A Constant-Rate Compiler for Continual Memory Leakage

Shweta Agrawal* Anuja Modi† Rajeev Raghunath‡

Abstract

Leakage-resilient compilers transform any computation so that physical leakage from its implementation reveals nothing beyond its input-output behaviour. In this work, we investigate this question in the well-motivated continual memory leakage (CML) model, that captures a variety of relevant real-world side-channel attacks and subsumes several other well studied models in the leakage domain. In this model with two independently leaking components (at least two are necessary), all known general compilers tolerate only an inverse-polynomial leakage-rate.

We give a compiler with *constant* leakage-rate, in the hybrid model with a simple inputless leak-free sampler as in Dziembowski and Faust (TCC 2012). It protects any deterministic computation on a fixed secret with public inputs and outputs, and comes with a transfer theorem: any security game in which the adversary has an oracle for the computation remains secure when every oracle call is replaced by a compiled evaluation that leaks continuously. Our compiler uses no computational assumption.

Our main application is public-key encryption. Applied to the decryption of any IND-CCA2 secure PKE, the compiler gives IND-CCLA2 security, i.e. CCA2 security even when every decryption leaks, including after the challenge ciphertext, with the same public key, ciphertexts and encryption algorithm. Instantiating the base scheme gives such PKE from LWE and from LPN improving upon the previous best post-quantum constructions in this model, which supported only inverse-polynomial leakage-rate. We also obtain signatures, PRFs and MACs with constant leakage-rate via a natural generalization.

Our key idea is to pack the entire configuration of a universal machine into one element of a large extension field, so that any computation keeps only a constant number of inner-product encodings in memory at any time, each for only a constant number of rounds. Our techniques are inspired by packing techniques in homomorphic encryption adapted to a completely different context. We are optimistic that our techniques will find other applications.

*IIT Madras, India, shweta@cse.iitm.ac.in
†IIT Madras, India, anujamodi97@gmail.com
‡IIT Madras, India, rajeev.mrug08@gmail.com

Contents

1 Introduction **4**
- 1.1 Our Results . . . 4
- 1.2 Related Work . . . 6

2 Technical Overview **7**
- 2.1 Our Template: Output Simulation . . . 7
- 2.2 From Output Simulation to Security: A Transfer Theorem . . . 8
- 2.3 Starting Point: The DF Compiler . . . 9
- 2.4 Our Approach . . . 11
- 2.5 Simulating Leakage from the Concrete DF Equations . . . 15
- 2.6 Supporting Polynomially Many Adaptive Queries . . . 16
- 2.7 Instantiations . . . 18

3 Definitions and Preliminaries **19**
- 3.1 Leakage from Computation and Memory . . . 19
- 3.2 Leakage rate . . . 20
- 3.3 Statistical simulation of protected computation from efficient adversaries . . . 20

4 Full-Field Encodings and the LFH Functionality **21**
- 4.1 Inner-product based Leakage-Resilient Storage (LRS) . . . 21
- 4.2 The inputless Leak-Free Hardware (LFH) . . . 22
- 4.3 Concrete DF gadgets . . . 23
- 4.4 Simulating Leakage of DF Gadgets . . . 25

5 A Universal Constant-Rate CML Compiler **26**
- 5.1 Compiling General Computations into Constant-Width DF Programs . . . 27
- 5.2 Expanding the Compiler to support Oracles . . . 29

6 Instantiations in the LFH Model **32**
- 6.1 Public-Key Encryption . . . 32
- 6.2 PRFs and MACs . . . 33
- 6.3 Digital Signatures . . . 36

7 Acknowledgement of AI usage **37**

A Discussion on the CML Leakage Model **45**
- A.1 The Continual Memory Leakage (CML) model . . . 45
- A.2 Related Work for Leakage-Resilient PKE, MAC, Signatures . . . 46
- A.3 A comparison of other leakage models with CML . . . 48

B Additional Preliminaries **51**
- B.1 Clocked cellular simulation . . . 51
- B.2 Continuous leakage-resilient public-key encryption . . . 52
- B.3 Continuous leakage resilient PRF and MAC . . . 53
- B.4 Continuous leakage-resilient digital signatures . . . 55

C Concrete Protocols and Reconstruction of the DF Gadgets **56**
C.1 Reconstructing local randomness . . . 57
C.2 Exact refresh protocol and reconstruction . . . 58
C.3 Exact multiplication reconstruction . . . 60
C.4 Exact copy protocol and reconstruction . . . 62
C.5 Exact translation protocol and reconstruction . . . 64
C.6 Exact public output gadget and reconstruction . . . 65
C.7 Proof of theorem 2 . . . 67
C.8 Relation to Prior DF Reconstruction Techniques . . . 71

D Register Scheduling and Exposure **71**

E Hardware Replacement from Indistinguishability Obfuscation **73**

1 Introduction

Physical implementations of cryptographic algorithms leak. Side-channel attacks exploit timing [65], power consumption [66], electro-magnetic radiation [80], temperature [55] and memory remanence [49] to recover secret keys, and they break schemes that are secure in the black-box model, including recent post-quantum schemes [75, 82, 94]. Leakage-resilient cryptography responds to this by giving the adversary, in addition to its usual interfaces, the output of leakage functions applied to the internal state of the computation [56, 72, 34, 3, 18, 26]. A central goal in this area is a *leakage-resilient compiler*: an efficient transformation that takes any computation on a secret and outputs a functionally equivalent implementation whose leakage reveals nothing beyond its input-output behaviour [56, 59, 43, 33, 44]. Such a compiler separates the task of designing a primitive from the task of protecting its implementation.

The model and the leakage-rate. Several models have been proposed to formalize leakage resilience. We work in the continual memory leakage (CML) model with independently leaking components [18, 26, 33, 22], which subsumes the wire-probing, bounded-leakage and only-computation-leaks models (please see Appendix A for a discussion on CML and Appendix A.3 for a detailed comparison with other models). The secret state is split between two components in an initial leak-free initialization step. Then, in every round, the adversary may apply any efficient, adaptively chosen function to the complete state of one component: its share of the secret, its local randomness, its intermediate values, and every value it keeps for later use. The total output in a round is bounded by λ bits. The state is refreshed over time, so the total leakage over the lifetime of the secret is unbounded. Splitting the state is necessary, since leakage-resilience against a function of the whole state is impossible in general [8, 44]. The key parameter is the *leakage-rate*, the ratio of λ to the size of a share of the persistent secret state.

