leq B < 1$,²⁴ then there exists an SK-DE-PIR with

$$\begin{aligned} |\tilde{D}| &= |D| + |D|^A \text{poly}(\lambda, \log |D|), & T_{\text{Resp}} &= |D|^B \text{poly}(\lambda, \log |D|), \\ T_{\text{Query}} &= \text{poly}(\lambda, \log |D|), & T_{\text{Dec}} &= \text{O}(1). \end{aligned}$$

Theorem 12 (SK-DE-PIR from y -fast PHFE). *If there exists a secret-key 1-ciphertext selectively secure full-fledged PHFE for RAM with*

$$|\text{ct}_x| = |y|^A \text{poly}(\lambda, |\varphi|, |x|), \quad T_{\text{Dec}} = |y|^B \text{poly}(\lambda, |\varphi|, T, |f|, |x|),$$

for constants A and $0 \leq B < 1$,²⁵ then there exists an SK-DE-PIR with

$$\begin{aligned} |\tilde{D}| &= |D|^A \text{poly}(\lambda, \log |D|), & T_{\text{Resp}} &= |D|^B \text{poly}(\lambda, \log |D|), \\ T_{\text{Query}} &= \text{poly}(\lambda, \log |D|), & T_{\text{Dec}} &= \text{O}(1). \end{aligned}$$

Note that in all of the theorems above, if $A = 1$ and $B = 0$, then the resultant DE-PIR will be optimal. We will only prove Theorems 10 and 11. The proof of Theorem 12 is analogous to that of Theorem 11.

Proof (Theorem 10). Let $\text{PHFE} = (\text{PHFE.Setup}, \text{PHFE.KeyGen}, \text{PHFE.Enc}, \text{PHFE.Dec})$ be the PHFE in the premise. We only prove the case of public-key PHFE, and the secret-key case is similar. The PK-DE-PIR protocol works as follows:

- $\text{Process}(D)$ constructs a 2-tape RAM $M^{D_1, D_2}() = D_1[i]$, where $i \in [|D_1|]$ is interpreted from D_2 , such that $|M| \leq \text{poly}(\log |D|)$. It sets $T_{\text{max}} = 2$, $f = D$, and runs

$$(\text{phmpk}, \text{phmsk}) \stackrel{\$}{\leftarrow} \text{PHFE.Setup}(\varphi_{M, T_{\text{max}}}), \quad \text{phsk} \stackrel{\$}{\leftarrow} \text{PHFE.KeyGen}(\text{phmsk}, f).$$

The algorithm outputs $\text{pk} = \text{phmpk}$ and $\tilde{D} = (D, \text{phsk})$.

- $\text{Query}(\text{pk}, i)$ sets $x = \perp$, $y = i$, runs

$$\text{phct} \stackrel{\$}{\leftarrow} \text{PHFE.Enc}(\text{phmpk}, x, y),$$

and outputs $\text{ct} = \text{phct}$ and $\sigma = \perp$.

- $\text{Resp}^{\tilde{D}}(\text{ct})$ runs and outputs $\rho \stackrel{\$}{\leftarrow} \text{PHFE.Dec}^{D, \perp, \text{phsk}, \text{phct}}(\text{phmpk})$.
- $\text{Dec}(\sigma, \rho)$ outputs ρ .

It is readily verified that the protocol is correct and enjoys the promised efficiency. Security follows from 1-key very selective security of PHFE in a straight-forward way. \square

²⁴Similarly to Footnote 23, we do not claim $A \geq 1$.

²⁵It is necessary that $A \geq 1$.

Note that in the above proof, the functionality of PHFE is extremely simple (namely, read-then-output). Theorem 10 provided strong indication of technical breakthrough required to construct PHFE with optimal decryption time, given that no candidate of optimal DE-PIR was known prior to the initial write-up of this work (and is still not known to a satisfactory degree).

Proof (Theorem 11). Let PHFE = (PHFE.Setup, PHFE.KeyGen, PHFE.Enc, PHFE.Dec) be the secret-key PHFE in the premise. Let SKE = (SKE.Gen, SKE.Enc, SKE.Dec) be a secret-key encryption scheme with pseudorandom ciphertexts. Our SK-DE-PIR protocol works as follows:

- Process(D) constructs a 2-tape RAM

$$M^{D_1, D_2}() = D'[\text{SKE.Dec}(k_\beta, c_\beta)] \quad \text{for } D_1 = c_0 \| c_1, D_2 = D' \| \beta \| k_\beta,$$

where c_0, c_1 are two possible SKE ciphertexts, β is a bit, and k_β is an SKE key. Such an M can be constructed with $|M| \leq \text{poly}(\lambda, \log |D|)$. The algorithm picks an appropriate $T_{\max} \leq \text{poly}(\lambda, \log |D|)$, and sets/samples/runs

$$\begin{aligned} x &\leftarrow D, & \beta &\stackrel{\$}{\leftarrow} \{0, 1\}, & k_\beta &\stackrel{\$}{\leftarrow} \text{SKE.Gen}(), & y &\leftarrow \beta \| k_\beta, \\ \text{phmsk} &\stackrel{\$}{\leftarrow} \text{PHFE.Setup}(\varphi_{M, T_{\max}}), & \text{phct} &\stackrel{\$}{\leftarrow} \text{PHFE.Enc}(\text{msk}, x, y). \end{aligned}$$

It outputs $\text{sk} = (\text{phmsk}, \beta, k_\beta)$ and $\tilde{D} = (D, \text{phct})$.

- Query(sk, i) samples/runs

$$c_\beta \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_\beta, i), \quad c_{1-\beta} \stackrel{\$}{\leftarrow} \{0, 1\}^{|c_\beta|}, \quad f \leftarrow c_0 \| c_1, \quad \text{phsk} \stackrel{\$}{\leftarrow} \text{PHFE.KeyGen}(\text{msk}, f),$$

and outputs $\text{ct} = (c_0, c_1, \text{phsk})$ and $\sigma = \perp$.

- Resp \tilde{D} (ct) runs and outputs $\rho \stackrel{\$}{\leftarrow} \text{PHFE.Dec}^{c_0 \| c_1, D, \text{phsk}, \text{phct}}()$.
- Dec(σ, ρ) outputs ρ .

Again, correctness and efficiency are readily verified. For security, let $k_{1-\beta} \stackrel{\$}{\leftarrow} \text{SKE.Gen}()$ and consider the following hybrids:

- H_0^b . This is one of the distributions in Definition 22, i.e.,

$$\begin{aligned} D, \quad \text{phct} &\stackrel{\$}{\leftarrow} \text{PHFE.Enc}(\text{msk}, D, \beta \| k_\beta), & \{i_{0,q}\}_{q \in [Q]}, & \{i_{1,q}\}_{q \in [Q]}, \\ (\text{in ct}_q) \quad \text{phsk}_q &\stackrel{\$}{\leftarrow} \text{PHFE.KeyGen}(\text{msk}, c_{q,0} \| c_{q,1}), & c_0 \| c_1, & \\ c_{q,\beta} &\stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_\beta, i_{\beta,q}), & c_{q,1-\beta} &\stackrel{\$}{\leftarrow} \{0, 1\}^{|c_{q,\beta}|}. \end{aligned}$$

- H_1^b . In this hybrid, we make $c_{q,1-\beta}$ an encryption of $i_{1-b,q}$ under $k_{1-\beta}$, i.e.,

$$\begin{aligned} D, \quad \text{phct} &\stackrel{\$}{\leftarrow} \text{PHFE.Enc}(\text{msk}, D, \beta \| k_\beta), & \{i_{0,q}\}_{q \in [Q]}, & \{i_{1,q}\}_{q \in [Q]}, \\ (\text{in ct}_q) \quad \text{phsk}_q &\stackrel{\$}{\leftarrow} \text{PHFE.KeyGen}(\text{msk}, c_{q,0} \| c_{q,1}), & c_0 \| c_1, & \\ c_{q,\beta} &\stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_\beta, i_{b,q}), & c_{q,1-\beta} &\stackrel{\$}{\leftarrow} \boxed{\text{SKE.Enc}(k_{1-\beta}, i_{1-b,q})}. \end{aligned}$$

$H_0^b \approx H_1^b$ for each $b \in \{0, 1\}$ by the ciphertext pseudorandomness of SKE.

- H_2^b . In this hybrid, we rename $(\beta \oplus b)$ to γ , making it

$$\begin{aligned}
D, \text{ phct} &\stackrel{\$}{\leftarrow} \text{PHFE.Enc}(\text{msk}, D, \boxed{\gamma \oplus b} \parallel k_{\boxed{\gamma \oplus b}}), \quad \{i_{0,q}\}_{q \in [Q]}, \quad \{i_{1,q}\}_{q \in [Q]}, \\
(\text{in ct}_q) \text{ phsk}_q &\stackrel{\$}{\leftarrow} \text{PHFE.KeyGen}(\text{msk}, c_{q,0} \parallel c_{q,1}), \quad c_0 \parallel c_1, \\
c_{q, \boxed{\gamma}} &\stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\boxed{\gamma}}, i_{\boxed{0}, q}), \quad c_{q, \boxed{1-\gamma}} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\boxed{1-\gamma}}, i_{\boxed{1}, q}),
\end{aligned}$$

where $\gamma \stackrel{\$}{\leftarrow} \{0, 1\}$. This change is conceptual, so $H_1^b \equiv H_2^b$ for each $b \in \{0, 1\}$.

$H_2^0 \approx H_2^1$ follows from the 1-ciphertext very selective security of PHFE. Therefore, the SK-DE-PIR protocol is secure. \square

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-size domains, which is in turn 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,

$$\begin{aligned}
\text{LOT} = & (\text{LOT.HashGen}, \text{LOT.Hash}, \text{LOT.SendRead}, \text{LOT.RecvRead}, \\
& \text{LOT.SendWrite}, \text{LOT.RecvWrite}, \text{LOT.Hash0s}),
\end{aligned}$$

- GC = (GC.Garble, GC.Eval) a circuit garbling scheme, and
- PPRF = (PPRF.Puncture, PPRF.Eval) a puncturable PRF.

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

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

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

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

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

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

samples PPRF key k , prepares GenAugCPU using M (Figure 3), and runs

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

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

$$\begin{aligned} \pi_1 : & \pi_1^w, \pi_1^{\text{st}}, \pi_1^{\text{rdata}}, \pi_1^h, \\ \{L_{1,i,b}\}_{i,b} : & \{L_{1,i,b}^w\}_{i \in [\ell_{\text{in}}], b}, \{L_{1,i,b}^{\text{st}}\}_{i \in [\ell_{\text{st}}], b}, \{L_{1,i,b}^{\text{rdata}}\}_{i \in [\ell_{\text{CELL}}], b}, \{L_{1,i,b}^h\}_{i \in [\ell_{\text{HASH}}], b}, \end{aligned}$$

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

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

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

Hardwired. $\text{hk}_\tau, \text{hk}_{\text{work}}$, LOT hash keys for the tapes;
 $h_\tau, |D_\tau|$, LOT hashes and lengths of the input tapes;
 k , PPRF key.

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

Output. $\widehat{\text{AugCPU}}_t$, the garbled “augmented step circuit” at time t , computed as follows.

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

Figure 3. The circuit GenAugCPU in Construction 1.

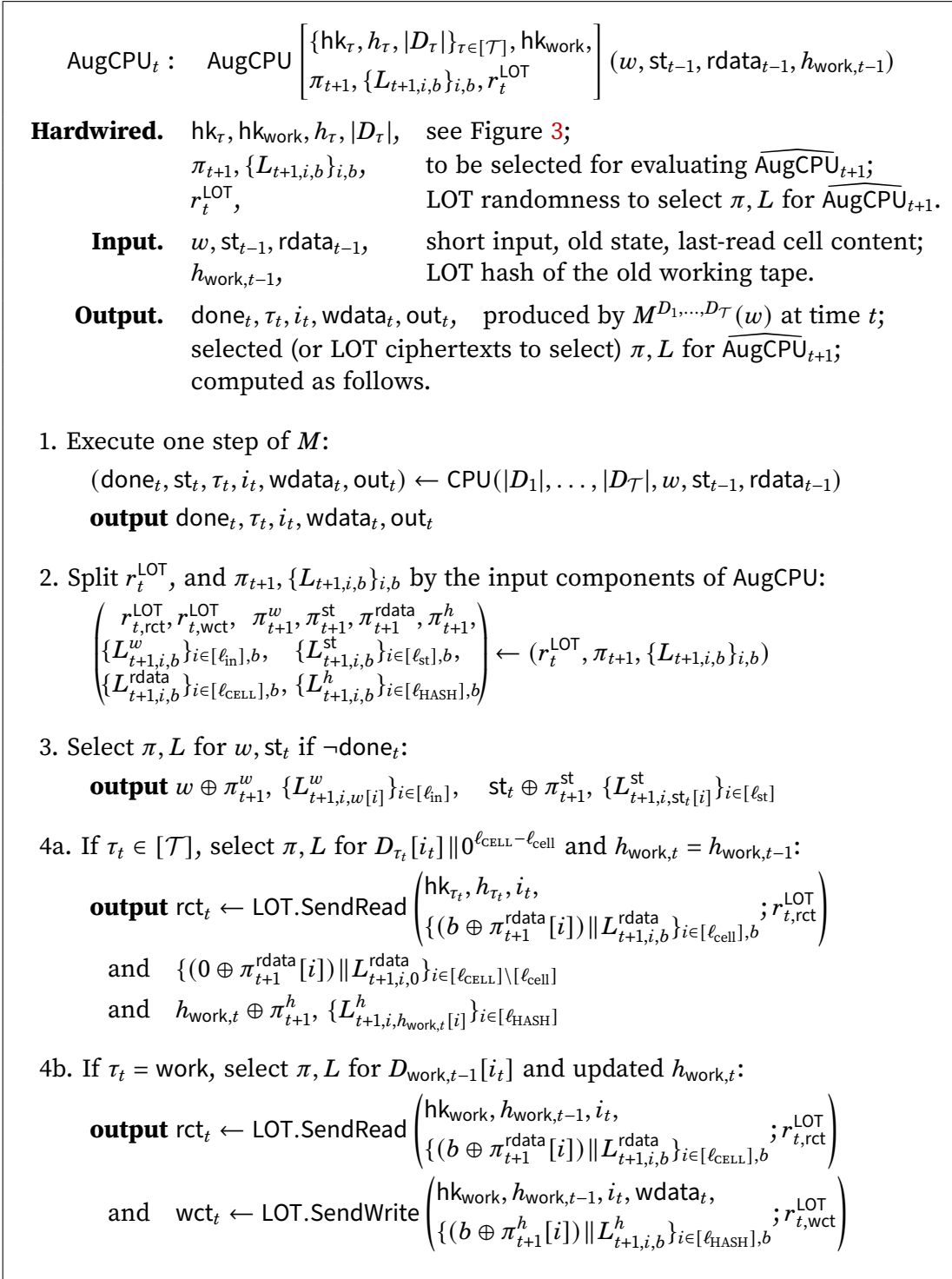


Figure 4. The circuit AugCPU in Construction 1.