The state of the art. The general compilers known to be secure in this model [59, 33, 44] (see [14, 22]) all have inverse-polynomial leakage-rate. The compiler of Dziembowski and Faust [33] works in the optimal two-independent-component model with a small leak-free sampler, but tolerates only a fraction of order $1/t$ where t is the width of the circuit. The compiler of Goldwasser and Rothblum [44] avoids leak-free hardware, but needs polynomially many components. The compiler of Juma and Vahlis [59] stores an FHE encryption of the secret, which is polynomially larger than the leakage it tolerates. Constant leakage-rate is known only for specific primitives in weaker settings: encryption and signatures with a single leaking component from bilinear-group assumptions, where leakage after the challenge ciphertext is impossible [18, 27, 67], CCA2 encryption with post-challenge leakage in the only-computation-leaks model from DDH [32], and general computation when every wire leaks independently with a constant probability [1, 6, 5], a much weaker form of leakage. This leads to the question:

Is constant-rate continual leakage-resilience possible for general computation in the two-component CML model?

1.1 Our Results

We answer the above question in the affirmative for every deterministic computation on a fixed secret with public inputs and outputs, in the hybrid model with a simple leak-free hardware (or sampler) as in [32, 33]. Key generation and the initial encoding of the secret are leak-free, after which, every round of every computation, including every refresh of the secret, may leak. The leak-free hardware (LFH) is called afresh in each refresh and multiplication step. As in prior work, it has no input, stores no long-term secret, and gives each component one of two random vectors whose

inner product is zero (see Section 4.2). We note that this leak-free hardware correlation can be removed using indistinguishability obfuscation (iO) and learning with errors (LWE) in the honestly generated public CRS model, by adapting the hardware replacement of [22] (see Appendix E). The leakage-rate is the ratio of λ, the per-round leakage bound, to the size of one persistent share stored by a component between evaluations.

A constant-rate compiler and a transfer theorem. Let $f(s, x)$ be any operation that is computed by a deterministic polynomial-time algorithm (with public bounds on its running time and space) from a fixed secret s and a public input x, and whose output $y = f(s, x)$ is public. Our compiler stores s as an inner-product encoding split between two components, and evaluates f using refresh and multiplication gadgets that we design, building upon [33]. Importantly, it does not change the public key or any public algorithm of the primitive. Its security guarantee is a transfer theorem: any security game in which the adversary has an oracle for $f(s, \cdot)$ remains secure when every oracle call is replaced by a compiled evaluation that leaks continuously.

Theorem 1 (Informal of Theorem 3)**.** *There is a constant H such that the following holds. For any constant $\rho < \frac{1}{2H}$ and any deterministic operation f as above, there is a compiled implementation of f with leakage-rate at least ρ in the LFH-hybrid model, with the following property. For any security game G in which the adversary has oracle access to $f(s, \cdot)$, any PPT adversary in the continual-leakage version of G can be converted into a PPT adversary in G whose output probability differs by a negligible amount. The latter queries its own oracle exactly the inputs on which the former runs the compiled evaluation. The compiler uses no computational assumption.*

To the best of our knowledge, this is the first general compiler with constant leakage-rate in the two-component CML model. We obtain the following applications, all with the same constant leakage-rate.

Continual leakage-resilient public-key encryption. Taking f to be the complete decryption algorithm and the base game to be IND-CCA2 gives protection against leakage from adaptive decryption queries both before and after the challenge. The challenge ciphertext itself is refused before protected computation or leakage, as required by the IND-CCLA2 experiment. Applying this result to the base schemes in section 6.1 yields the following concrete instantiations.

Corollary 1 (Informal, see corollary 4)**.** *For every fixed $\rho < \frac{1}{2H}$, there exists an IND-CCLA2-secure PKE with leakage-rate at least ρ in the two-component LFH-hybrid model under any one of the following assumptions:*

1. *polynomial hardness of LWE with polynomial modulus-to-noise ratio used by [68],*
2. *polynomial hardness of decisional learning parity with noise (LPN) with low noise rate $\mu = \Theta(1/\sqrt{n})$ and polynomially many samples,*
3. *$2^{\omega(\sqrt{n})}$ hardness of decisional LPN with constant noise rate $0 < \mu \leq 1/10$ required by [95].*

The schemes support polynomially many adaptive decryption queries with continual leakage, including post-challenge queries, and have negligible correctness error.

To our knowledge, before this work every plausibly post-quantum PKE in the CML model had only inverse-polynomial leakage-rate. Such schemes are obtained from the construction of [59] with a lattice-based FHE, or by applying the compiler of [33] (shown secure in this model in [22]) or that of [44] to a lattice-based PKE. As mentioned previously, the compiler of [44] also needs polynomially many components. The known constant-rate PKE schemes in this model [18, 27, 67] use a single leaking component, so they do not allow leakage after the challenge, and they rely on assumptions over bilinear groups, which are broken by quantum computers (please see Table 3 in Appendix A.2).

Continual leakage-resilient PRFs and MACs. These applications need only one-way functions, so we use decisional LWE with polynomial modulus-to-noise ratio, or decisional constant-noise LPN with a fixed noise rate in $(0, 1/2)$ and polynomially many samples, as specified in section 6.2.

Corollary 2 (Informal, see corollaries 5 and 6)**.** *Assume either of the LWE or constant-noise LPN assumptions as specified above. For every fixed $\rho < \frac{1}{2H}$, there exist the following primitives in the two-component LFH-hybrid model, each with leakage-rate at least ρ and negligible correctness error over polynomially many adaptive calls:*

1. *a PRF that stays pseudorandom under continual leakage. In more detail, the outputs of the protected evaluation, together with its leakage, are indistinguishable from the outputs of a truly random function together with simulated leakage.*
2. *a MAC that stays unforgeable under continual leakage. In more detail, the adversary may interleave tagging and verification queries on the same protected key, and both leak, yet it cannot make a verification query accept, or output a valid forgery, on a message it never asked to be tagged.*

Continual leakage-resilient digital signatures. The same assumptions also give ordinary EUF-CMA-secure signatures. We compile a deterministic stateless signer, and observe that a randomized stateless signer can first be derandomized using an independent PRF, with the PRF key and its evaluation included in the protected computation (section 6.3).