- $\text{Eval}^{D_1, \dots, D_{\mathcal{T}}}(T_{\max}, M, \{\text{digest}_{\tau}\}_{\tau \in [\mathcal{T}]}, \widehat{M}, \{L_i\}_{i \in [\ell_{\text{in}]}})$ 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

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

2. Writing

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

run²⁶

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

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

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

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

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, $\tau_t = \text{work}$, then pick $\text{rct}_t, \text{wct}_t$ from Y_t , run

$$\begin{aligned} \{(\text{rdata}_t[i] \oplus \pi_{t+1,i}^{\text{rdata}}) \parallel L_{t+1,i,\text{rdata}_t[i]}^{\text{rdata}}\}_{i \in [\ell_{\text{CELL}}]} &\leftarrow \text{LOT.RecvRead}^{\widehat{D}_{\text{work},t-1}}(\text{hk}_{\text{work}}, h_{\text{work},t-1}, i_t, \text{rct}_t), \\ \{(h_{\text{work},t} \oplus \pi_{t+1,i}^h) \parallel L_{t+1,i,h_{\text{work},t}[i]}^h\}_{i \in [\ell_{\text{HASH}}]} &\leftarrow \text{LOT.RecvWrite}^{\widehat{D}_{\text{work},t-1}} \left(\begin{array}{l} \text{hk}_{\text{work}}, h_{\text{work},t-1}, \text{wct}_t \\ i_t, \text{wdata}_t, \end{array} \right), \end{aligned}$$

updating $\widehat{D}_{\text{work},t-1}$ into $\widehat{D}_{\text{work},t}$. 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 $\widehat{\text{AugCPU}}_t$ is evaluated on $w, \text{st}_{t-1}, \text{rdata}_{t-1}, h_{\text{work},t-1}$ 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 $T_{\max} \text{poly}(\lambda, |M|)$.

²⁶ 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 $\widehat{\text{AugCPU}}_{t-1}$ as explained later.

4.2 Security

Theorem 13 (¶). *Suppose in Construction 1, $i\mathcal{O}$ is secure for polynomial-size 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 $\text{Exp}_{\text{LGRAM}}^\beta$, the LGRAM labels are selected for w_β , and each AugCPU_t performs a step of $M^{D_1, \dots, D_T}(w_\beta)$, for which we write $M^\circ(w_\beta)$ hereafter in this proof. Along the hybrids from $\text{Exp}_{\text{LGRAM}}^0$ to $\text{Exp}_{\text{LGRAM}}^1$, we gradually switch the trailing steps from those for $M^\circ(w_0)$ to those for $M^\circ(w_1)$.

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

- when $t < t$, it runs the t^{th} step of $M^\circ(w_0)$;
 - 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 $t = t$, it is simulated and outputs the labels so that AugCPU_{t+1} runs the $(t + 1)^{\text{st}}$ step of $M^\circ(w_1)$, i.e., it switches the execution from $M^\circ(w_0)$ to $M^\circ(w_1)$;
- when $t > t$, it is garbled normally but runs the t^{th} step of $M^\circ(w_1)$, due to the behavior of AugCPU_t .

There are two special cases:

- when $t = 0$, 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 $M^\circ(w_1)$ when the LGRAM is evaluated normally;
- when $t > T$ for $t \in \{t\} \cup \mathfrak{s}$, 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.²⁷

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

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

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

Formally, let

$$\begin{aligned} \text{stS}(M, D_1, \dots, D_T, w_0) &= (\text{st}_{0,1}, \dots, \text{st}_{0,T-1}), \\ \text{stS}(M, D_1, \dots, D_T, w_1) &= (\text{st}_{1,1}, \dots, \text{st}_{1,T-1}), \end{aligned}$$

²⁷This 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 $(t + 1)^{\text{st}}$ step, and consequently (and inductively), we can pretend that the set of labels for the $(t + 1)^{\text{st}}$ step corresponding to the input to the last existent step were revealed, so that the $(t + 1)^{\text{st}}$ 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.

$$\begin{aligned} \text{addrS}(\dots, w_0) &= \text{addrS}(\dots, w_1) = (\tau_1, i_1, \dots, \tau_{T-1}, i_{T-1}), \\ \text{writeS}(\dots, w_0) &= \text{writeS}(\dots, w_1) = (\text{wdata}_1, \dots, \text{wdata}_{T-1}), \\ \text{outS}(\dots, w_0) &= \text{outS}(\dots, w_1) = (\text{out}_1, \dots, \text{out}_{T-1}). \end{aligned}$$

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

$$X_{b,t} = w_b \parallel \text{st}_{b,t-1} \parallel \text{rdata}_{t-1} \parallel h_{\text{work},t-1} \quad \forall b \in \{0, 1\}.$$

In $H_{t,s}$, we change $\widehat{\text{GenAugCPU}}$ in \widehat{M} to

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

where

- $\widehat{\text{GenAugCPU}}'$ is shown in Figure 5,
- $\mathring{k}_{\{t\} \cup s}$ is a PPRF key punctured at $\{t\} \cup s$,

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

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

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

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

1. Check for hardwired steps, if $t \in \{t\} \cup s$:
output hardwired $\widehat{\text{AugCPU}}_t$ and **terminate**
2. Otherwise, obtain randomness, either hardwired or from PPRF:
 $(\pi_t, \{L_{t,i,b}\}_{i,b}, r_t^{\text{LOT}}, r_t^{\text{GC}}) \leftarrow \text{PPRF.Eval}(\mathring{k}_{\{t\} \cup s}, t)$
 $(\pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}) \leftarrow \begin{cases} \text{hardwired,} & \text{if } t+1 \in \{t\} \cup s; \\ \text{PPRF.Eval}(\mathring{k}_{\{t\} \cup s}, t+1) \text{ with } r_{t+1}^{\text{LOT}}, r_{t+1}^{\text{GC}} \text{ removed,} & \text{if } t+1 \notin \{t\} \cup s; \end{cases}$
3. Garble AugCPU (Figure 4) as done in $\widehat{\text{GenAugCPU}}$ (Figure 3):
 $\text{AugCPU}_t \leftarrow \text{AugCPU} \left[\begin{array}{l} \{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, \\ \pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}, r_t^{\text{LOT}} \end{array} \right]$
output $\widehat{\text{AugCPU}}_t \leftarrow \text{GC.Garble}(\text{AugCPU}_t, \pi_t, \{L_{t,i,b}\}_{i,b}, r_t^{\text{GC}})$

Figure 5. The circuit $\widehat{\text{GenAugCPU}}'$ in the proof of Theorem 13.

- π_t 's and $L_{t,i,b}$'s are random strings, and
- $\widetilde{\text{AugCPU}}_t$'s are the simulated garbled step circuits.

We still use $\pi_t, L_{t,i,b}$ for part of the PPRF output when $t \notin \{t\} \cup \mathfrak{s}$.
 $\widetilde{\text{AugCPU}}_t$'s are generated by

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

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

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

$$w_b \oplus \pi_{t+1}^w, \{L_{t+1,i,w_b[i]}^w\}_{i \in [\ell_{\text{in}}]}, \quad \text{st}_{b,t} \oplus \pi_{t+1}^w, \{L_{t+1,i,\text{st}_{b,t}[i]}^{\text{st}}\}_{i \in [\ell_{\text{st}}]}.$$

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

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

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

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

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

Claim 14 (¶). $\text{Exp}_{\text{LGRAM}}^0 \approx H_{T_{\max}+1, \emptyset}$ and $\text{Exp}_{\text{LGRAM}}^1 \approx H_{0, \emptyset}$.

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

Claim 15 (Rule A; ¶). *A gray pebble can be placed on or removed from t^* if $t^* = 1$ or a gray pebble is placed on $(t^* - 1)$. Formally, $H_{t,\mathfrak{s}} \approx H_{t,\{t^*\} \cup \mathfrak{s}}$ for $t^* \notin \mathfrak{s}$ and $t^* < t$ if $t^* = 1$ or $t^* - 1 \in \mathfrak{s}$.*

Claim 16 (Rule B; ¶). *A gray pebble on t^* can be replaced by a black pebble if black pebbles are placed on all vertices greater than t^* . Formally, $H_{t,\mathfrak{s}} \approx H_{t-1,\mathfrak{s} \setminus \{t-1\}}$ if $t-1 \in \mathfrak{s}$ (here, $t^* = t-1$). Moreover, $H_{1,\emptyset} \approx H_{0,\emptyset}$.*

Claim 16 is different from rule B in [GOS18] (no need for a gray pebble on $(t^* - 1)$ 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 \bar{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 $t^* = 1$ in rule A is just a special case of $t^* - 1 = 0 \in \mathfrak{s}$, and the separate case $H_{1,\emptyset} \approx H_{0,\emptyset}$ in rule B becomes a special case of the general rule (which would read $H_{1,\{0\}} \approx H_{0,\emptyset}$ 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 17 ([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 $O(\log T_{\max})$ gray pebbles at any time during the $\text{poly}(T_{\max})$ 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*

$$t_0 = T_{\max} + 1, \mathfrak{s}_0 = \emptyset, \quad c_1, t_1, \mathfrak{s}_1, \quad c_2, t_2, \mathfrak{s}_2, \quad \dots,$$

$$c_{\text{poly}(T_{\max})-1}, t_{\text{poly}(T_{\max})-1} = 1, \mathfrak{s}_{\text{poly}(T_{\max})-1} = \emptyset, \quad c_{\text{poly}(T_{\max})}, t_{\text{poly}(T_{\max})} = 0, \mathfrak{s}_{\text{poly}(T_{\max})} = \emptyset,$$

such that $\max_j |\mathfrak{s}_j| = O(\log T_{\max})$ and $H_{t_{j-1}, \mathfrak{s}_{j-1}} \approx H_{t_j, \mathfrak{s}_j}$ by Claim $c_j \in \{15, 16\}$.

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

Proof (Theorem 13). By Claims 14, 15, and 16, and Lemma 17, $\text{Exp}_{\text{LGRAM}}^0 \approx \text{Exp}_{\text{LGRAM}}^1$. \square

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.

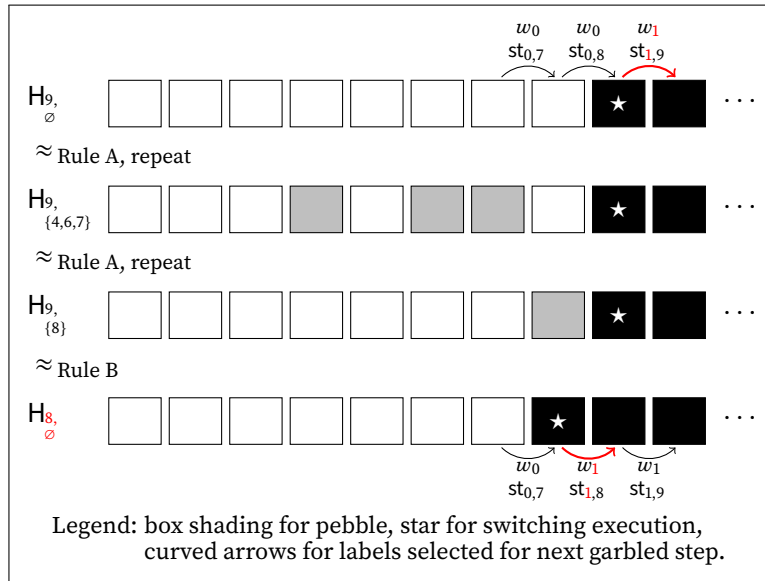


Figure 6. Typical hybrid transitions in the proof of Theorem 13.

Proof (Claim 14). Both follow from the security of $i\mathcal{O}$ for polynomial-size domains. \square

Proof (Claim 15). Let t, \mathfrak{s}, t^* be such that $t^* \notin \mathfrak{s}$ and $t^* < t$ and either $t^* = 1$ or $t^* - 1 \in \mathfrak{s}$. The hybrids below are mostly identical to $H_{t, \mathfrak{s}}$ except the contents hardwired into $\text{GenAugCPU}'$ are modified. We specify them:

- G_0 . This is just $H_{t,s}$, where the hardwired information is

$$\{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, t, s, \mathring{k}_{\{t\} \cup s},$$

$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\text{AugCPU}}_t\}_{t \in \{t\} \cup s}.$$

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

$$\{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, t, \boxed{\{t^*\} \cup s}, \boxed{\mathring{k}_{\{t,t^*\} \cup s}},$$

$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\text{AugCPU}}_t\}_{t \in \{t\} \cup s}, \boxed{\text{AugCPU}_{t^*}},$$

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

$$(\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}, r_{t^*}^{\text{LOT}}, r_{t^*}^{\text{GC}}) \leftarrow \text{PPRF.Eval}(k, t^*),$$

$$\text{AugCPU}_{t^*} \leftarrow \text{AugCPU} \left[\begin{array}{l} \{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, \\ \pi_{t^*+1}, \{L_{t^*+1,i,b}\}_{i,b}, r_{t^*}^{\text{LOT}} \end{array} \right],$$

$$\widehat{\text{AugCPU}}_{t^*} \leftarrow \text{GC.Garble}(\text{AugCPU}_{t^*}, \pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}; r_{t^*}^{\text{GC}}).$$

This is different from $\widehat{\text{AugCPU}}_t$'s for $t \in \{t\} \cup s$, which are simulated. The absence of π_{t^*} and $\{L_{t^*,i,b}\}_{i,b}$ in $\text{GenAugCPU}'$ is fine, because they are only used in Step 2 (Figure 5) for $t = t^* - 1$ and $t = t^*$. 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 $\text{GenAugCPU}'$ in G_0 and G_1 have identical truth tables (and are padded appropriately to the same size), and $G_0 \approx G_1$ follows from the security of $i\mathcal{O}$ for polynomial-size domains.

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

$$\{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, t, \{t^*\} \cup s, \mathring{k}_{\{t,t^*\} \cup s},$$

$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\text{AugCPU}}_t\}_{t \in \{t\} \cup s}, \widehat{\text{AugCPU}}_{t^*},$$

where

$$(\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}, r_{t^*}^{\text{LOT}}, r_{t^*}^{\text{GC}}) \leftarrow \boxed{\text{uniformly random}},$$

$$\text{AugCPU}_{t^*} \leftarrow \text{AugCPU} \left[\begin{array}{l} \{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, \\ \pi_{t^*+1}, \{L_{t^*+1,i,b}\}_{i,b}, r_{t^*}^{\text{LOT}} \end{array} \right],$$

$$\widehat{\text{AugCPU}}_{t^*} \leftarrow \text{GC.Garble}(\text{AugCPU}_{t^*}, \pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}; r_{t^*}^{\text{GC}}).$$

$G_1 \approx G_2$ follows from the security of PPRF.

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

$$\{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in [\mathcal{T}]}, \text{hk}_{\text{work}}, t, \{t^*\} \cup s, \mathring{k}_{\{t,t^*\} \cup s},$$

$$\{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\text{AugCPU}}_t\}_{t \in \{t\} \cup s}, \widehat{\text{AugCPU}}_{t^*},$$

where

$$\begin{aligned}
& (\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}, r_{t^*}^{\text{LOT}}, r_{t^*}^{\text{GC}}) \leftarrow \text{uniformly random}, \\
& \widehat{\text{AugCPU}}_{t^*} \stackrel{\$}{\leftarrow} \text{GC.Garble}(1^{|\text{AugCPU}|}, Z_{0,t^*}, X_{0,t^*} \oplus \pi_{t^*}, \{L_{t^*,i,X_{0,t^*}[i]}\}_i), \\
& Z_{0,t^*} : \pi_{t^*+1}, \{L_{t^*+1,i,b}\}_{i,b} \begin{cases} \text{in the clear,} \\ \text{in } \widetilde{\text{rct}}_{t^*} \leftarrow \text{LOT.SendRead}(\dots; r_{t^*,\text{rct}}^{\text{LOT}}), \\ \text{in } \widetilde{\text{wct}}_{t^*} \leftarrow \text{LOT.SendWrite}(\dots; r_{t^*,\text{wct}}^{\text{LOT}}). \end{cases}
\end{aligned}$$

This is different from $Z_{b,t}$'s for $t \in \{t\} \cup \mathfrak{s}$, which might contain simulated $\widetilde{\text{rct}}_t$'s and $\widetilde{\text{wct}}_t$'s. In G_2 , the augmented step circuit $\widehat{\text{AugCPU}}_{t^*}$ is garbled with truly random $\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}, r_{t^*}^{\text{GC}}$. If $t^* = 1$, then $\pi_1, \{L_{1,i,b}\}_{i,b}$ only additionally 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 $t^* - 1 \in \mathfrak{s}$, then they are only additionally used to simulate

$$\widehat{\text{AugCPU}}_{t^*-1} \stackrel{\$}{\leftarrow} \text{GC.Garble}(\dots, Z_{0,t^*-1}, \dots),$$

where Z_{0,t^*-1} contains only $(X_{0,t^*} \oplus \pi_{t^*})$ and $\{L_{t^*,i,X_{0,t^*}[i]}\}_i$, potentially in the clear, in $\widetilde{\text{rct}}_{t^*-1}$, in $\widetilde{\text{wct}}_{t^*-1}$, and by imagination.²⁸ Also, $\widehat{\text{AugCPU}}_t(X_{0,t^*}) = Z_{0,t^*}$. It follows from the security of GC that $G_2 \approx G_3$.

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

$$\begin{aligned}
& \{\text{hk}_\tau, h_\tau, |D_\tau|\}_{\tau \in \{T\}}, \text{hk}_{\text{work}}, \quad t, \quad \{t^*\} \cup \mathfrak{s}, \quad \mathring{k}_{\{t,t^*\} \cup \mathfrak{s}}, \\
& \{\pi_t, \{L_{t,i,b}\}_{i,b}, \widehat{\text{AugCPU}}_t\}_{t \in \{t\} \cup \mathfrak{s}}, \quad \widehat{\text{AugCPU}}_{t^*},
\end{aligned}$$

where

$$\begin{aligned}
& (\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}, r_{t^*}^{\text{LOT}}) \leftarrow \text{uniformly random}, \\
& \widehat{\text{AugCPU}}_{t^*} \stackrel{\$}{\leftarrow} \text{GC.Garble}(1^{|\text{AugCPU}|}, Z_{0,t^*}, X_{0,t^*} \oplus \pi_{t^*}, \{L_{t^*,i,X_{0,t^*}[i]}\}_i), \\
& Z_{0,t^*} : \begin{cases} \text{in the clear,} \\ \text{in } \widetilde{\text{rct}}_{t^*} \stackrel{\$}{\leftarrow} \text{LOT.SimRead}(\dots), \\ \text{in } \widetilde{\text{wct}}_{t^*} \stackrel{\$}{\leftarrow} \text{LOT.SimWrite}(\dots). \end{cases}
\end{aligned}$$

The LOT ciphertexts $\text{rct}_{t^*}, \text{wct}_{t^*}$ are encrypted using truly random $r_{t^*}^{\text{LOT}}$ in G_3 . Each database is known before its LOT hash key is provided to the adversary: for an input tape, hk_τ is provided after D_τ is chosen; for the working tape, D_{work,t^*-1} and $i_{t^*}, \text{wdata}_{t^*}$ are known once the adversary commits to the challenge M, w_0, w_1 , after which hk_{work} is provided. Therefore, it follows from the read/write security of LOT that $G_3 \approx G_4$.

²⁸When $t^* > T$, none of $\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}$ appears in Z_{0,t^*-1} . We pretend $X_{0,t^*} = X_{0,T}$ and that $(X_{0,t^*} \oplus \pi_{t^*})$ and $\{t^*, i, X_{0,t^*}[i]\}_i$ are revealed, imagining that the $(t^*)^{\text{th}}$ step repeated the last existent step.

- G_5 . In this hybrid, we hardwire $\pi_{t^*}, \{L_{t^*,i,b}\}_{i,b}$ into $\text{GenAugCPU}'$, i.e., the hardwired information is

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

$G_4 \approx G_5$ follows from the security of $i\mathcal{O}$ for polynomial-size domains.

By inspection, G_5 is just $H_{t,\{t^*\} \cup \mathfrak{s}}$. Therefore, $H_{t,\mathfrak{s}} \equiv G_0 \approx G_5 \equiv H_{t,\{t^*\} \cup \mathfrak{s}}$. \square

Proof (Claim 16). Let t, \mathfrak{s} be such that $t = 1$ or $t - 1 \in \mathfrak{s}$. The hybrids are mostly identical to $H_{t,\mathfrak{s}}$ except for the contents hardwired into $\text{GenAugCPU}'$:

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

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

- G_1 . In this hybrid, we no longer directly hardwire $\pi_t, \{L_{t,i,b}\}_{i,b}$ into $\text{GenAugCPU}'$ for $t = t$, i.e., the hardwired information is

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

where

$$\begin{aligned} & (\pi_t, \{L_{t,i,b}\}_{i,b}) \stackrel{\mathfrak{s}}{\leftarrow} \text{uniformly random}; \\ \left\{ \begin{array}{ll} \widehat{M}, \text{ LGRAM labels} \leftarrow X_{0,1} \oplus \pi_1, \{L_{1,i,X_{0,1}[i]}\}_i, \dots, & \text{if } t = 1; \\ \widehat{\text{AugCPU}}_{t-1} \stackrel{\mathfrak{s}}{\leftarrow} \text{GC.Garble} \left(\begin{array}{c} 1^{|\text{AugCPU}|}, Z_{0,t-1}, \\ X_{0,t-1} \oplus \pi_{t-1}, \{L_{t-1,i,X_{0,t-1}[i]}\}_i \end{array} \right), \\ Z_{0,t-1} : X_{0,t} \oplus \pi_t, \{L_{t,i,X_{0,t}[i]}\}_i \text{ in the clear, } \widetilde{\text{rct}}_{t-1}, \widetilde{\text{wct}}_{t-1}, & \text{if } t - 1 \in \mathfrak{s}; \\ \widehat{\text{AugCPU}}_t \stackrel{\mathfrak{s}}{\leftarrow} \text{GC.Garble} \left(\begin{array}{c} 1^{|\text{AugCPU}|}, Z_{1,t}, \\ X_{0,t} \oplus \pi_t, \{L_{t,i,X_{0,t}[i]}\}_i \end{array} \right), \\ Z_{1,t} : X_{1,t+1} \oplus \pi_{t+1}, \{L_{t+1,i,X_{1,t+1}[i]}\}_i \text{ in the clear, } \widetilde{\text{rct}}_t, \widetilde{\text{wct}}_t, & \text{if } t \leq T_{\max}. \end{array} \right. \end{aligned}$$

Note that $\text{GenAugCPU}'$ still indirectly contains $\pi_t, \{L_{t,i,b}\}_{i,b}$ via \widehat{M} and LGRAM labels, or $Z_{0,t-1}$. Removal of their direct hardwiring does not alter the truth table of $\text{GenAugCPU}'$ as they are only used in Step 2 (Figure 5) for $t = t$ and $t = t - 1$, 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-size domains, $G_0 \approx G_1$.

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

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

where

$$\begin{aligned}
& (\pi_t, \{L_{t,i,b}\}_{i,b}) \stackrel{\$}{\leftarrow} \text{uniformly random;} \\
& \left\{ \begin{array}{l} \widehat{M}, \text{ LGRAM labels} \leftarrow \boxed{X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i}, \dots, \quad \text{if } t = 1; \\ \widehat{\text{AugCPU}}_{t-1} \stackrel{\$}{\leftarrow} \text{GC.Garble} \left(\begin{array}{l} 1^{|\text{AugCPU}|}, Z_{0,t-1}, \\ X_{0,t-1} \oplus \pi_{t-1}, \{L_{t-1,i,X_{0,t-1}[i]}\}_i \end{array} \right), \\ Z_{0,t-1} : \boxed{X_{1,t} \oplus \pi_1, \{L_{t,i,X_{1,t}[i]}\}_i} \text{ in the clear, } \widetilde{\text{rct}}_{t-1}, \widetilde{\text{wct}}_{t-1}, \quad \text{if } t-1 \in \mathfrak{s}; \\ \widehat{\text{AugCPU}}_t \stackrel{\$}{\leftarrow} \text{GC.Garble} \left(\begin{array}{l} 1^{|\text{AugCPU}|}, Z_{1,t}, \\ \boxed{X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i} \end{array} \right), \\ Z_{1,t} : X_{1,t+1} \oplus \pi_{t+1}, \{L_{t+1,i,X_{1,t+1}[i]}\}_i \text{ in the clear, } \widetilde{\text{rct}}_t, \widetilde{\text{wct}}_t, \quad \text{if } t \leq T_{\max}. \end{array} \right.
\end{aligned}$$

This change does not alter the distribution, i.e., $G_1 \equiv G_2$, because π_t is a one-time pad that hides $X_{0,t}$ or $X_{1,t}$ and either set of $\{L_{t,i,b}\}_{i,b}$ chosen by $X_{0,t}$ and $X_{1,t}$ is simply random.

- G_3 . In this hybrid, we undo the simulation of $\widetilde{\text{rct}}_t$ and $\widetilde{\text{wct}}_t$, i.e., the hardwired information is

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

where

$$\begin{aligned}
& (\pi_t, \{L_{t,i,b}\}_{i,b}, \boxed{r_t^{\text{LOT}}}) \stackrel{\$}{\leftarrow} \text{uniformly random;} \\
& \left\{ \begin{array}{l} \widehat{M}, \text{ LGRAM labels} \leftarrow X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i, \dots, \quad \text{if } t = 1; \\ \widehat{\text{AugCPU}}_{t-1} \stackrel{\$}{\leftarrow} \text{GC.Garble} \left(\begin{array}{l} 1^{|\text{AugCPU}|}, Z_{0,t-1}, \\ X_{0,t-1} \oplus \pi_{t-1}, \{L_{t-1,i,X_{0,t-1}[i]}\}_i \end{array} \right), \\ Z_{0,t-1} : X_{1,t} \oplus \pi_1, \{L_{t,i,X_{1,t}[i]}\}_i \text{ in the clear, } \widetilde{\text{rct}}_{t-1}, \widetilde{\text{wct}}_{t-1}, \quad \text{if } t-1 \in \mathfrak{s}; \\ \widehat{\text{AugCPU}}_t \stackrel{\$}{\leftarrow} \text{GC.Garble} \left(\begin{array}{l} 1^{|\text{AugCPU}|}, Z_{1,t}, \\ X_{1,1} \oplus \pi_1, \{L_{1,i,X_{1,1}[i]}\}_i \end{array} \right), \\ Z_{1,t} : \boxed{\pi_{t+1}, \{L_{t+1,i,b}\}_{i,b}} \text{ in the clear, } \boxed{\text{rct}_t, \text{wct}_t}, \quad \text{if } t \leq T_{\max}. \end{array} \right.
\end{aligned}$$

$G_2 \approx G_3$ by the read/write security of LOT.

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

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

where

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

Since $\text{AugCPU}_t(X_{1,t}) = Z_{1,t}$, it follows from the security of GC that $G_3 \approx G_4$.

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

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

where

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

$G_4 \approx G_5$ 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

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

and \widehat{M} and the LGRAM labels given to the adversary contain $(X_{1,1} \oplus \pi_1)$ and $\{L_{1,i,X_{1,1}[i]}\}_i$ if $t = 1$. By the security of $i\mathcal{O}$ for polynomial-size domains, $G_5 \approx G_6$.

By inspection, G_6 is just $H_{t-1, \mathfrak{s} \setminus \{t-1\}}$. Therefore, $H_{t, \mathfrak{s}} \equiv G_0 \approx G_6 \equiv H_{t-1, \mathfrak{s} \setminus \{t-1\}}$. \square

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

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

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

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

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

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

$$\{L_b^{\text{B}}\}_b, \{L_{i,b}^{\text{W}_0}\}_{i,b}, \{L_{i,b}^{\text{W}_1}\}_{i,b}, \{L_{i,b}^{\text{K}_0}\}_{i,b}, \{L_{i,b}^{\text{K}_1}\}_{i,b}, \{L_{i,b}^{\text{T}'}\}_{i,b}, \{L_{i,b}^{\text{wdata}'}\}_{i,b}.$$

The algorithm sets and outputs

$$\begin{aligned} \widehat{M} &= (\widehat{M}', \{\text{labels for B} = \beta', K_0 = k_0, K_1 = k_1, T' = 0, \text{wdata}' = \perp\}), \\ L_{i,b} &= L_{i,b}^{\text{W}_0} \parallel L_{i,b}^{\text{W}_1} \quad \text{for } i \in [\ell_{\text{in}}], b \in \{0, 1\}. \end{aligned}$$

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

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

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);
 $\widetilde{st}_{0,t-1}, \widetilde{st}_{1,t-1}$, 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 $\widetilde{rdata}_{0,t-1} || 0^{\ell'_{\text{CELL}} - \ell_{\text{CELL}}} = \widetilde{rdata}_{1,t-1} || 0^{\ell'_{\text{CELL}} - \ell_{\text{CELL}}}$

if $t \neq 1$ and $\tau_{t-1} = \text{work}$:

compute $\widetilde{rdata}_{0,t-1}, \widetilde{rdata}_{1,t-1}$

$\left\{ \begin{array}{l} \text{from } rdata'_{t-1} = t' || (\widetilde{rdata}_{0,t-1} \oplus \text{PPRF.Eval}(k_0, t')) || (\widetilde{rdata}_{1,t-1} \oplus \text{PPRF.Eval}(k_1, t')), \\ \hspace{15em} \text{if } t' \neq 0; \\ \text{as } 0^{\ell_{\text{CELL}}}, 0^{\ell_{\text{CELL}}}, \hspace{15em} \text{otherwise;} \end{array} \right.$

2. Execute one step in the active copy:

$(\text{done}_t, \widetilde{st}_{B,t}, \tau_t, i_t, \widetilde{wdata}_{B,t}, \text{out}_t) \leftarrow \text{CPU}(\ell_1, \dots, \ell_{\mathcal{T}}, w_B, \widetilde{st}_{B,t-1}, \widetilde{rdata}_{B,t-1})$

3. Optionally execute one step in the inactive copy:

$(\widetilde{st}_{1-B,t}, \widetilde{wdata}_{1-B,t}) \leftarrow \begin{cases} (0^{\ell_{\text{st}}}, 0^{\ell_{\text{CELL}}}), & \text{if } t > T'; \\ \text{from CPU} \left(\ell_1, \dots, \ell_{\mathcal{T}}, w_{1-B}, \widetilde{st}_{1-B,t-1}, \widetilde{rdata}_{1-B,t-1} \right), & \text{if } t \leq T'. \end{cases}$

4. Translate $wdata$ for M into $wdata'$ for M' :

$wdata'_t \leftarrow \begin{cases} \perp, & \text{if } \tau_t \in [\mathcal{T}]; \\ wdata', & \text{if } \tau_t = \text{work} \text{ and } t = T' \text{ and } wdata' \neq \perp; \\ t || (\widetilde{wdata}_{0,t} \oplus \text{PPRF.Eval}(k_0, t)) || (\widetilde{wdata}_{1,t} \oplus \text{PPRF.Eval}(k_1, t)), & \text{otherwise;} \end{cases}$

5. Output and finish:

output $(\text{done}_t, \text{newst}', \tau_t, i_t, wdata'_t, \text{out}_t)$

where newst' contains incremented t and $\widetilde{st}_{0,t}, \widetilde{st}_{1,t}, \tau_t, i_t$

Figure 7. The machine M' in Construction 2.