Corollary 3 (Informal, see corollary 7)**.** *Under either of the LWE or constant-noise LPN assumptions above, for every fixed $\rho < \frac{1}{2H}$, there exists a CML-secure signature scheme in the two-component LFH-hybrid model with leakage-rate at least ρ. It remains existentially unforgeable under polynomially many adaptive signing requests with continual leakage from signing and refreshes, and has negligible total correctness error.*

Removing the leak-free sampler. As noted by [22], a LFH for such simple randomized functionalities can in principle be replaced by a leakage-resilient protocol, at the expense of adding the assumption of indistinguishability obfuscation (iO) for general functionalities. We show in Appendix E that the compiler of [22] can be used to replace the leak-free hardware used by our theorem 3, using iO and LWE with an honestly generated public CRS.

1.2 Related Work

Leakage-resilient compilers. General compilers that protect any computation are known in several other leakage models. In the wire-probing model, the compiler of [56] has leakage-rate about $\frac{1}{t}$ for t probes, and constant rate is possible when every wire leaks independently with a constant probability [1, 6, 5]. Compilers are also known for noisy leakage [37, 79, 30] and for bounded communication leakage [46, 40, 57]. These models restrict which values leak or how the leakage is computed, and are either subsumed by CML or are not directly comparable to it (please see Appendix A.3).

The most relevant prior works for us are the general compilers of [33] and [44]. Dziembowski and Faust [33, Sec. 6] compile any circuit into a functionally equivalent circuit that is continuous leakage-resilient in the independent-leakage model, using a leakage free hardware (LFH) for random encodings of 0. The work of [22] later extends it to the CML model [22, App. A and App. B.2]. However, when restricted to two independent components, their construction only supports $\frac{1}{\mathrm{poly}(\kappa)}$ leakage-rate. The work of Goldwasser and Rothblum [44] show how to get leakage-resilience without any LFH via "ciphertext banks" which inherently requires number of independent components as a function of circuit size for leakage-resilience security.

Application to PKE. Our main application is PKE, for which the state of the art is summarized in Table 1. Please see Appendix A.2 for a detailed discussion on other prior work for PKE, PRF, MAC and signatures.

Work	Assumption	Leakage Rate	#Comp
[59]	FHE + LFH	$\frac{1}{\text{poly}}$	2
[33]	PKE + LFH	$\frac{1}{\text{poly}}$	2
[44]	PKE	$\frac{1}{\text{poly}}$	poly
This work	PKE + LFH	constant	2

Table 1: Leakage-resilient PKE in the CML model.

2 Technical Overview

We now outline the main technical ideas developed in this work. We work in the 2-component CML model, with the components denoted by Left, Right, and a common memory Common that is used for public communication. Let $f(s, x)$ be the deterministic secret-key operation that we want to protect, with a fixed secret s, a public input x and a public output $y = f(s, x)$. Our running example is PKE: for an IND-CCA2 secure PKE scheme $P_0 = (\mathsf{Gen}_0, \mathsf{Enc}_0, \mathsf{Dec}_0)$ (the subscript 0 indicates the base scheme, before compilation), we take $s = \mathsf{sk}_0$ and $f = \mathsf{Dec}_0$.

The CML model. In the CML model, a protected computation runs in rounds. In each round, one of the components Left and Right is active: it reads its own memory and Common, performs one complete local computation, and writes to its own memory and to Common. The two components communicate only through Common, and everything written there is public. In each round, the adversary makes adaptive leakage queries, given by efficient functions, with total output length at most λ bits, however long the local computation of that round is. Each leakage query is a function of Common together with one component: either the complete view of the active component in this round (its memory, random coins, intermediate values and candidate output), or the current memory of the inactive component. These queries are made before the component updates its memory and deletes the values that are no longer needed. Thus, a value that a component keeps for a later round can leak in every round until it is deleted. Please refer to Definition 2 for the formal definition, and to Appendix A for a discussion of the model, including the requirement that leakage functions are efficient.

Roadmap. In Sections 2.1 and 2.2 we describe how to reduce our task to building a protected evaluation of f whose leakage can be simulated from its inputs and outputs alone. The rest of this section builds such an evaluation, by adapting the inner-product encodings and gadgets of Dziembowski and Faust [33] (Section 2.3). Finally, Section 2.7 describes how we instantiate the transfer theorem for PKE, PRFs, MACs and signatures.

2.1 Our Template: Output Simulation

As a first step, we isolate the property that the protected computation must satisfy, so that security in the presence of leakage follows from the security of the base primitive in a modular way. For this, we use a simple simulation based notion called output simulation (please refer to Definition 3 for the formal definition). Informally, a stateful protected computation with public input x, private persistent state and public output y is output-simulatable if there exists a stateful PPT simulator Sim such that, for any adversary $\mathcal{A}$, the following experiments are statistically indistinguishable.

- $\mathcal{A}$ adaptively chooses inputs $x_1, x_2, \ldots, x_N$. On each input x_i, the protected computation is run, and y_i denotes its actual output.
- In the Real experiment, for each round of the computation on x_i, $\mathcal{A}$ gets λ-bits of adaptive leakage from either sub-component, computed on the real secret state and local randomness.
- In the Simulated experiment, $\mathcal{A}$ gets the public messages and the answers to its leakage queries from Sim, which is given only (x_i, y_i). The simulator retains its own private state for the next query, but never gets the secret state.

The distribution we compare is the complete view of $\mathcal{A}$, including its leakage answers, the public messages and its adaptively chosen inputs.

An important point is that the definition covers arbitrary public inputs together with their actual outputs. For PKE, this means that the adversary may ask for leakage on the decryption of malformed or adversarially crafted ciphertexts. This makes the definition stronger than a definition restricted to honestly generated inputs, and this is what enables a direct reduction to the security of the base primitive in the next step.

2.2 From Output Simulation to Security: A Transfer Theorem

Suppose that the base primitive is secure in a game G in which the adversary gets an oracle for $f(s, \cdot)$. The oracle may refuse some inputs according to a public rule – for example, the decryption oracle of IND-CCA2 refuses the challenge ciphertext. The game may have any other oracles, challenges and winning condition. Its continual-leakage version $\mathsf{G}_\lambda^{\mathsf{CML}}$ is the same game, except that s is stored by our compiler, and every allowed oracle query x runs the compiled evaluation of f on x, giving the adversary both the output and the leakage. Refused queries return $\perp$ without running any protected computation, and hence without leakage.