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

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

The execution of M' has the following important invariants:

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

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

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

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

Theorem 18 (¶). *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 18). $\text{Exp}_{\text{LGRAM}}^{\beta}$ of the constructed scheme corresponds to $\text{Exp}_{\text{LGRAM}}^{\beta}$ of LGRAM' with machine M' and input

$$B = \beta' \stackrel{\$}{\leftarrow} \{0, 1\}, \quad W_0 = w_{\beta}, \quad K_0 = k_0, \quad W_1 = w_{\beta}, \quad K_1 = k_1, \quad T' = 0, \quad \text{wdata}' = \perp,$$

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_0^{β} . This is just $\text{Exp}_{\text{LGRAM}'}^{\beta}$, i.e.,

$$B = \beta', \quad W_{\beta'} = w_{\beta}, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{\beta}, \quad K_{1-\beta'} = k_{1-\beta'}, \quad T' = 0, \quad \text{wdata}' = \perp.$$

- H_1^{β} . In this hybrid, we change W_{1-B} to $w_{1-\beta}$, i.e.,

$$B = \beta', \quad W_{\beta'} = w_{\beta}, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = \boxed{w_{1-\beta}}, \quad K_{1-\beta'} = k_{1-\beta'}, \quad T' = 0, \quad \text{wdata}' = \perp.$$

The only place M' uses W_{1-B} is the branch $t > T'$ in Step 3 of M' (Figure 7). But $T' = 0$ and that branch is never taken. Therefore, for each $\beta \in \{0, 1\}$, the executions of M' in H_0^{β} and H_1^{β} satisfy the condition of fixed-memory security of LGRAM', and $H_0^{\beta} \approx H_1^{\beta}$.

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

$$B = \beta', \quad W_{\beta'} = w_{\beta}, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = k_{1-\beta'}, \quad T' = \boxed{t}, \quad \text{wdata}' = \perp.$$

Clearly, $H_1^{\beta} \equiv H_{2,0}^{\beta}$ for each $\beta \in \{0, 1\}$.

Claim 19 (¶). $H_{2,t-1}^\beta \equiv H_{2,t}^\beta$ for all $t \in [T]$ and $\beta \in \{0, 1\}$.

- H_3^β . In this hybrid, we set T' to T , i.e.,

$$B = \beta', \quad W_{\beta'} = w_\beta, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = k_{1-\beta'}, \quad T' = \boxed{T}, \quad \text{wdata}' = \perp.$$

By definition, $H_{2,T}^\beta \equiv H_3^\beta$ for each $\beta \in \{0, 1\}$.

- H_4^β . In this hybrid, we rename $(\beta' \oplus \beta)$ to γ . It can be described as

$$B = \boxed{\gamma \oplus \beta}, \quad W_{\gamma} = w_{\boxed{0}}, \quad K_{\gamma} = k_{\boxed{0}}, \quad W_{1-\gamma} = w_{\boxed{1}}, \quad K_{1-\gamma} = k_{\boxed{1}}, \quad T' = T, \quad \text{wdata}' = \perp.$$

Since the change is syntactical, $H_3^\beta \equiv H_4^\beta$ for each $\beta \in \{0, 1\}$.

In H_4^0 and H_4^1 , the machine M' fully executes both $M^{\dots}(w_0)$ on the copy γ and $M^{\dots}(w_1)$ on the copy $(1-\gamma)$. Their only difference is from which execution M' takes $(\text{done}_t, \tau_t, i_t, \text{out}_t)$. The two executions of M satisfy the constraint of fixed-address security, so their $(\text{done}_t, \tau_t, i_t, \text{out}_t)$ sequences are identical. Therefore, $H_4^0 \approx H_4^1$ by the fixed-memory security of LGRAM'.

We conclude that $\text{Exp}_{\text{LGRAM}}^0 \equiv H_0^0 \approx H_0^1 \equiv \text{Exp}_{\text{LGRAM}}^1$ 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'. \square

Proof (Claim 19). We further alter the input to M' to show $H_{2,t-1}^\beta \approx H_{2,t}^\beta$.

- G_0 . This is just $H_{2,t-1}^\beta$, described by

$$B = \beta', \quad W_{\beta'} = w_\beta, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = k_{1-\beta'}, \quad T' = t-1, \\ \text{wdata}' = \perp.$$

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

$$B = \beta', \quad W_{\beta'} = w_\beta, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = \boxed{\dot{k}_{1-\beta',\{t\}}}, \quad T' = \boxed{t}, \\ \text{wdata}' = \begin{cases} \perp, & \text{if } \tau_t \in [T]; \\ t \parallel (\text{wdata}_{\beta,t} \oplus \text{PPRF.Eval}(k_0, t)) \parallel \text{PPRF.Eval}(k_1, t), & \text{if } \tau_t = \text{work and } \beta' = 0; \\ t \parallel \text{PPRF.Eval}(k_0, t) \parallel (\text{wdata}_{\beta,t} \oplus \text{PPRF.Eval}(k_1, t)), & \text{if } \tau_t = \text{work and } \beta' = 1; \end{cases}$$

where $\text{wdata}_{\beta,t}$ is from $M^{\dots}(w_\beta)$ and $\widetilde{\text{wdata}}_{\beta',t} = \text{wdata}_{\beta,t}$ in the execution of M' . $G_0 \approx G_1$ 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.,

$$B = \beta', \quad W_{\beta'} = w_\beta, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = \dot{k}_{1-\beta',\{t\}}, \quad T' = t, \\ \text{wdata}' = \begin{cases} \perp, & \text{if } \tau_t \in [T]; \\ t \parallel (\text{wdata}_{\beta,t} \oplus \text{PPRF.Eval}(k_0, t)) \parallel \boxed{\text{random}}, & \text{if } \tau_t = \text{work and } \beta' = 0; \\ t \parallel \boxed{\text{random}} \parallel (\text{wdata}_{\beta,t} \oplus \text{PPRF.Eval}(k_1, t)), & \text{if } \tau_t = \text{work and } \beta' = 1. \end{cases}$$

Note that the altered portion is $\text{PPRF.Eval}(k_{1-\beta'}, t)$ in G_1 . By the security of PPRF, it follows that $G_1 \approx G_2$.

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

$$B = \beta', \quad W_{\beta'} = w_{\beta}, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = \overset{\circ}{k}_{1-\beta', \{t\}}, \quad T' = t,$$

$$wdata' = \begin{cases} \perp, & \text{if } \tau_t \in [\mathcal{T}]; \\ t \parallel (wdata_{\beta, t} \oplus \text{PPRF.Eval}(k_0, t)) \parallel \boxed{(wdata_{1-\beta, t} \oplus \text{PPRF.Eval}(k_1, t))}, & \text{if } \tau_t = \text{work and } \beta' = 0; \\ t \parallel \boxed{(wdata_{1-\beta, t} \oplus \text{PPRF.Eval}(k_0, t))} \parallel (wdata_{\beta, t} \oplus \text{PPRF.Eval}(k_1, t)), & \text{if } \tau_t = \text{work and } \beta' = 1. \end{cases}$$

$G_2 \approx G_3$ by the security of PPRF.

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

$$B = \beta', \quad W_{\beta'} = w_{\beta}, \quad K_{\beta'} = k_{\beta'}, \quad W_{1-\beta'} = w_{1-\beta}, \quad K_{1-\beta'} = \boxed{k_{1-\beta'}}, \quad T' = t,$$

$$wdata' = \boxed{\perp}.$$

$G_3 \approx G_4$ by the fixed-memory security of LGRAM', analogously to $G_0 \approx G_1$.

By inspection, G_4 is just $H_{2,t}^\beta$. Therefore, $H_{2,t-1}^\beta \equiv G_0 \approx G_4 \equiv H_{2,t}^\beta$. \square

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'd 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 $\text{poly}(\lambda, |M|, \log T_{\max})$, 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

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

Construction 3. Our scheme works as follows:

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

$$\text{digest}'_\tau \stackrel{\$}{\leftarrow} \text{Compress}'(1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, \tau, D_\tau)$$

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

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

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

prepares M' as shown in Figure 8. It sets

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

runs

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

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

$$\{L_b^B\}_b, \{L_{i,b}^{W_0}\}_{i,b}, \{L_{i,b}^{W_1}\}_{i,b}, \{L_{i,b}^{K_0}\}_{i,b}, \{L_{i,b}^{K_1}\}_{i,b}, \{L_{i,b}^{T'}\}_{i,b}, \{L_{i,b}^{R'}\}_{i,b}, \{L_{i,b}^{K'}\}_{i,b}, \{L_{i,b}^{\text{addr}S'}\}_{i,b},$$

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

$$\begin{aligned} \widehat{M} &= (\widehat{M}', \{\text{labels for } B = \beta', K_0 = k_0, K_1 = k_1, T' = 0, R' = \perp, K' = k', \text{addr}S' = \perp\}), \\ L_{i,b} &= L_{i,b}^{W_0} \parallel L_{i,b}^{W_1} \quad \text{for } i \in [\ell_{\text{in}}], b \in \{0, 1\}. \end{aligned}$$

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

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

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 $M^{D_1, \dots, D_\mathcal{T}}(w)$ in the clear and outputs it.

Machine M' Transformed from M

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

Input (w'). $B, W_0, W_1, K_0, K_1, T',$
 see Figure 7;
 $R',$ hardwired ORAM randomness if K_B is punctured;
 $K',$ PPRF key for ORAM simulation;
 $\text{addrS}',$ 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 $(2T_0 + 4);$
 $\widetilde{\text{st}}_{0,t-1}, \widetilde{\text{rdata}}_{0,t-1}, \text{ost}_{0,t-1,t_0-1}, \widetilde{\text{st}}_{1,t-1}, \widetilde{\text{rdata}}_{1,t-1}, \text{ost}_{1,t-1,t_0-1},$
 for the two copies of M and their ORAM;
 $\text{done}'_t, \text{out}'_t,$ 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 $\widetilde{\ell}_{<\tau} = \sum_{\pi < \tau} |D_\pi| < t \leq \sum_{\pi \leq \tau} |D_\pi|$ 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 - B) is $\begin{cases} \text{simulated using } K', & \text{if } t - \widetilde{\ell}_{<\mathcal{T}+1} > T'; \\ \text{hardwired in addrS}', & \text{if } t - \widetilde{\ell}_{<\mathcal{T}+1} = T'; \end{cases}$

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

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.

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' . We remark that the evaluation time has linear dependency on the input tape lengths for *two* reasons (other than the inherent lower bound): *i*) the bounded scheme has such dependency; and *ii*) the machine M' reads all the input tapes in their entirety at the beginning.

Theorem 20 (¶). *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 20). The proof is analogous to that of Theorem 18. 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 19 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, $\text{PPRF.Eval}(k_{1-B}, i)$ 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 [CH15; 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 $\{\text{PPRF.Eval}(k_{1-B}, i)\}_{i \in R_t}$ into R' , and hardwire $\text{SimORAM}(\dots, \text{PPRF.Eval}(k', t))$ into addrS' .
 2. (PPRF security.) Change $\text{PPRF.Eval}(k', t)$ in addrS' into true randomness.
 3. (Fixed-address security.) Change $\{\text{PPRF.Eval}(k_{1-B}, i)\}_{i \in R_t}$ 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 $\{\text{PPRF.Eval}(k_{1-B}, i)\}_{i \in R_t}$.
 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. Recall that bounded schemes has the following three deficiencies: *i*) its evaluation time scales with the time bound T_{\max} instead of the instance running time T ; *ii*) it only works if

the maximum address accessed by the machine is within T_{\max} ,²⁹ and *iii*) its security only holds if T_{\max} is polynomially bounded. To get rid of these restrictions, 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 any larger garblings are hidden by encryption, removing the need to apply the LGRAM security on those large garblings.

Ingredients of Construction 4. Let

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

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

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

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 $t \leq 2^E$:
1 step for one step of M
 $\text{poly}(\lambda, |M|, \log T_{\max})$ 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 $2^E < t < 2^E + \text{poly}(\lambda)$:
report running time exceeded
output K

Figure 9. The machine M' in Construction 4.

²⁹The issue is only syntactical for the scheme from Construction 3, i.e., it does allow arbitrary address on the working tape because our notion of ORAM implicitly requires handling it.