Given an adversary $\mathcal{A}$ for $\mathsf{G}_\lambda^{\mathsf{CML}}$, the adversary $\mathcal{B}$ for G runs $\mathcal{A}$ and forwards everything unchanged, except as follows. For each allowed query x of $\mathcal{A}$, it asks its own oracle for $y = f(s, x)$, gives (x, y) to Sim, and answers the leakage queries of $\mathcal{A}$ using the simulator. By output simulation, the view of $\mathcal{A}$, and hence the output of the game, changes by a negligible statistical amount. Here, $\mathcal{B}$ must be efficient, so it must evaluate the leakage functions of $\mathcal{A}$ on simulated views, which is why we require the efficiency of leakage functions. Please refer to Theorem 3 for the formal statement.

Upgrading to IND-CCLA2 security. For PKE, we let $f = \mathsf{Dec}_0$ and G be the IND-CCA2 game of P_0. Key generation runs Gen_0 and stores an encoding of sk_0 between Left and Right, encryption is Enc_0, and decryption of c is the protected computation with public input c and output $\mathsf{Dec}_0(\mathsf{sk}_0, c)$, which also refreshes the encoding of sk_0. The challenge ciphertext $c^* \leftarrow \mathsf{Enc}_0(\mathsf{pk}, m_b)$ is computed by the challenger from the public key alone, without the secret key, so it involves no protected computation. Thus, $\mathsf{G}_\lambda^{\mathsf{CML}}$ is exactly the IND-CCLA2 experiment. The reduction $\mathcal{B}$ forwards pk and the challenge, and asks its own decryption oracle for every query $c_i \neq c^*$. Since the exact challenge ciphertext is rejected before any protected computation (and hence before any leakage), every oracle query is legal. Hence, the advantage of $\mathcal{A}$ is bounded by the IND-CCA2 advantage against P_0 plus a negligible statistical term (Theorem 3).

On Naor-Yung and Fujisaki-Okamoto. A common template to get strong security is to start with a weakly-secure scheme and apply a generic transformation, such as the Fujisaki-Okamoto transform [39] or the Naor-Yung transform [73, 69]. The latter has been used in the context of leakage in [74] to upgrade CPA secure PKE to CCA security in the bounded leakage model, using a one-time simulation-sound NIZK [86]. Our compiler does not add a second public key, a second ciphertext, or a NIZK but instead compiles an already CCA2 secure decryption algorithm. If P_0 itself uses Naor-Yung or a public proof, those objects remain inside P_0. In the FO transform, at the

core of the decryption algorithm is a re-encryption and verify check that enables non-malleability. Unfortunately, as shown by recent works [75, 82, 94], this re-encryption step is extremely vulnerable to side-channel attacks. In particular, it allows the adversary to get leakage on arbitrary bogus ciphertexts that do not verify (please see [54] and the references therein). We note that in our template, leakage on the decryption of arbitrary ciphertexts is exactly what the simulator is required to handle, and the complete decryption algorithm of P_0, including any verification or re-encryption check, is evaluated inside the protected computation.

Thus, the remaining technical challenge is to build an output-simulatable protected evaluation for an arbitrary deterministic f, with constant leakage-rate and for polynomially many adaptive queries. We describe this next.

2.3 Starting Point: The DF Compiler

Let $\mathbb{K}$ be a finite field with $Q = |\mathbb{K}|$ and $N = \log_2 Q$, and let ν be a parameter which will be closely related to the leakage-bound supported. We use inner-product (IP) based encoding to store secrets: a value $z \in \mathbb{K}$ is encoded as $(L, R) \in \mathbb{K}^\nu \times \mathbb{K}^\nu$ such that $\langle L, R\rangle = z$, where L is stored in Left and R is stored in Right.

IP based encoding has been used before in other models [24, 32, 33] and IP based masking schemes have also been used in practice [7]. Since the inner product is a two-source extractor [81, 32], this encoding is a leakage-resilient storage (LRS) scheme: an adversary that gets adaptive and independent leakage of (in total) a little less than $\frac{\nu}{2}N$ bits from L and R cannot distinguish an encoding of z from an encoding of any other $\tilde{z}$ (see Lemma 1).

Dziembowski and Faust [32, 33] show how to compute on such encodings using an LFH that outputs a fresh random encoding of 0. We use the following variant of it, denoted $\mathcal{O}$ (this is $\mathcal{O}'_{\mathbb{K},\nu}$ of Section 4.2). A call to $\mathcal{O}$ samples $A, \widetilde{A}, B, \widetilde{B} \in \mathbb{K}^\nu$ uniformly subject to

$$\langle A, B\rangle + \langle \widetilde{A}, \widetilde{B}\rangle = 0, \qquad A \neq 0, \qquad \widetilde{B} \neq 0,$$

writes $(A, \widetilde{A})$ privately to Left and $(B, \widetilde{B})$ privately to Right, and writes nothing to Common. Let $\mathrm{GL}_\nu(\mathbb{K})$ denote the set of invertible $\nu \times \nu$ matrices over $\mathbb{K}$. We use column vectors, and for a vector V we write V_i for its i-th coordinate and $V_{a..b}$ for the sub-vector $(V_a, \ldots, V_b)$. The refresh of an encoding (L, R) of z, with $L \neq 0$, takes three rounds, followed by a commit step.

- In component Left, sample M uniformly from $\{M \in \mathrm{GL}_\nu(\mathbb{K}) : ML = A\}$, and write M to Common.
- In component Right, read M from Common, set $R^+ := R + M^T B$, and announce whether $R^+_{2..\nu} \neq 0$. If so, sample $\widetilde{M}$ uniformly from $\{\widetilde{M} \in \mathrm{GL}_\nu(\mathbb{K}) : \widetilde{M}R^+ = \widetilde{B}\}$, and write $\widetilde{M}$ to Common.
- In component Left, read $\widetilde{M}$ from Common, set $L^+ := L + \widetilde{M}^T \widetilde{A}$, and announce whether $L^+_1 \neq 0$.

The two sets of matrices are nonempty, since $L, A \neq 0$ and $R^+, \widetilde{B} \neq 0$. If both checks pass, the refresh commits the new encoding (L^+, R^+), otherwise, both components keep (L, R). The checks fail with probability at most $2/Q$, since each fails only when a uniformly random value is zero (see Appendix C.2). The first check ensures that the *right tail* $R^+_{2..\nu}$ of the new encoding is nonzero. In particular $R^+ \neq 0$, so that $\widetilde{M}$ exists. The second ensures that the first coordinate of the new left share is nonzero, which the compression and translation steps below need, since they divide by it. Using $\langle u, Mv\rangle = \langle M^T u, v\rangle$, we get

$$\begin{aligned}\langle L^+, R^+\rangle &= \langle L, R^+\rangle + \langle \widetilde{M}^T \widetilde{A}, R^+\rangle = \langle L, R\rangle + \langle ML, B\rangle + \langle \widetilde{A}, \widetilde{M}R^+\rangle \\ &= \langle L, R\rangle + \langle A, B\rangle + \langle \widetilde{A}, \widetilde{B}\rangle = z,\end{aligned}$$

where the last step uses $\langle A, B\rangle + \langle \widetilde{A}, \widetilde{B}\rangle = 0$. Thus, the matrices M and $\widetilde{M}$ are public, while $(A, \widetilde{A})$ and $(B, \widetilde{B})$ stay in their components. Publishing these matrices is harmless on its own: a counting argument shows that, conditioned on the refresh committing, $(M, \widetilde{M})$ is uniform over $\mathrm{GL}_\nu(\mathbb{K})^2$ and independent of the pair of encodings $\big((L, R), (L^+, R^+)\big)$. We handle these matrices together with the leakage in Section 2.5.

To compute on encodings, a computation is written as a program in which every operation on secret values is a copy, a translation by a public constant, or the gate $G_c(s_0, s_1) = c - s_0 s_1$ for a public constant c (for $c = 1$ this is NAND on bits). Every secret value is kept as an IP encoding, and every operation is replaced by a gadget that maps encodings of its inputs to an encoding of its output, without reconstructing any secret value. The gadget for G_c has two steps.

First, it forms an encoding of the product. Given encodings $(L^{(0)}, R^{(0)})$ and $(L^{(1)}, R^{(1)})$ of s_0 and s_1, Left computes $L^{(0)} \otimes L^{(1)}$ and Right computes $R^{(0)} \otimes R^{(1)}$, each from its own shares only. Since

$$\langle L^{(0)} \otimes L^{(1)}, R^{(0)} \otimes R^{(1)}\rangle = \langle L^{(0)}, R^{(0)}\rangle \langle L^{(1)}, R^{(1)}\rangle = s_0 s_1,$$

the pair $(L^{(0)} \otimes L^{(1)}, R^{(0)} \otimes R^{(1)})$ is an encoding of $s_0 s_1$, but of dimension ν^2.

Second, it refreshes this pair to (L', R') (with an LFH call of dimension ν^2) and compresses it to dimension ν, applying $z \mapsto c - z$ along the way. We split L' and R' into their first ν coordinates (the *front*) and their last $\nu^2 - \nu$ coordinates (the *back*), so that

$$s_0 s_1 = \langle L', R'\rangle = \langle L'_{1..\nu}, R'_{1..\nu}\rangle + \langle L'_{\nu+1..\nu^2}, R'_{\nu+1..\nu^2}\rangle.$$

The new left share is set as the front of L'. If the new right share were simply the front of R', the output would encode $\langle L'_{1..\nu}, R'_{1..\nu}\rangle = s_0 s_1 - t$, where $t = \langle L'_{\nu+1..\nu^2}, R'_{\nu+1..\nu^2}\rangle$ is the contribution of the backs. This is wrong in two ways: it is off by t, and it is not of the form $c - s_0 s_1$. Right fixes both by changing its front in two steps.

- It negates its front. The inner product becomes $-(s_0 s_1 - t) = t - s_0 s_1$.
- It adds the vector $e_1 x^{-1}(c - t)$, where $e_1 = (1, 0, \ldots, 0)^T$ is the first standard basis vector of $\mathbb{K}^\nu$ and $x := L'_1$. Since the first coordinate of the new left share is x, that is, $\langle L'_{1..\nu}, e_1\rangle = x$, this adds exactly $x \cdot x^{-1}(c - t) = c - t$ to the inner product.

To do this, Right needs t and x, and both depend on L'. So Left first writes the back $L'_{\nu+1..\nu^2}$ and the single field element x to Common. The refresh of the tensor pair succeeds only if $x \neq 0$ (otherwise the gadget takes a public failure branch), so x^{-1} exists. Right then computes t from the back of L' and its own back. In summary, the output is

$$L^{(w)} = L'_{1..\nu}, \qquad R^{(w)} = -R'_{1..\nu} + e_1 x^{-1}(c - t),$$

and $\langle L^{(w)}, R^{(w)}\rangle = (t - s_0 s_1) + (c - t) = c - s_0 s_1$. As with the refresh, the gadget commits this output only if $R^{(w)}_{2..\nu} \neq 0$, and otherwise takes the public failure branch.

Copy and translation are simpler. To copy an encoding (L, R) of z, we run two independent refreshes, both starting from (L, R). If both succeed, we keep the two new encodings of z, one for each later use, and delete (L, R), otherwise, we keep (L, R). To translate an encoding (L, R) of z by a public constant c, we use the same idea as in the compression step. Left writes $x := L_1$ to Common. If x was already published for this encoding, this repeats the same value, so nothing new becomes public (see Section 2.5). Here $x \neq 0$, since every encoding stored between gadgets has a nonzero first left coordinate: the encoder samples it nonzero, every refresh commits only if the new first left coordinate is nonzero, and for a multiplication output, $L^{(w)}_1 = L'_1$, which the refresh inside

the gadget checks. Then Right sets $\widehat{R} = R + e_1 x^{-1} c$. Since $\langle L, e_1 \rangle = x$, we get $\langle L, \widehat{R} \rangle = z + c$. Finally, $(L, \widehat{R})$ is refreshed, so that the output is a fresh encoding of $z + c$.

Finally, an encoding is opened by writing its left share to Common, so that Right can compute and publish $\langle L, R \rangle$. This is the only step that makes a whole share public, and we use it only for the final output y of the computation. Since its left share is now public, the opened pair is deleted right after opening and is never used again. We always open a fresh pair, obtained from the value to be opened by a dedicated refresh; we call this the *public output gadget*, and explain in Section 2.5 why the extra refresh is needed.

Leakage rate is inverse-polynomial. The DF compiler is general and elegant: it compiles any circuit gate by gate, its security is information-theoretic, and the only leak-free component is the simple sampler $\mathcal{O}$. DF also note that for continual evaluation the memory of the gadgets can be reused by erasing values that are no longer needed [33, Sec. 6]. However, its leakage-rate is inherently inverse-polynomial. Every wire-value of the circuit is a separate encoding, so the number t' of encodings that must be stored at the same time is lower bounded by the width of the circuit, and each component stores one share of each of them, that is, $t'\nu$ field elements. The adversary can direct its entire leakage in a round to the share of a single wire's encoding, and each encoding only tolerates leakage of less than about half of one share. Thus, the analysis can only support a leakage-bound of $\lambda < \frac{\nu}{2} N$ (recall that $N = \log_2 Q$), which gives a leakage-rate of at most about $\frac{(\nu/2)N}{t'\nu N} = \frac{1}{2t'}$. Consider the decryption circuit of a PKE scheme whose secret key consists of n field elements (for example, n is the LWE dimension for Regev-style decryption [83]). Then $t' \geq n$, and the leakage-rate is at most about $\frac{1}{2n}$.