Construction 4. Our scheme works as follows:

- $\text{Compress}(1^{\ell_{\text{addr}}}, 1^{\ell_{\text{cell}}}, \tau, D_\tau)$ is the same as $\text{Compress}'$.
- $\text{Garble}(T_{\max}, M, \{\text{digest}_\tau\}_{\tau \in [\mathcal{T}]})$ 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

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

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

$$\widetilde{M}'_e \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_e, \widehat{M}'_e), \quad \widetilde{L}'_{e,i,b} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_e, L'_{e,i,b}),$$

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

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

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

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

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 21 (♣). *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 21). 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_{-1}^β . This is just $\text{Exp}_{\text{LGRAM}}^\beta$ for the constructed scheme, where

$$\begin{aligned} \widehat{M} &: k_1, \{\widetilde{M}'_e, \widetilde{L}_{e,i,b}^E \text{'s and } \widetilde{L}_{e,i,b}^K \text{'s for } E = e, K = k_{e+1}\}_{e \in [\lceil \log_2 T_{\max} \rceil]}, \\ L_i &: \{\widetilde{L}_{e,i,b}^W \text{'s for } W = w_\beta\}_{e \in [\lceil \log_2 T_{\max} \rceil]}. \end{aligned}$$

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

$$\widehat{M} : k_1, \begin{cases} \widetilde{M}'_e, \widetilde{L}_{e,i,b}^E \text{'s and } \widetilde{L}_{e,i,b}^K \text{'s for } E = e, K = k_{e+1}, & \text{if } e < \bar{e}; \\ \widetilde{M}'_{\bar{e}}, \widetilde{L}_{\bar{e},i,b}^E \text{'s and } \widetilde{L}_{\bar{e},i,b}^K \text{'s for } E = \bar{e}, K = \perp, & \text{if } e = \bar{e}; \\ \text{random}, & \text{if } e > \bar{e} \text{ and } e \leq e; \\ \widetilde{M}'_e, \widetilde{L}_{e,i,b}^E \text{'s and } \widetilde{L}_{e,i,b}^K \text{'s for } E = e, K = k_{e+1}, & \text{if } e > \bar{e} \text{ and } e > e; \end{cases}$$

$$L_i : \begin{cases} \widetilde{L}_{e,i,b}^W \text{'s for } W = w_\beta, & \text{if } e \leq \bar{e}; \\ \text{random}, & \text{if } e > \bar{e} \text{ and } e \leq e; \\ \widetilde{L}_{e,i,b}^W \text{'s for } W = w_\beta, & \text{if } e > \bar{e} \text{ and } e > e. \end{cases}$$

To see indistinguishability:

- $H_{e-1}^\beta \approx H_0^\beta$. Compared to H_{e-1}^β , in H_0^β , the garbling \widehat{M}'_e has K in its input changed from $k_{\bar{e}+1}$ to \perp . 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, $H_{e-1}^\beta \approx H_0^\beta$ by the bounded security of LGRAM'.
- $H_{e-1}^\beta \approx H_e^\beta$. The two hybrids are different only when $e > \bar{e}$, in which case the ciphertexts under k_e changes into random strings and k_e is not used in the two hybrids. Therefore, $H_{e-1}^\beta \approx H_e^\beta$ by the ciphertext pseudorandomness of SKE.
- $H_{\lceil \log_2 T_{\max} \rceil}^0 \approx H_{\lceil \log_2 T_{\max} \rceil}^1$. 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, $H_{\lceil \log_2 T_{\max} \rceil}^0 \approx H_{\lceil \log_2 T_{\max} \rceil}^1$ follows from a standard hybrid argument by the bounded security of LGRAM'.

We conclude that $\text{Exp}_{\text{LGRAM}}^0 \equiv H_{-1}^0 \approx H_{-1}^1 \equiv \text{Exp}_{\text{LGRAM}}^1$. \square

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 $O(T + |f| + |x|)$, 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

- $\text{ckt} = (\text{ckt.Setup}, \text{ckt.KeyGen}, \text{ckt.Enc}, \text{ckt.Dec})$ be an FE scheme for circuits,
- $\text{LGRAM} = (\text{LGRAM.Compress}, \text{LGRAM.Garble}, \text{LGRAM.Eval})$ a 2-tape LGRAM scheme,

- PPRF = (PPRF.Puncture, PPRF.Eval) a puncturable PRF, and
- SKE = (SKE.Gen, SKE.Enc, SKE.Dec) a secret-key encryption scheme.

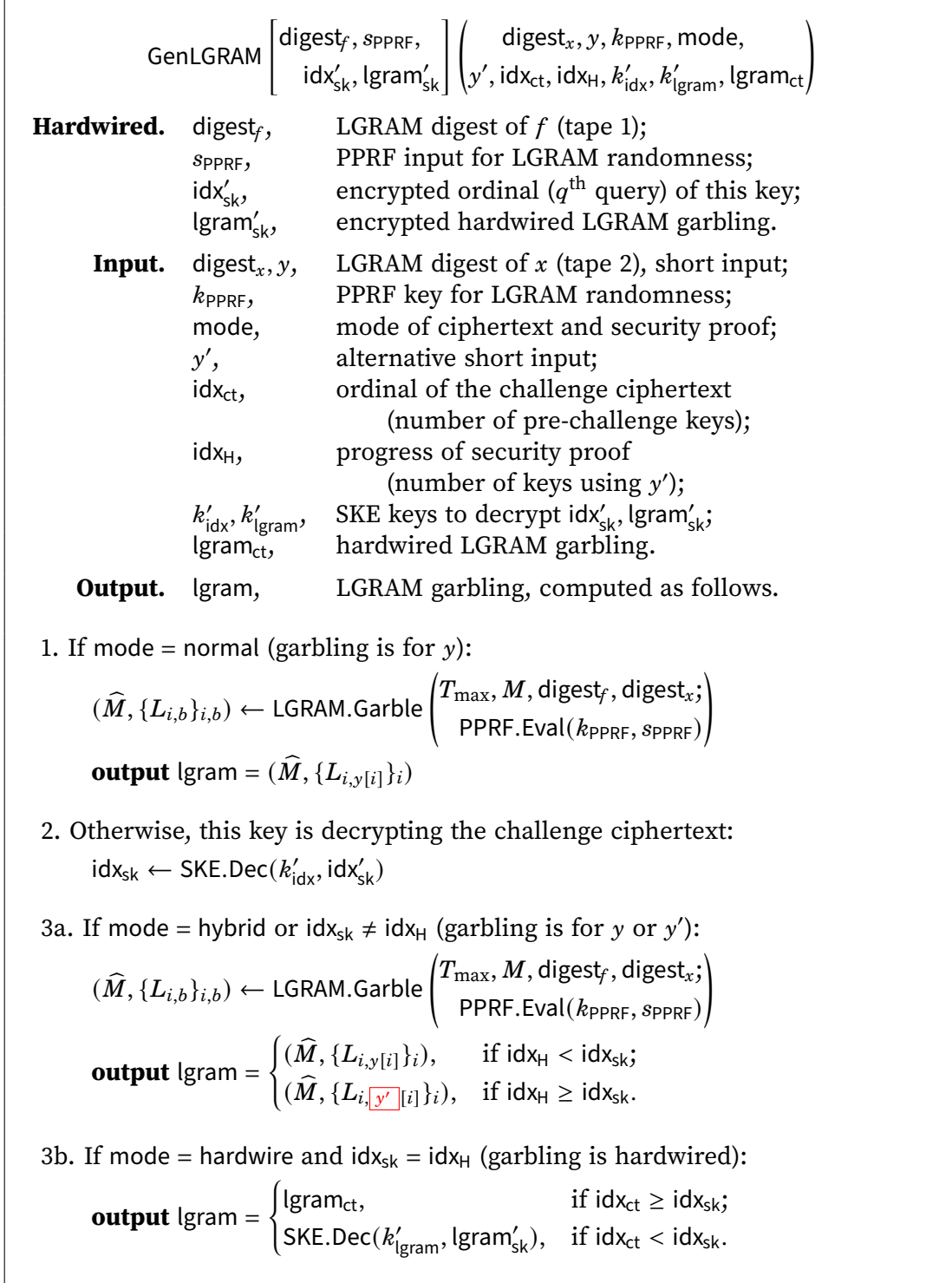


Figure 10. The circuit GenLGRAM in Construction 5.

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

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

$$(\text{ckt.mpk}, \text{ckt.msk}) \stackrel{\$}{\leftarrow} \text{ckt.Setup}(1^{\dots}, 1^{\dots}),$$

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

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

$$\begin{aligned} \text{digest}_f &\stackrel{\$}{\leftarrow} \text{LGRAM.Compress}(1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, 1, f), \\ \text{ckt.sk} &\stackrel{\$}{\leftarrow} \text{ckt.KeyGen}(\text{ckt.msk}, \text{GenLGRAM}[\text{digest}_f, s_{\text{PPRF}}, \text{idX}'_{\text{sk}}, \text{lgram}'_{\text{sk}}]), \end{aligned}$$

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

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

$$\begin{aligned} \text{digest}_x &\stackrel{\$}{\leftarrow} \text{LGRAM.Compress}(1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, 2, x), & y', \text{idX}_{\text{ct}}, \text{idX}_{\text{H}}, k'_{\text{idX}}, k'_{\text{lgram}}, \text{lgram}_{\text{ct}} \\ \text{ckt.ct} &\stackrel{\$}{\leftarrow} \text{ckt.Enc}(\text{ckt.mpk}, \perp, (\text{digest}_x, y, k_{\text{PPRF}}, \text{normal}, \overbrace{\perp, \perp, \perp, \perp, \perp, \perp}^{\text{mode}})), \end{aligned}$$

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

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

$$(\widehat{M}, \{L_i\}_i) \stackrel{\$}{\leftarrow} \text{ckt.Dec}^{\text{GenLGRAM}[\text{digest}_f, s_{\text{PPRF}}, \text{idX}'_{\text{sk}}, \text{lgram}'_{\text{sk}}], \perp, \text{ckt.sk}, \text{ckt.ct}}(\text{ckt.mpk}),$$

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

Correctness and Efficiency. To ensure correctness, it suffices to set the input/circuit sizes of the underlying FE for circuits to be $\text{poly}(\lambda, M, \log T_{\max})$ – the normal branch of GenLGRAM is just LGRAM.Eval , which runs in time $\text{poly}(\lambda, M, \log T_{\max})$. 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 $(T + |f| + |x|) \text{poly}(\lambda, M, T_{\max})$.

Theorem 22 (¶). *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

$$\text{digest}_{f,q}, s_{\text{PPRF},q}, \text{idX}'_{\text{sk},q}, \text{lgram}'_{\text{sk},q}$$

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

$$y', \text{idX}_{\text{ct}}, \text{idX}_{\text{H}}, \text{lgram}_{\text{ct}}, k'_{\text{idX}}, k'_{\text{lgram}}$$

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:

- $\text{id}_{\text{sk},q}'$ is an encryption of q so that each key “knows” its ordinal number.
- id_{ct} is the ordinal number of the challenge ciphertext.
- id_{H} indicates the progress of the proof and increases from 0 to Q :
 - initially, $q > \text{id}_{\text{H}}$, decrypting ct by sk_q yields an LGRAM garbling for y_0 ;
 - when transitioning, $q = \text{id}_{\text{H}}$, 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 $\text{lgram}'_{\text{sk},q}$ (in sk_q) or lgram_{ct} (in ct), whichever is generated later the security game (decided by comparing id_{ct} and q);
 - * when mode = hybrid, the behavior is the same as when $q < \text{id}_{\text{H}}$;
 - eventually, $q < \text{id}_{\text{H}}$, decrypting ct by sk_q yields an LGRAM garbling for y_1 .

Proof (Theorem 22). Let Q_1, Q_2 be the number of secret key queries in **Query I** and **Query II**, respectively. Writing k_{id_x} 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 $\text{Exp}_{\text{PHFE}}^0$, described as

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

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

$$\begin{array}{lll}
\text{sk}_q : & s_{\text{PPRF},q} \xleftarrow{\$} \text{distinct}, & \text{id}'_{\text{sk},q} \xleftarrow{\$} \text{random}, & \text{lgram}'_{\text{sk},q} \xleftarrow{\$} \text{random}; \\
\text{ct} : & y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow k, & \text{mode} \leftarrow \text{normal}, \\
& y' \leftarrow \perp, & \text{id}_{\text{ct}} \leftarrow \perp, & \text{id}_{\text{H}} \leftarrow \perp, \\
& k'_{\text{id}_x} \leftarrow \perp, & k'_{\text{lgram}} \leftarrow \perp, & \text{lgram}_{\text{ct}} \leftarrow \perp.
\end{array}$$

H_0 and H_1 are statistically indistinguishable.

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

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

$H_1 \approx H_2$ follows from the ciphertext pseudorandomness of SKE, as the key k_{id_x} is not used anywhere else.

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

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

$$\begin{array}{lll}
\text{ct} : & y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow k, & \text{mode} \leftarrow \boxed{\text{hybrid}}, \\
& y' \leftarrow \boxed{y_1}, & \text{idx}_{\text{ct}} \leftarrow \boxed{Q_1}, & \text{idx}_{\text{H}} \leftarrow \boxed{0}, \\
& k'_{\text{idx}} \leftarrow \boxed{k_{\text{idx}}}, & k'_{\text{lgram}} \leftarrow \perp, & \text{lgram}_{\text{ct}} \leftarrow \perp.
\end{array}$$

It is readily verified that the decryption outcome (as perceived by ckt) does not change. Therefore, $H_2 \approx H_3$ follows from the security of ckt.

- $H_{4,q}$ for $q = 0, \dots, Q_1 + Q_2$. In this hybrid, idx_{H} is incremented to q , i.e.,

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

$H_{4,0}$ is just H_3 . The proofs of indistinguishability between $H_{4,q-1}$ and $H_{4,q}$ depend on whether $q \leq Q_1$ or $q > Q_1$.

Claim 23 (¶). $H_{4,q-1} \approx H_{4,q}$ for $q = 1, \dots, Q_1$.

Claim 24 (¶). $H_{4,q-1} \approx H_{4,q}$ for $q = Q_1 + 1, \dots, Q_1 + Q_2$.

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

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

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.,

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

$H_5 \approx H_6$ follows from the security of ckt, analogously to $H_2 \approx H_3$.

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

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

$H_6 \approx H_7$ by the ciphertext pseudorandomness of SKE, analogously to $H_1 \approx H_2$.

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

$$\text{sk}_q : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \boxed{\text{random}}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}, \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random};$$

$$\begin{array}{lll}
\text{ct} : & y \leftarrow y_1, & k_{\text{PPRF}} \leftarrow k, & \text{mode} \leftarrow \text{normal}, \\
& y' \leftarrow \perp, & \text{idx}_{\text{ct}} \leftarrow \perp, & \text{idx}_{\text{H}} \leftarrow \perp, \\
& k'_{\text{idx}} \leftarrow \perp, & k'_{\text{lgram}} \leftarrow \perp, & \text{lgram}_{\text{ct}} \leftarrow \perp.
\end{array}$$

H_8 is statistically indistinguishable from H_7 .

By inspection, H_8 is just $\text{Exp}_{\text{PHFE}}^1$, and therefore, $\text{Exp}_{\text{PHFE}}^0 \equiv H_0 \approx H_8 \equiv \text{Exp}_{\text{PHFE}}^1$. \square

Proof (Claim 23). To show indistinguishability between $H_{4,q-1}$ and $H_{4,q}$ when $q \leq Q_1$, we temporarily hardwire the LGRAM garbling yielded by decryption ct using sk_q into lgram_{ct} , which is generated when both f_q and x, y_0, y_1 are known.

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

- G_0 . This is just $H_{4,q-1}$, described as

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

- G_1 . In this hybrid, we puncture k_{PPRF} , hardwire the decryption result of ct by sk_q into ct at lgram_{ct} , indicating hardwiring using mode, and increment idx_{H} , i.e.,

$$\begin{array}{lll}
\text{sk}_q : & s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), & \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : & y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow \hat{k}_{\{q\}}, & \text{mode} \leftarrow \text{hardwire}, \\
& y' \leftarrow y_1, & \text{idx}_{\text{ct}} \leftarrow Q_1, & \text{idx}_{\text{H}} \leftarrow q, \\
& k'_{\text{idx}} \leftarrow k_{\text{idx}}, & k'_{\text{lgram}} \leftarrow \perp, &
\end{array}$$

$$\text{lgram}_{\text{ct}} \leftarrow (\hat{M}, \{L_{i,y_0[i]}\}_i) \text{ from LGRAM.Garble} \left(\begin{array}{l} T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{PPRF.Eval}(k, s_{\text{PPRF},q}) \end{array} \right).$$

$G_0 \approx G_1$ follows from the security of ckt.