To get constant leakage-rate, we need two things: (i) a way to evaluate an arbitrary deterministic computation with only a constant number of live encodings, and (ii) an analysis showing that the leakage on each physical encoding, accumulated over all the rounds in which it is present, stays below the LRS bound, even across polynomially many adaptive queries. We describe our approach to (i) in Section 2.4, and address (ii) in Sections 2.5 and 2.6.

2.4 Our Approach

Our key departure from the DF compiler is to use a sufficiently large extension field to pack the entire state of a computation into a single field element, rather than maintaining a separate encoding for each circuit wire. The motivation is that the fraction of a share that may leak does not depend on the field: a share has about νN bits, and the encoding tolerates leakage of a little less than $\frac{\nu}{2} N$ bits, that is, about half of one share, for every large enough field. Thus, increasing the field size allows us to store more information in each encoding without reducing the fraction of leakage tolerated by that encoding. In contrast, in the gate-by-gate approach of the DF compiler, increasing the field size does not remove the dependence of the leakage rate on the number of simultaneously stored encodings.

Concretely, fix once and for all a deterministic universal single-tape machine $\mathcal{U}$, whose finite control and transition rule are independent of f and of the security parameter. $\mathcal{U}$ interprets the public description of f (along with its public time and space bounds) on the secret s and the public input x. Let $D \geq \kappa$ bound the bit length of one padded configuration of $\mathcal{U}$, i.e. its tape, head position and control state. We choose a prime $D < p < 2D$, which always exists, set $\mathbb{K} = \mathbb{F}_{p^D}$, fix a public basis $\beta_0, \ldots, \beta_{D-1}$ of $\mathbb{K}$ over $\mathbb{F}_p$, and pack a configuration, written as a bit string $b = (b_0, \ldots, b_{D-1}) \in \{0,1\}^D$, as the single field element $X = \sum_i b_i \beta_i$. Here we view the bits $0, 1$ as elements of $\mathbb{F}_p$. Since $\beta_0, \ldots, \beta_{D-1}$ is a basis, X determines b. We emphasize that any basis will do, since the restriction to $\{0,1\}$ is on the coefficients, not on the basis. Most elements of $\mathbb{K}$ have a coefficient outside $\{0,1\}$, and many bit strings encode no configuration, but we ensure that neither

ever occurs as X – we start from a packed configuration, and each step maps it to the packed next configuration.

The challenge is to manipulate this packed representation using only the operations supported by the DF gadgets, namely the gates G_c and translations by public constants, and with a constant number of field registers, so that only a constant number of encodings is live at any time, independently of f.

The obstacle: reading a packed element needs addition. To compute a step of the machine, we must read the bits of the configuration packed in X. For this, we use the trace from $\mathbb{K}$ to $\mathbb{F}_p$. For $Z \in \mathbb{K}$, it is the sum of the D powers $Z, Z^p, \ldots, Z^{p^{D-1}}$, which are the images of Z under the automorphisms of $\mathbb{K}$ that fix $\mathbb{F}_p$ (the *Galois conjugates* of Z):

$$\mathrm{Tr}(Z) = Z + Z^p + \cdots + Z^{p^{D-1}}.$$

The trace maps $\mathbb{K}$ to $\mathbb{F}_p$, and it is $\mathbb{F}_p$-linear. Every basis $\beta_0, \ldots, \beta_{D-1}$ of $\mathbb{K}$ over $\mathbb{F}_p$ has a unique *trace-dual basis* $\beta_0^\vee, \ldots, \beta_{D-1}^\vee$, defined by $\mathrm{Tr}(\beta_i^\vee \beta_j) = 1$ if $i = j$, and 0 otherwise. It exists because the bilinear map $(Y, Z) \mapsto \mathrm{Tr}(YZ)$ is nondegenerate, and it is computed by inverting a $D \times D$ matrix over $\mathbb{F}_p$. Then the i-th bit of the packed configuration is recovered as

$$\mathrm{Tr}(\beta_i^\vee X) = \mathrm{Tr}\Big(\beta_i^\vee \sum_{j=0}^{D-1} b_j \beta_j\Big) = \sum_{j=0}^{D-1} b_j \,\mathrm{Tr}(\beta_i^\vee \beta_j) = b_i,$$

where the second step uses that Tr is $\mathbb{F}_p$-linear and each $b_j \in \mathbb{F}_p$, and the third uses $\mathrm{Tr}(\beta_i^\vee \beta_j) = 1$ if $i = j$ and 0 otherwise. This is how packed values are read in homomorphic encryption [90, 41, 21] too. In that setting, the powers Z^{p^k} are obtained by Frobenius automorphisms and then added, which is cheap since adding two ciphertexts is cheap. With IP encodings, however, adding two secret values is not a local operation. Let $(L^{(0)}, R^{(0)})$ and $(L^{(1)}, R^{(1)})$ encode s_0 and s_1, with independent left shares. Adding the shares coordinate-wise gives

$$\langle L^{(0)} + L^{(1)},\, R^{(0)} + R^{(1)} \rangle = s_0 + s_1 + \langle L^{(0)}, R^{(1)} \rangle + \langle L^{(1)}, R^{(0)} \rangle,$$

and each cross term needs one share from each component, so neither component can remove it alone. If both encodings used the same left share L, the right component alone could output $(L, R^{(0)} + R^{(1)})$, an encoding of $s_0 + s_1$; but then the two encodings are correlated through L, so leakage on L is leakage on both, and the two can no longer be analysed one at a time. So an addition gadget has to be interactive, like the multiplication gadget: for example, one can concatenate the two encodings into an encoding $(L^{(0)} \| L^{(1)},\, R^{(0)} \| R^{(1)})$ of $s_0 + s_1$ of length 2ν, refresh it, and compress it to length ν. Such a gadget would need its own simulation proof. However, as we show next, products and translations by public constants already suffice to read the packed configuration, so we keep to the five gadgets of Section 2.3, for which we prove simulation.

Overcoming the obstacle. To read X, we compute the required traces through norms, which can be evaluated using only products. The norm from $\mathbb{K}$ to $\mathbb{F}_p$ is the product of the same D powers,

$$\mathrm{N}(Z) = Z \cdot Z^p \cdots Z^{p^{D-1}} = Z^{(Q-1)/(p-1)},$$

since $1 + p + \cdots + p^{D-1} = (Q-1)/(p-1)$ (recall that $Q = |\mathbb{K}| = p^D$). Like the trace, it maps $\mathbb{K}$ to $\mathbb{F}_p$, and it is computed by repeated squaring and multiplication, with a constant number of registers.

The key observation concerns the norms $\mathrm{N}(1+tZ)$ for $t \in \mathbb{F}_p$:

$$\mathrm{N}(1+tZ) = \prod_{k=0}^{D-1} (1+tZ)^{p^k} = \prod_{k=0}^{D-1} \left(1+t^{p^k} Z^{p^k}\right) = \prod_{k=0}^{D-1} \left(1+tZ^{p^k}\right) = 1+t\,\mathrm{Tr}(Z)+t^2(\cdots)+\cdots+t^D\,\mathrm{N}(Z).$$

The second equality holds since $x \mapsto x^{p^k}$ is a field automorphism of $\mathbb{K}$ (in characteristic p, $(a+b)^p = a^p + b^p$). The third holds since $t^{p^k} = t$ for every $t \in \mathbb{F}_p$, as $t^p = t$ by Fermat's little theorem. Thus, the p values $\mathrm{N}(1+tZ)$, for $t \in \mathbb{F}_p$, are the evaluations of a fixed polynomial of degree at most D over $\mathbb{F}_p$: its coefficient of t^j is the j-th elementary symmetric function of the D powers $Z, Z^p, \ldots, Z^{p^{D-1}}$ (counted with repetition), and in particular its linear coefficient is $\mathrm{Tr}(Z)$ (see eq. (11)).[1]

Since $D < p$, these p values determine the polynomial, and we can write:

$$\mathrm{Tr}(Z) = -\sum_{t \in \mathbb{F}_p} t^{p-2}\,\mathrm{N}(1+tZ).$$

To see this, write $\mathrm{N}(1+tZ) = \sum_{j=0}^{D} h_j t^j$, where $h_1 = \mathrm{Tr}(Z)$ by the expansion above, and use the identity

$$\sum_{t \in \mathbb{F}_p} t^m = \begin{cases} -1, & (p-1) \mid m, \\ 0, & \text{otherwise}, \end{cases} \qquad \text{for every integer } m \geq 1,$$

which holds since $t = 0$ contributes 0 and $\mathbb{F}_p^*$ is cyclic of order $p-1$. Then $\sum_t t^{p-2}\,\mathrm{N}(1+tZ) = \sum_{j=0}^{D} h_j \sum_t t^{p-2+j}$, where each exponent $p-2+j$ is at least 1 since $p \geq 3$, and for $0 \leq j \leq D$ it is a multiple of $p-1$ only for $j = 1$, since $D < p$. Hence $\sum_t t^{p-2}\,\mathrm{N}(1+tZ) = -h_1 = -\mathrm{Tr}(Z)$.[2]

The formula for $\mathrm{Tr}(Z)$ still adds secret values, namely the p terms $-t^{p-2}\,\mathrm{N}(1+tZ)$. However, these terms lie in the small field $\mathbb{F}_p$, where $p < 2D$, so we can add them by a polynomial-size lookup that uses only products and public translations. For $u, v \in \mathbb{F}_p$ and each $(a,b) \in \mathbb{F}_p^2$, let

$$I_{a,b}(u,v) = \left(1-(u-a)^{p-1}\right)\left(1-(v-b)^{p-1}\right).$$

By Fermat's little theorem, $I_{a,b}(u,v)$ equals 1 if $(u,v) = (a,b)$, and 0 otherwise. Hence,

$$u+v = \prod_{(a,b) \in \mathbb{F}_p^2} \left(1 - I_{a,b}(u,v)(1-a-b)\right),$$

since the factor for $(a,b) = (u,v)$ equals $u+v$, and all other factors equal 1. The factors are evaluated sequentially, with $O(p^2 \log p)$ multiplications and a constant number of live registers.

To read the bit $b_i = \mathrm{Tr}(\beta_i^\vee X)$, we first compute $Z = \beta_i^\vee X$, which is a product with a public constant. Then, for each $t \in \mathbb{F}_p$ in public order, we compute the norm $\mathrm{N}(1+tZ) = (1+tZ)^{(Q-1)/(p-1)}$ by repeated squaring and multiplication, multiply it by the public constant $-t^{p-2}$, and add the result to a running sum in $\mathbb{F}_p$ by this lookup. We never store all p terms, so only a constant number of registers is live. Thus, reading one bit takes p norms, each with $O(D \log p)$ multiplications, and p lookups; this is polynomial, since $p < 2D$.

Building the next configuration. One step of $\mathcal{U}$ maps the current configuration b, packed in X, to the next configuration b'. We use a standard cellular simulation of $\mathcal{U}$ (see, e.g., [91],

[1] For example, let $p = 3$, $D = 2$ and $\mathbb{K} = \mathbb{F}_3[\alpha]/(\alpha^2+1)$. Then $\mathrm{Tr}(Z) = Z + Z^3$ and $\mathrm{N}(Z) = Z^4$. For $Z = \alpha$, we have $\alpha^3 = -\alpha$, so $\mathrm{N}(1+t\alpha) = (1+t\alpha)(1-t\alpha) = 1+t^2$, whose coefficient of t is $0 = \mathrm{Tr}(\alpha)$.

[2] This is where we need $p > D$; in particular, reading over $\mathbb{F}_2$ (that is, $p = 2$) would not do.

and Appendix B.1 for the variant we use), in which each cell stores a tape symbol and possibly a head/control marker, encoded by a constant number of bits.

In this simulation, the cells are arranged in a row, and the new content of cell j is a function of the old contents of the cells $j-r,\dots,j+r$, determined by the transition table of $\mathcal{U}$; here the radius r is a constant that depends only on $\mathcal{U}$ (for a single-tape machine, $r=1$ suffices, since the head moves by at most one cell per step). Since $\mathcal{U}$ is fixed, this function is the same for every f, which enters only through the tape contents. It covers the symbol written under the head, the new control state and the head move.