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

$$\begin{array}{lll}
\text{sk}_q : & s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), & \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : & y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow \hat{k}_{\{q\}}, & \text{mode} \leftarrow \text{hardwire}, \\
& y' \leftarrow y_1, & \text{idx}_{\text{ct}} \leftarrow Q_1, & \text{idx}_{\text{H}} \leftarrow q, \\
& k'_{\text{idx}} \leftarrow k_{\text{idx}}, & k'_{\text{lgram}} \leftarrow \perp, &
\end{array}$$

$$\text{lgram}_{\text{ct}} \leftarrow (\hat{M}, \{L_{i,y_0[i]}\}_i) \text{ from LGRAM.Garble} \left(\begin{array}{l} T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{random} \end{array} \right).$$

$G_1 \approx G_2$ follows from the security of PPRF.

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

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

$$\begin{aligned}
\text{ct} : \quad & y \leftarrow y_0, & k_{\text{PPRF}} & \leftarrow \mathring{k}_{\{q\}}, & \text{mode} & \leftarrow \text{hardware}, \\
& y' \leftarrow y_1, & \text{idx}_{\text{ct}} & \leftarrow Q_1, & \text{idx}_{\text{H}} & \leftarrow q, \\
& k'_{\text{idx}} \leftarrow k_{\text{idx}}, & k'_{\text{lgram}} & \leftarrow \perp, & & \\
& \text{lgram}_{\text{ct}} \leftarrow (\widehat{M}, \{L_{i, \boxed{y_1}}[i]\}_i) \text{ from LGRAM.Garble} \left(\begin{array}{l} T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{random} \end{array} \right).
\end{aligned}$$

$G_2 \approx G_3$ follows from the security of LGRAM.

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

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

$G_3 \approx G_4$ follows from the security of PPRF.

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

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

$G_4 \approx G_5$ follows from the security of ckt.

By inspection, G_5 is exactly $H_{4, q}$. Therefore, $H_{4, q-1} \equiv G_0 \approx G_5 \equiv H_{4, q}$. \square

Proof (Claim 24). To show indistinguishability between $H_{4, q-1}$ and $H_{4, q}$ when $\boxed{q > Q_1}$, we temporarily hardwire the LGRAM garbling yielded by decryption ct using sk_q into $\boxed{\text{lgram}'_{\text{sk}, q}}$, generated when both x, y_0, y_1 and f_q are known. Let $\mathring{k}_{\{q\}}$ be k punctured at $\{q\}$ and k_{lgram} a random secret key of SKE. We specify the hybrids.

- G_0 . This is just $H_{4, q-1}$, described as

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

- G_1 . In this hybrid, we encrypt the LGRAM garbling into $\text{lgram}'_{\text{sk}, q}$, i.e.,

$$\text{sk}_{q \neq q} : \quad s_{\text{PPRF}, q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk}, q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \quad \text{lgram}'_{\text{sk}, q} \stackrel{\$}{\leftarrow} \text{random};$$

$$\begin{aligned}
\text{ct} : & \quad y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow k, & \text{mode} \leftarrow \text{hybrid}, \\
& \quad y' \leftarrow y_1, & \text{idx}_{\text{ct}} \leftarrow Q_1, & \text{idx}_{\text{H}} \leftarrow q - 1, \\
& \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, & k'_{\text{lgram}} \leftarrow \perp, & \text{lgram}_{\text{ct}} \leftarrow \perp; \\
\text{sk}_{q > Q_1} : & \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \\
& \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc} \left(k_{\text{lgram}}, \text{LGRAM.Garble} \left(\begin{array}{l} (\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from} \\ T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{PPRF.Eval}(k, s_{\text{PPRF},q}) \end{array} \right) \right).
\end{aligned}$$

$G_0 \approx G_1$ by the ciphertext pseudorandomness of SKE.

- G_2 . In this hybrid, we puncture k at $\{q\}$, activate the hardwiring using k'_{lgram} , mode, and increment idx_{H} , i.e.,

$$\begin{aligned}
\text{sk}_{q \neq q} : & \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), & \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : & \quad y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow \hat{k}_{\{q\}}, & \text{mode} \leftarrow \text{hardwire}, \\
& \quad y' \leftarrow y_1, & \text{idx}_{\text{ct}} \leftarrow Q_1, & \text{idx}_{\text{H}} \leftarrow q, \\
& \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, & k'_{\text{lgram}} \leftarrow k_{\text{lgram}}, & \text{lgram}_{\text{ct}} \leftarrow \perp; \\
\text{sk}_{q > Q_1} : & \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \\
& \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc} \left(k_{\text{lgram}}, \text{LGRAM.Garble} \left(\begin{array}{l} (\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from} \\ T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{PPRF.Eval}(k, s_{\text{PPRF},q}) \end{array} \right) \right).
\end{aligned}$$

$G_1 \approx G_2$ follows from the security of ckt.

- G_3 . In this hybrid, we generate the hardwired garbling with true randomness instead of $\text{PPRF.Eval}(k, q)$, i.e.,

$$\begin{aligned}
\text{sk}_{q \neq q} : & \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), & \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : & \quad y \leftarrow y_0, & k_{\text{PPRF}} \leftarrow \hat{k}_{\{q\}}, & \text{mode} \leftarrow \text{hardwire}, \\
& \quad y' \leftarrow y_1, & \text{idx}_{\text{ct}} \leftarrow Q_1, & \text{idx}_{\text{H}} \leftarrow q, \\
& \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, & k'_{\text{lgram}} \leftarrow k_{\text{lgram}}, & \text{lgram}_{\text{ct}} \leftarrow \perp; \\
\text{sk}_{q > Q_1} : & \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, & \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \\
& \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc} \left(k_{\text{lgram}}, \text{LGRAM.Garble} \left(\begin{array}{l} (\widehat{M}, \{L_{i,y_0[i]}\}_i) \text{ from} \\ T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{random} \end{array} \right) \right).
\end{aligned}$$

$G_2 \approx G_3$ follows from the security of PPRF.

- G_4 . In this hybrid, the hardwired garbling becomes one for y_1 , i.e.,

$$\text{sk}_{q \neq q} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random};$$

$$\begin{array}{l}
\text{ct} : \quad y \leftarrow y_0, \quad k_{\text{PPRF}} \leftarrow \hat{k}_{\{q\}}, \quad \text{mode} \leftarrow \text{hardware}, \\
\quad \quad y' \leftarrow y_1, \quad \text{idx}_{\text{ct}} \leftarrow Q_1, \quad \text{idx}_{\text{H}} \leftarrow q, \\
\quad \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, \quad k'_{\text{lgram}} \leftarrow k_{\text{lgram}}, \quad \text{lgram}_{\text{ct}} \leftarrow \perp; \\
\text{sk}_{q>Q_1} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \\
\quad \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc} \left(k_{\text{lgram}}, \begin{array}{l} (\widehat{M}, \{L_{i,y_1}[i]\}_i) \text{ from} \\ \text{LGRAM.Garble} \left(\begin{array}{l} T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{random} \end{array} \right) \end{array} \right).
\end{array}$$

$G_3 \approx G_4$ 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.,

$$\begin{array}{l}
\text{sk}_{q \neq q} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : \quad y \leftarrow y_0, \quad k_{\text{PPRF}} \leftarrow \hat{k}_{\{q\}}, \quad \text{mode} \leftarrow \text{hardware}, \\
\quad \quad y' \leftarrow y_1, \quad \text{idx}_{\text{ct}} \leftarrow Q_1, \quad \text{idx}_{\text{H}} \leftarrow q, \\
\quad \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, \quad k'_{\text{lgram}} \leftarrow k_{\text{lgram}}, \quad \text{lgram}_{\text{ct}} \leftarrow \perp; \\
\text{sk}_{q>Q_1} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \\
\quad \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc} \left(k_{\text{lgram}}, \begin{array}{l} (\widehat{M}, \{L_{i,y_1}[i]\}_i) \text{ from} \\ \text{LGRAM.Garble} \left(\begin{array}{l} T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{PPRF.Eval}(k, q) \end{array} \right) \end{array} \right).
\end{array}$$

$G_4 \approx G_5$ 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.,

$$\begin{array}{l}
\text{sk}_{q \neq q} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : \quad y \leftarrow y_0, \quad k_{\text{PPRF}} \leftarrow k, \quad \text{mode} \leftarrow \text{hybrid}, \\
\quad \quad y' \leftarrow y_1, \quad \text{idx}_{\text{ct}} \leftarrow Q_1, \quad \text{idx}_{\text{H}} \leftarrow q, \\
\quad \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, \quad k'_{\text{lgram}} \leftarrow \perp, \quad \text{lgram}_{\text{ct}} \leftarrow \perp; \\
\text{sk}_{q>Q_1} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \\
\quad \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc} \left(k_{\text{lgram}}, \begin{array}{l} (\widehat{M}, \{L_{i,y_1}[i]\}_i) \text{ from} \\ \text{LGRAM.Garble} \left(\begin{array}{l} T_{\text{max}}, M, \text{digest}_{f_q}, \text{digest}_x; \\ \text{PPRF.Eval}(k, q) \end{array} \right) \end{array} \right).
\end{array}$$

$G_5 \approx G_6$ follows from the security of ckt.

- G_7 . In this hybrid, we revert $\text{lgram}'_{\text{sk},q}$ to random, i.e.,

$$\begin{array}{l}
\text{sk}_{q \neq q} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{idx}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{idx}}, q), \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{random}; \\
\text{ct} : \quad y \leftarrow y_0, \quad k_{\text{PPRF}} \leftarrow k, \quad \text{mode} \leftarrow \text{hybrid}, \\
\quad \quad y' \leftarrow y_1, \quad \text{idx}_{\text{ct}} \leftarrow Q_1, \quad \text{idx}_{\text{H}} \leftarrow q, \\
\quad \quad k'_{\text{idx}} \leftarrow k_{\text{idx}}, \quad k'_{\text{lgram}} \leftarrow \perp, \quad \text{lgram}_{\text{ct}} \leftarrow \perp;
\end{array}$$

$$\text{sk}_{q>Q_1} : \quad s_{\text{PPRF},q} \stackrel{\$}{\leftarrow} \text{distinct}, \quad \text{id}_{\text{sk},q} \stackrel{\$}{\leftarrow} \text{SKE.Enc}(k_{\text{id}_x}, q), \quad \text{lgram}'_{\text{sk},q} \stackrel{\$}{\leftarrow} \boxed{\perp}.$$

$G_6 \approx G_7$ by the ciphertext pseudorandomness of SKE.

By inspection, G_7 is exactly $H_{4,q}$. Therefore, $H_{4,q-1} \equiv G_0 \approx G_7 \equiv H_{4,q}$. \square

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

- PHFE' = (Setup', KeyGen', Enc', 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 $(T' + |f'| + |x'|) \text{poly}(\lambda, M', \log T'_{\max})$, and
- PRF a pseudorandom function.

Construction 6 (full-fledged PHFE for RAM). Our scheme works as follows:

- Setup(M, T_{\max}) runs and outputs

$$(\text{mpk}, \text{msk}) = (\text{mpk}', \text{msk}') \stackrel{\$}{\leftarrow} \text{Setup}'(M', T_{\max}),$$

where M' is shown in Figure 11.

- KeyGen(msk, f) pads f by setting $\text{one cell on an input tape of } M'$

$$f' \leftarrow (f[1] || 0^{\ell_{\text{cell}}}) || \dots || \overbrace{(f[|f|] || 0^{\ell_{\text{cell}}})}^{\text{one cell on an input tape of } M'}$$

and runs and outputs

$$\text{sk} = \text{sk}' \stackrel{\$}{\leftarrow} \text{KeyGen}'(\text{msk}', f').$$

- Enc(mpk, x, y) samples $\beta' \stackrel{\$}{\leftarrow} \{0, 1\}$, a PRF key $k_{\beta'}$, 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 $w_{\beta'}$ encrypts y , by setting

$$w_{\beta'}[i] \leftarrow y[i] \oplus \text{PRF}(k_{\beta'}, i) \quad \text{for } i \in [|y|],$$

$$w \leftarrow \underbrace{(w_0[1] || w_1[1])}_{\text{one cell on an input tape of } M'} || \dots || (w_0[|y|] || w_1[|y|]),$$

one cell on an input tape of M'

$$x' \leftarrow \underbrace{(x[0] || 0^{\ell_{\text{cell}}})}_{\text{one cell on an input tape of } M'} || \dots || (x[|x|] || 0^{\ell_{\text{cell}}}) || w, \quad y' \leftarrow (|x|, \beta', k_{\beta'}).$$

The algorithm runs $\text{ct}' \stackrel{\$}{\leftarrow} \text{Enc}'(\text{mpk}', x', y')$ and outputs $\text{ct} = (\text{ct}', w)$.

- Dec $^{f,x,\text{sk},\text{ct}}$ (mpk) prepares oracles for f', x' from f, x, ct as specified in KeyGen, Enc. It runs and outputs

$$(\text{Dec}')^{f',x',\text{sk}',\text{ct}'}(\text{mpk}').$$

2-Tape RAM M'

- Lengths.** $\ell'_{\text{st}} = \ell_{\text{st}} + \text{poly}(\lambda, |M|, \log T_{\text{max}})$,
the state of M , the location of the last-read cell;
 $\ell'_{\text{in}} = \text{poly}(\lambda, |M|, \log T_{\text{max}}) + 1 + \text{poly}(\lambda, |M|, \log T_{\text{max}})$,
to encode $|x|$, choice bit β' , and PRF key $k_{\beta'}$;
 $\ell'_{\text{addr}} = \ell_{\text{addr}} + 1$, $\ell'_{\text{cell}} = 2\ell_{\text{cell}}$,
twice as many cells with each cell twice as large
on an input tape (to encode x, y_0, y_1 in x');
 $\ell'_{\text{ADDR}} = \ell_{\text{ADDR}}$, $\ell'_{\text{CELL}} = \ell_{\text{CELL}}$,
exactly the working tape of M .
- Input.** $w' = (|x|, \beta', k_{\beta'})$, length of x , choice bit, PRF key for y or $y_{\beta'}$;
 $D'_1 = f'$, $\ell'_1 = \ell_1$, 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 $D'_1[i] = f'[i] = f[i] \parallel 0^{\ell_{\text{cell}}}$ 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 $D'_2[i] = x'[i] = x[i] \parallel 0^{\ell_{\text{cell}}}$
provide $x[i]$
 - 2b. If M reads $D_2[|x| + i] = y[i]$ for $i \in [|y|]$:
read $D'_2[|x| + i] = w[i] = w_0[i] \parallel w_1[i]$
provide $w_{\beta'}[i] \oplus \text{PRF}(k_{\beta'}, i)$
 3. If M reads $D_{\text{work}}[i]$:
read and provide $D'_{\text{work}}[i]$

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 $|M'| = \text{poly}(\lambda, |M|, \log T_{\max})$, with parameters for PHFE' denoted with primes,

$$\begin{aligned}
|\text{sk}| &= |\text{sk}'| = \text{poly}(\lambda, |M'|, \log T_{\max}) \\
&= \text{poly}(\lambda, |M|, \log T_{\max}), \\
|\text{ct}| &= 2|y| + |\text{ct}'| = 2|y| + \text{poly}(\lambda, |M'|, \log T_{\max}) \\
&= 2|y| + \text{poly}(\lambda, |M|, \log T_{\max}), \\
T_{\text{KeyGen}} &= T'_{\text{KeyGen}} = |f'| \text{poly}(\lambda, |M'|, \log T_{\max}) \\
&= |f| \text{poly}(\lambda, |M|, \log T_{\max}), \\
T_{\text{Enc}} &= T'_{\text{Enc}} = |x'| \text{poly}(\lambda, |M'|, \log T_{\max}) \\
&= (|x| + |y|) \text{poly}(\lambda, |M|, \log T_{\max}), \\
T_{\text{Dec}} &= T'_{\text{Dec}} = (T' + |f'| + |x'|) \text{poly}(\lambda, |M'|, \log T_{\max}) \\
&= (T + |f| + |x| + |y|) \text{poly}(\lambda, |M|, \log T_{\max}).
\end{aligned}$$

Theorem 25 (¶). *Suppose in Construction 6, PHFE', PRF are secure (Definitions 7 and 17), then the constructed scheme is secure (Definition 7).*

Proof (Theorem 25). Let $\beta' \stackrel{\$}{\leftarrow} \{0, 1\}$ 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^β . This is just $\text{Exp}_{\text{PHFE}}^\beta$, where

$$\begin{aligned}
w_{\beta'}[i] &\leftarrow y_\beta[i] \oplus \text{PRF}(k_{\beta'}, i), & w_{1-\beta'}[i] &\stackrel{\$}{\leftarrow} \text{random}, \\
x' &\leftarrow (x[1] \| 0^{\ell_{\text{cell}}}) \| \dots \| (x[|x|] \| 0^{\ell_{\text{cell}}}) \| (w_0[1] \| w_1[i]) \| \dots \| (w_0[|y|] \| w_1[|y|]), \\
\text{ct}' &\stackrel{\$}{\leftarrow} \text{Enc}'(\text{mpk}, x', (|x|, \beta', k_{\beta'})).
\end{aligned}$$

- H_1^β . In this hybrid, we make $w_{1-\beta'}$ an encryption of $y_{1-\beta}$ under $k_{1-\beta'}$, i.e.,

$$\begin{aligned}
w_{\beta'}[i] &\leftarrow y_\beta[i] \oplus \text{PRF}(k_{\beta'}, i), & w_{1-\beta'}[i] &\leftarrow y_{1-\beta}[i] \oplus \text{PRF}(k_{1-\beta'}, i), \\
x' &\leftarrow (x[1] \| 0^{\ell_{\text{cell}}}) \| \dots \| (x[|x|] \| 0^{\ell_{\text{cell}}}) \| (w_0[1] \| w_1[i]) \| \dots \| (w_0[|y|] \| w_1[|y|]), \\
\text{ct}' &\stackrel{\$}{\leftarrow} \text{Enc}'(\text{mpk}, x', (|x|, \beta', k_{\beta'})).
\end{aligned}$$

$H_0^\beta \approx H_1^\beta$ for each $\beta \in \{0, 1\}$ by the security of PRF.

- H_2^β . In this hybrid, we rename $(\beta' \oplus \beta)$ to γ , making it

$$\begin{aligned}
w_{\gamma}[i] &\leftarrow y_0[i] \oplus \text{PRF}(k_{\gamma}, i), & w_{1-\gamma}[i] &\leftarrow y_1[i] \oplus \text{PRF}(k_{1-\gamma}, i), \\
x' &\leftarrow (x[1] \| 0^{\ell_{\text{cell}}}) \| \dots \| (x[|x|] \| 0^{\ell_{\text{cell}}}) \| (w_0[1] \| w_1[i]) \| \dots \| (w_0[|y|] \| w_1[|y|]), \\
\text{ct}' &\stackrel{\$}{\leftarrow} \text{Enc}'(\text{mpk}, x', (|x|, \gamma \oplus \beta, k_{\gamma \oplus \beta})),
\end{aligned}$$

where $\gamma \stackrel{\$}{\leftarrow} \{0, 1\}$. This change is conceptual, hence $H_1^\beta \equiv H_2^\beta$ for each $\beta \in \{0, 1\}$.

$H_2^0 \approx H_2^1$ follows from the security of PHFE'. Therefore, $\text{Exp}_{\text{PHFE}}^0 \equiv H_0^0 \approx H_0^1 \equiv \text{Exp}_{\text{PHFE}}^1$. \square

7 Applications

7.1 Rate-1 PHFE for RAM

We can obtain rate-1 ciphertexts if we abandon both long output and adaptive security. The root of the rate being two in Construction 6 is that the security proof needs to put both candidate private inputs y_0, y_1 in the challenge ciphertext during the proof. The same proof strategy is also used in Construction 5 (see y, y' in Enc). When we only aim for semi-adaptive security,³⁰ we can instead hardwire the garblings into the secret keys during the security proof. The garblings themselves are short and can be simulated using the short output.³¹

Corollary 26. *Assuming the existence of secure FE for circuits (Definition 8), there exists semi-adaptively secure full-fledged PHFE for RAM whose only non- \perp output is always produced at the last step with constant-size keys, rate-1 ciphertexts, and linear-time KeyGen, Enc, Dec.*

7.2 ABE for RAM

By using PHFE for RAM with bounded private input as an attribute-based KEM for RAM and upgrading AB-KEM to ABE using the standard technique, we obtain ABE for RAM with constant-size keys and rate-1 ciphertexts.

Corollary 27. *Assuming FE for circuits, there exists ABE for RAM with*

$$\begin{aligned} |\text{mpk}| = |\text{sk}| &= \text{poly}(\lambda, |M|, \log T_{\max}), & |\text{ct}| &= |y| + \text{poly}(\lambda, |M|, \log T_{\max}), \\ T_{\text{KeyGen}} &= |f| \text{poly}(\lambda), & T_{\text{Enc}} &= (|x| + |y|) \text{poly}(\lambda), \\ T_{\text{Dec}} &= (T + |f| + |x| + |y|) \text{poly}(\lambda), \end{aligned}$$

where y is the message.

The above result matches the lower bound in [Luo22]. By slightly tweaking our construction, we obtain additional trade-offs matching the lower bound:

Corollary 28 (¶). *Assuming FE for circuits, for all $A, B \in \{0, 1\}$, there exists ABE for RAM with*

$$\begin{aligned} T_{\text{KeyGen}} &= |f| \text{poly}(\lambda), & |\text{sk}| &= |f|^A + \text{poly}(\lambda, |M|, \log T_{\max}), \\ T_{\text{Enc}} &= (|x| + |y|) \text{poly}(\lambda), & |\text{ct}| &= |x|^B + |y| + \text{poly}(\lambda, |M|, \log T_{\max}), \\ |\text{mpk}| &= \text{poly}(\lambda, |M|, \log T_{\max}), & T_{\text{Dec}} &= (T + |f|^{1-A} + |x|^{1-B} + |y|) \text{poly}(\lambda), \end{aligned}$$

where y is the message.

Proof Sketch (Corollary 28). It suffices to consider AB-KEM. Applying Construction 4 on top of Construction 1 yields an unbounded LGRAM with fixed-memory security. Plugging such an LGRAM into Construction 5, we obtain an AB-KEM for RAM with the following M :

³⁰Semi-adaptive security means that there is no **Query I** in Definition 7, and is only a slight strengthening [GKW16] of selective security.

³¹To be more precise, the machine will allow an alternative form of short input (T, z) for simulation, on which it idles for $(T - 2)$ steps, outputs z , and halts.

- **Tapes.** The two tapes contain f, x .
- **Short Input.** The short input is the encapsulated key k .
- **Steps.** The machine evaluates $f(x)$ then outputs k if and only if $f(x) = 1$.

The resultant scheme achieves the trade-off with $A, B = 0$. (This is *not* the same scheme as the one obtained in Corollary 27.)

In the proof of Lemma 3, the LOT scheme sets the processed database \widehat{D} to be the Merkle hash tree of D , whose bit-length can be made to be that of D .³² By inspecting the constructions, it follows that the linear time-dependencies on $|f|, |x|$ of LGRAM evaluation and AB-KEM decapsulation come exactly from recomputing their respective Merkle trees. Therefore, by choosing to store the Merkle tree of f, x (or not) in sk, ct , we achieve the four promised trade-offs. \square

7.3 Constant-Overhead $i\mathcal{O}$ for RAM

We obtain constant-overhead $i\mathcal{O}$ for RAM using the transformation in [KLW15,BGL⁺15,CHJV15,AJS17] with a circuit obfuscator $i\mathcal{O}_{\text{ckt}}$ and our LGRAM (Compress, Garble, Eval).

Corollary 29 (¶). *Assuming subexponentially secure FE for circuits, there exists a subexponentially secure $i\mathcal{O}$ for RAM, where the obfuscated program is of size $2|f| + \text{poly}(\lambda, N)$.*

Proof Sketch (Corollary 29). To obfuscate a RAM program $U^{f,x}()$, we sample $\beta' \xleftarrow{\$} \{0, 1\}$, a PRF key $k_{\beta'}$, PPRF keys k_{Compress} and k_{Garble} , and a random string $f_{1-\beta'}$ of the same length as f , set

$$\begin{aligned} f_{\beta'} &\leftarrow f \oplus (\text{PRF}(k_{\beta'}, 1) \parallel \cdots \parallel \text{PRF}(k_{\beta'}, |f|)), \\ w &\leftarrow (f_0[1] \parallel f_1[1]) \parallel \cdots \parallel (f_0[|f|] \parallel f_1[|f|]), \end{aligned}$$

run

$$\begin{aligned} \text{digest}_1 &\xleftarrow{\$} \text{Compress}(1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, 1, w), \\ \widehat{\text{GenLGRAM}} &\xleftarrow{\$} i\mathcal{O}_{\text{ckt}}(\text{GenLGRAM}[\text{digest}_1, k, \beta', k_{\beta'}]), \end{aligned}$$

and output $(\widehat{\text{GenLGRAM}}, w)$ as the obfuscation of f , where $\widehat{\text{GenLGRAM}}$ is the following circuit:

- **Hardwired.** As described above.
- **Input.** x , up to length N .
- **Output.** \widehat{U} and labels for $B = \beta'$ and $K = k_{\beta'}$ from

$$\begin{aligned} \text{digest}_2 &\leftarrow \text{Compress}(1^{\ell_{\text{cell}}}, 1^{\ell_{\text{addr}}}, 2, x; \text{PPRF.Eval}(k_{\text{Compress}}, x)), \\ (\widehat{U}, \{L_{i,b}\}_{i,b}) &\leftarrow \text{Garble}(T_{\text{max}}, U', \text{digest}_1, \text{digest}_2; \text{PPRF.Eval}(k_{\text{Garble}}, x)), \end{aligned}$$

where $(U')^{w,x}(B, K)$ works as follows.

³²The standard Merkle tree is twice as long as the database as the leaves constitute the database itself. In our syntax of PHFE (Definition 5), the databases are given in the clear for free, so we can exclude them from *the* Merkle trees stored in sk, ct .

- It parses w defined above, with $\beta' = B$ and $k_{\beta'} = K$.
- It simulates the program f over the input x , with each cell of f computed from w, B, K on demand.

Assuming subexponential security of the primitives underlying our LGRAM construction yields a subexponentially secure LGRAM. Furthermore, assuming $i\mathcal{O}_{\text{ckt}}$ is subexponentially secure, then we can employ the standard positional proof to show that the above scheme is an $i\mathcal{O}$ for RAM, at security loss $2^{\Theta(N)}$. Therefore, by bumping the security parameter up to $\text{poly}(\lambda, N)$, our $i\mathcal{O}$ for RAM is (subexponentially) secure. \square

Acknowledgement. Aayush Jain was supported by departmental funds from CMU Computer Science Department and a gift from CyLab. Huijia Lin and Ji Luo were supported by NSF grants CNS-1936825 (CAREER), CNS-2026774, a JP Morgan AI Research Award, a Cisco Research Award, and a Simons Collaboration on the Theory of Algorithmic Fairness. The views expressed are those of the authors and do not reflect the official policy or position of the funding agencies. The authors thank the anonymous reviewers of Eurocrypt 2023 for their valuable comments.

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.
- [ACFQ22] Prabhanjan Ananth, Kai-Min Chung, Xiong Fan, and Luowen Qian. Collusion-resistant functional encryption for RAMs. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 160–194. Springer, Heidelberg, December 2022.
- [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] Nuttapon 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] Nuttapon 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] Nuttapon 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 H. 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.
- [BS18] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. *Journal of Cryptology*, 31(1):202–225, January 2018.
- [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.
- [BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.
- [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.
- [CH15] Ran Canetti and Justin Holmgren. Succinct garbled RAM. Cryptology ePrint Archive, Report 2015/388, 2015. <https://eprint.iacr.org/2015/388>.
- [CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016*, pages 169–178. ACM, January 2016.
- [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.
- [CP13] Kai-Min Chung and Rafael Pass. A simple ORAM. Cryptology ePrint Archive, Report 2013/243, 2013. <https://eprint.iacr.org/2013/243>.
- [CR73] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973.
- [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.
- [GKVW20] Rishab Goyal, Venkata Koppula, Satyanarayana Vusirikala, and Brent Waters. On perfect correctness in (lockable) obfuscation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 229–259. Springer, Heidelberg, November 2020.
- [GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 361–388. Springer, Heidelberg, October / November 2016.
- [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 $i\mathcal{O}$ 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.
- [Iva82] A. G. Ivanov. Theorems on the time hierarchy for random access machines. *Journal of Soviet Mathematics*, 20(4):2299–2304, 1982.
- [JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd FOCS*, pages 1023–1034. IEEE Computer Society Press, October / November 2022.
- [JLL23] Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 479–510. Springer, Heidelberg, April 2023.
- [JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials over \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 Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.

- [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. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 680–710. Springer, Heidelberg, November 2022.
- [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.
- [LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 595–608. ACM Press, June 2023.
- [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* 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.

- [NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.
- [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, Nuttapon 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.

A Generically Lifting DE-PIR Security

We present the standard definitions of DE-PIR security and show that schemes per Definition 22 imply those satisfying the strongest security.

Definition 23 (DE-PIR indistinguishability). Let $(\text{Process}, \text{Query}, \text{Resp}, \text{Dec})$ be a (public- or secret-key) DE-PIR scheme (Definition 22). Consider $\text{Exp}_{\text{DE-PIR}}^\beta(1^\lambda, \mathcal{A})$:

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive from it $D_0, D_1 \in \{0, 1\}^*$ such that $|D_0| = |D_1| = N$. Run

$$(k, \tilde{D}) \stackrel{\$}{\leftarrow} \text{Process}(1^\lambda, D_\beta),$$

where k is either pk or sk, and send pk (in case of PK-DE-PIR) and \tilde{D} to \mathcal{A} .

- **Challenge.** Repeat the following for arbitrarily many rounds determined by \mathcal{A} . In each round, \mathcal{A} submits $i_{q,0}, i_{q,1} \in [N]$. Upon such a challenge, run

$$(\text{ct}_q, \sigma_q) \stackrel{\$}{\leftarrow} \text{Query}(1^\lambda, k, i_{q,\beta})$$

and send ct_q to \mathcal{A} .

- **Guess.** The adversary outputs a bit β' , which is the output of the experiment if i) $D_0[i_{q,0}] = D_1[i_{q,1}]$ for all q , and ii) $D_0 = D_1$. Otherwise, the output is set to 0.

The scheme satisfies *adaptive indistinguishability* if $\text{Exp}_{\text{DE-PIR}}^0 \approx \text{Exp}_{\text{DE-PIR}}^1$. The following modifications can be applied independently to obtain different levels of security:

- For *very selective security*, all the challenges must be chosen together with D_0, D_1 before the adversary gets (pk and) \tilde{D} .
- For *output-hiding*, the first condition in **Guess** is not required.
- For *database-hiding* (SK-DE-PIR only), the second condition in **Guess** is not required.

We remark that very selective indistinguishability per Definition 23 is the same as the security notion in Definition 22.

Definition 24 (simulation security of DE-PIR). Let $(\text{Process}, \text{Query}, \text{Resp}, \text{Dec})$ be a PK-DE-PIR scheme (Definition 22). A simulator is a *stateless* algorithm \mathcal{S} , whose syntax varies by the level of security being defined. Consider $\text{Exp}_{\text{PK-DE-PIR}}^{\text{real}}(1^\lambda, \mathcal{A})$ and $\text{Exp}_{\text{PK-DE-PIR}}^{\text{sim}}(1^\lambda, \mathcal{A})$:

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive $D \in \{0, 1\}^*$ from it. Run

$$(\text{pk}, \tilde{D}) \stackrel{\$}{\leftarrow} \text{Process}(1^\lambda, D)$$

and send (pk, \tilde{D}) to \mathcal{A} .

- **Challenge.** The adversary \mathcal{A} submits an index $i \in [|D|]$. Upon the challenge, run

$$\begin{cases} (\text{ct}, \sigma) \stackrel{\$}{\leftarrow} \text{Query}(1^\lambda, \text{pk}, i), & \text{in } \text{Exp}_{\text{PK-DE-PIR}}^{\text{real}}; \\ \text{ct} \stackrel{\$}{\leftarrow} \mathcal{S}(1^\lambda, D, \text{pk}, \tilde{D}, D[i]), & \text{in } \text{Exp}_{\text{PK-DE-PIR}}^{\text{sim}}; \end{cases}$$

and send ct to \mathcal{A} .

- **Guess.** The adversary outputs a bit β' , which is the output of the experiment.

The scheme satisfies *adaptive simulation security* if $\text{Exp}_{\text{PK-DE-PIR}}^{\text{real}} \approx \text{Exp}_{\text{PK-DE-PIR}}^{\text{sim}}$ for some efficient \mathcal{S} .

Now let the scheme be an SK-DE-PIR (Definition 22) and let \mathcal{S} be *stateful*. Consider $\text{Exp}_{\text{SK-DE-PIR}}^{\text{real}}(1^\lambda, \mathcal{A})$ and $\text{Exp}_{\text{SK-DE-PIR}}^{\text{sim}}(1^\lambda, \mathcal{A})$:

- **Setup.** Launch $\mathcal{A}(1^\lambda)$ and receive $D \in \{0, 1\}^*$ from it. Run

$$\begin{cases} (\text{sk}, \tilde{D}) \stackrel{\$}{\leftarrow} \text{Process}(1^\lambda, D), & \text{in } \text{Exp}_{\text{SK-DE-PIR}}^{\text{real}}; \\ \tilde{D} \stackrel{\$}{\leftarrow} \mathcal{S}(1^\lambda, D), & \text{in } \text{Exp}_{\text{SK-DE-PIR}}^{\text{sim}}; \end{cases}$$

and send \tilde{D} to \mathcal{A} .

- **Challenge.** Repeat the following for arbitrarily many rounds determined by \mathcal{A} . In each round, \mathcal{A} submits an index $i_q \in [|D|]$. Upon such a challenge, run

$$\begin{cases} (\text{ct}_q, \sigma_q) \stackrel{\$}{\leftarrow} \text{Query}(1^\lambda, \text{sk}, i_q), & \text{in } \text{Exp}_{\text{SK-DE-PIR}}^{\text{real}}; \\ \text{ct}_q \stackrel{\$}{\leftarrow} \mathcal{S}(1^\lambda, D[i_q]), & \text{in } \text{Exp}_{\text{SK-DE-PIR}}^{\text{sim}}; \end{cases}$$

and send ct_q to \mathcal{A} .

- **Guess.** The adversary outputs a bit β' , which is the output of the experiment.

The scheme satisfies *adaptive simulation security* if $\text{Exp}_{\text{SK-DE-PIR}}^{\text{real}} \approx \text{Exp}_{\text{SK-DE-PIR}}^{\text{sim}}$ for some efficient \mathcal{S} .³³

For PK- and SK-DE-PIR, the following modifications can be applied independently to obtain different levels of security:

- For *very selective security*, all the challenges must be chosen together with D before the adversary gets $(\text{pk}$ and) \tilde{D} .
- For *output-hiding*, in **Challenge**, \mathcal{S} does not get $D[i]$ or $D[i_q]$ as input.
- For *database-hiding* (SK-DE-PIR only), in **Setup**, \mathcal{S} gets $1^{|D|}$ instead of D as input.

The following four proofs or transformations can be applied in arbitrary order for both PK- and SK-DE-PIR with minor tweaks. We exemplify them in the order of writing for SK-DE-PIR, i.e., we start with an SK-DE-PIR per Definition 22, and lift it to obtain first adaptive security, next output-hiding, then simulation security, and lastly database-hiding.

From Selective to Adaptive. A very selectively secure scheme is also adaptively secure. The reduction prepares itself for all possible queries.

Proof. Let \mathcal{A}' be an adaptive adversary and $\bar{Q} = \text{poly}(\lambda)$ an upper bound of the number of its queries. Consider the following very selective adversary \mathcal{A} :

³³The simulator must run in total polynomial time for any polynomial-time adversary (it is not allowed to violate the time limit by bumping up the state size through the iterations).

1. Launch \mathcal{A}' and receive $D \in \{0, 1\}^*$ from it. Chooses the same D together with $\bar{Q}|D|^2$ challenges

$$i_{q,i_0,i_1,1} = i_0, \quad i_{q,i_0,i_1,1} = \begin{cases} i_0, & \text{if } D[i_0] \neq D[i_1]; \\ i_1, & \text{if } D[i_0] = D[i_1]; \end{cases} \quad \forall q \in [\bar{Q}], i_0, i_1 \in [|D|].$$

2. Receive \tilde{D} and $\{\text{ct}_{q,i_0,i_1}\}_{q \in [\bar{Q}], i_0, i_1 \in [|D|]}$. Send \tilde{D} to \mathcal{A}' and respond to its adaptive challenges. For its q^{th} challenge $(i_{q,0}, i_{q,1})$, send $\text{ct}_{q,i_{q,0},i_{q,1}}$ to \mathcal{A}' .
3. Output whatever \mathcal{A}' outputs.

It is readily verified that \mathcal{A} is efficient, always makes admissible challenges, and perfectly emulates \mathcal{A}' in $\text{Exp}_{\text{DE-PIR}}^0$ [resp. $\text{Exp}_{\text{DE-PIR}}^1$] when it is given the left [resp. right] distribution in Definition 22. \square

Obtaining Output-Hiding. This property can be obtained by concatenating the database with its bitwise negation and randomly making the original or the negated query.

Proof Sketch. Let $(\text{Process}', \text{Query}', \text{Resp}', \text{Dec}')$ be an SK-DE-PIR with adaptive indistinguishability. We construct the following SK-DE-PIR:

- $\text{Process}(D)$ sets $N = |D|$, runs

$$D' = D[1] \parallel \dots \parallel D[|D|] \parallel (1 - D[1]) \parallel \dots \parallel (1 - D[|D|]), \quad (\text{sk}', \tilde{D}') \stackrel{\$}{\leftarrow} \text{Process}'(D'),$$

and outputs $\text{sk} = (\text{sk}', N)$ and $\tilde{D} = \tilde{D}'$.

- $\text{Query}(\text{sk}, i)$ samples $\beta' \stackrel{\$}{\leftarrow} \{0, 1\}$, runs

$$(\text{ct}', \sigma') \stackrel{\$}{\leftarrow} \text{Query}'(\text{sk}, \beta'N + i),$$

and outputs $\text{ct} = \text{ct}'$ and $\sigma = (\beta', \sigma')$.

- $\text{Resp}^{\tilde{D}}(\text{ct})$ runs and outputs $(\text{Resp}')^{\tilde{D}'}(\text{ct}')$.

- $\text{Dec}(\sigma, \rho)$ runs and outputs $(\beta' \oplus \text{Dec}'(\sigma', \rho))$.

The proof of adaptive indistinguishability with output-hiding is standard. \square

From Indistinguishability to Simulation. A scheme with indistinguishability security is automatically simulation-secure. The simulator works by issuing dummy queries.

Proof Sketch. Let $(\text{Process}, \text{Query}, \text{Resp}, \text{Dec})$ be an SK-DE-PIR with adaptive indistinguishability and output-hiding. Consider the following simulator \mathcal{S} :

- In **Setup**, $\mathcal{S}(D)$ runs

$$(\text{sk}, \tilde{D}) \stackrel{\$}{\leftarrow} \text{Process}(D),$$

keeps sk in its state, and outputs \tilde{D} as the simulated processed database.

- In **Challenge**, \mathcal{S} simulates a query by running

$$(\text{ct}, \sigma) \stackrel{\$}{\leftarrow} \text{Query}(\text{sk}, 1)$$

and outputting ct .

It is readily verified that \mathcal{S} is efficient and makes $\text{Exp}_{\text{SK-DE-PIR}}^{\text{real}} \approx \text{Exp}_{\text{SK-DE-PIR}}^{\text{sim}}$. \square

Obtaining Database-Hiding. This property can be obtained by encrypting the database.

Proof Sketch. Let $(\text{Process}', \text{Query}', \text{Resp}', \text{Dec}')$ be an SK-DE-PIR with adaptive simulation security and output-hiding. We construct the following SK-DE-PIR using a pseudorandom function PRF (suppose its output length is 1 and \mathbb{N} is appropriately encoded into its domain):

- $\text{Process}(D)$ samples a PRF key k_{PRF} , runs

$$D' \leftarrow (D[1] \oplus \text{PRF}(k_{\text{PRF}}, 1)) \parallel \dots \parallel (D[|D|] \oplus \text{PRF}(k_{\text{PRF}}, |D|)), \quad (\text{sk}', \tilde{D}') \stackrel{\$}{\leftarrow} \text{Process}'(D'),$$

and outputs $\text{sk} = (k_{\text{PRF}}, \text{sk}')$ and $\tilde{D} = \tilde{D}'$.

- $\text{Query}(\text{sk}, i)$ runs

$$(\text{ct}', \sigma') \stackrel{\$}{\leftarrow} \text{Query}'(\text{sk}', i), \quad \beta' \leftarrow \text{PRF}(k_{\text{PRF}}, i),$$

and outputs $\text{ct} = \text{ct}'$ and $\sigma = (\beta', \sigma')$.

- $\text{Resp}^{\tilde{D}}(\text{ct})$ runs and outputs $(\text{Resp}')^{\tilde{D}'}(\text{ct}')$.
- $\text{Dec}(\sigma, \rho)$ runs and outputs $(\beta' \oplus \text{Dec}(\sigma', \rho))$.

Let \mathcal{S}' be the old simulator, the new simulator \mathcal{S} works as follows:

- In **Setup**, $\mathcal{S}(1^N)$ samples $D \stackrel{\$}{\leftarrow} \{0, 1\}^N$ and delegates its work to $\mathcal{S}'(D)$.
- In **Challenge**, \mathcal{S} delegates its work to \mathcal{S}' .

The proof of adaptive simulation security with output- and database-hiding is standard. \square

B Efficiency, Security, and $\text{poly}(\lambda)$ Factors

We explain the complication about the efficiency parameters. The theme of this work is to obtain highly efficient schemes from any polynomially efficient scheme. It is expected that the concrete efficiency of the resultant scheme depends on that of the underlying scheme we start with, which could inherit a polynomial factor out of our control. Therefore, it is not useful to differentiate between $\text{poly}(\lambda)$ factors, except when it can be made concrete and independent of the building blocks (e.g., the rate being 2 in this work).

For a computationally secure primitive, suppose some input is of length N , then we can always assume $N \leq 2^\lambda$. In practice, this assumption has been implicit for hash functions — SHA-256 only accepts input of up to $(2^{64} - 1)$ bits, not fitting the usual theoretical formulation $\{0, 1\}^* \rightarrow \{0, 1\}^\lambda$. Of course, such limit does not jeopardize the usefulness of the primitive. This can be formalized by “truncating” the scheme. A primitive often has a correct yet insecure (trivial) implementation. Suppose a scheme accepts input of up to 2^λ bits, then it can be made to work with arbitrarily long input by using the trivial implementation when the input exceeds 2^λ bits. It does not affect computational security (asymptotic or concrete), because the security definition does not consider the case when the input is 2^λ -bit long.

For this reason, any $\text{poly}(\log N)$ factor can be “absorbed” into $\text{poly}(\lambda)$ factors (and ignored). The trick to notice is that when only considering polynomial security, truncation can happen at any super-polynomial length, potentially much smaller than 2^λ . For example, to hide $2^{\sqrt{\log N}}$ of [LMW23] inside $\text{poly}(\lambda)$, it suffices to truncate³⁴ the scheme at $N = 2^{\log^2 \lambda}$. In summary, efficiency parameters are only meaningful at specific levels of security.

However, truncation below exponential means that the scheme is *never* exponentially secure, a property desired (and even arguably required) in practice. For example, the aforementioned truncated scheme is not secure whenever $N \geq 2^{\log^2 \lambda}$. In contrast, the constructions in this work *are* exponentially secure if we start with exponentially secure ingredients. We should always aim for constructions that have polynomial security from polynomial hardness and exponential security from exponential hardness. Using truncation for apparent efficiency at the cost of security is “gaming” and “messing with the security parameter”. It should be avoided whenever possible.

Combining the two perspectives of the discussion, the schemes of [LMW23] does, but not *satisfactorily*, achieve the ideal efficiency for DE-PIR, and we regard the question of optimal DE-PIR open. (We shall clarify that [LMW23] never claims the better apparent efficiency via truncation. This discussion is our own.)

³⁴For DE-PIR, this means whenever $|D| \geq 2^{\log^2 \lambda}$, the scheme works by not preprocessing D at all and sending the query index in the clear to the server.