Since cells are written in binary, each bit b'_i of the next configuration is a fixed Boolean function, determined by the public position i, of the bits of b that encode the cells within distance r of the cell containing position i (its *neighbourhood*). So we compute b'_i in two steps: we read the bits of the neighbourhood from X, one at a time, as described above, and we evaluate the Boolean function with NAND gates G_1.

Packing these bits into the next configuration also needs no secret addition. We build the next configuration in a new field element Y, which we call the *accumulator*: it starts at $Y=0$, and we write the bits of the next configuration into it one position at a time, in public order. To write the secret bit b'_i at the public position i, we must replace Y by $Y+b'_i\beta_i$. Computing this directly would add two secret values, Y and $b'_i\beta_i$. Instead, we compute both candidates, Y and $Y^+=Y+\beta_i$ (the latter is a translation by the public constant β_i), and select the right one with G_c gates:

$$F_0 = 1-(1-b'_i)(1-Y), \qquad F_1 = 1-b'_i(1-Y^+), \qquad Y' = F_0F_1.$$

If $b'_i=0$, then $F_0=Y$, $F_1=1$ and $Y'=Y$; if $b'_i=1$, then $F_0=1$, $F_1=Y^+$ and $Y'=Y+\beta_i$. So $Y'=Y+b'_i\beta_i$ in both cases. After the last position, Y is the packed next configuration $\sum_i b'_i\beta_i$, and it replaces X for the next step. The machine is run for its public number of steps, and a halting state simply repeats itself.

Thus, norms allow us to extract bits from the packed state using products, lookups replace the remaining additions over the small field $\mathbb{F}_p$, and the selection above writes bits without adding secret values. Consequently, every step of the machine can be implemented using only G_c gates and public translations, with a constant number of live registers.

Between two evaluations, the secret state is a single encoding $(L^{(S)},R^{(S)})$ of the packed secret $S\in\mathbb{K}$. At the start of a query, two refreshed copies of it are made: one enters the working configuration (via a translation by the packed public input), and the other continues as the encoding reserved for the persistent secret state. At the end of the query, the output (for PKE, a message or $\perp$) is packed into one field element, and the public output gadget makes it public. Note that each component now stores only ν elements of $\mathbb{K}$ between queries, which is what allows constant leakage-rate.

Protecting the evaluation. To protect the evaluation of $\mathcal{U}$ described above, which is a sequence of G_c gates and translations by public constants fixed by $\mathcal{U}$ and the public bounds, we evaluate every G_c gate by the multiplication gadget, every translation by the translation gadget, and make a copy whenever a value is used twice.

Together with refresh and the public output gadget, these are the five gadgets described in Section 2.3. Every protected computation in the present work is built from these five, and we prove a simulation theorem (Theorem 2) for these. It is easy to see that they suffice for any computation since G_1 gives NAND, copies give fanout, and over $\mathbb{K}$ these gadgets compute products, negations and translations by public constants. During a step, the program keeps only the old configuration, the new accumulator, a separately refreshed encoding of the packed secret and a constant number of scratch registers (for the current neighbourhood bits, the trace computation and the lookups).

Thus, the number W of live encoded field registers is an absolute constant for the fixed machine $\mathcal{U}$, and does not depend on f. Moreover, after every instruction, each live register that is not used by that instruction is also refreshed (we say that it is *carried*), and its old encoding is deleted. So no physical encoding stays in memory for more than a constant number of rounds, although the evaluation itself takes polynomially many rounds. Please refer to Lemma 3 and Table 2 for details.

2.5 Simulating Leakage from the Concrete DF Equations

Our security argument follows the leakage-resilient storage approach of Dziembowski and Faust [32, 33]. The inner-product encoding provides the following guarantee (Lemma 1): even under adaptive leakage queries on its two shares, an adversary cannot distinguish an encoding of a secret z from an encoding of 0, provided that the total leakage remains within the prescribed bound. Following the output-simulation approach of DF [33], we aim to extend this guarantee from stored encodings to computations on them.

The main difficulty is that leakage during a computation is not restricted to the stored shares. The adversary may obtain leakage from a component's complete view of a gadget, including its intermediate values, randomness, and received messages. To apply LRS security, we therefore need to express such leakage as leakage from individual shares of the target encoding. This is achieved through *one-sided local reconstruction*: after fixing the other encodings, the complete view of each component in every gadget incident to the target encoding can be reconstructed using only that component's share of the target, together with independently sampled randomness, fixed values, and a small amount of public information. Consequently, leakage from a reconstructed component view becomes an admissible leakage query on the corresponding share. LRS security then guarantees that replacing the target encoding by an encoding of 0 is indistinguishable.

While DF [33] establish output simulation for their compiler, our proof requires this explicit one-sided reconstruction property to carry out the encoding-replacement hybrids across polynomially many adaptive queries. We establish it directly from the concrete gadget equations (Theorem 2), by reconstructing each gadget backwards from its incident encodings, as follows.

Inverse refresh. In the hybrids, we do not run a refresh forwards. Instead, we first fix both encodings (L, R) and (L^+, R^+), and then fill in a refresh transcript that joins them. For this, sample $M, \widetilde{M} \leftarrow \mathrm{GL}_\nu(\mathbb{K})$ independently and set

$$\begin{aligned} A &= ML, & B &= M^{-T}(R^+ - R),\\ \widetilde{B} &= \widetilde{M}R^+, & \widetilde{A} &= \widetilde{M}^{-T}(L^+ - L). \end{aligned}$$

These values replay the refresh equations exactly: $R + M^T B = R^+$ and $L + \widetilde{M}^T \widetilde{A} = L^+$. Moreover,

$$\langle A, B\rangle + \langle \widetilde{A}, \widetilde{B}\rangle = \langle L, R^+ - R\rangle + \langle L^+ - L, R^+\rangle = \langle L^+, R^+\rangle - \langle L, R\rangle.$$

So if both encodings encode the same value, the filled-in correlation is a possible output of $\mathcal{O}$; the conditions $A \neq 0$ and $\widetilde{B} \neq 0$ hold since $L \neq 0$ and $R^+ \neq 0$. In this case, the filled-in transcript has exactly the honest conditional distribution of the refresh view, given its two encodings: for fixed encodings, each pair $(M, \widetilde{M})$ gives exactly one honest execution, and a counting argument shows that all these executions are equally likely. If the encodings encode different values, as happens in a hybrid where only one of them has been switched to an encoding of 0, the same formulas give the view of a refresh that uses a correlation with offset $\delta = \langle L^+, R^+\rangle - \langle L, R\rangle$. Such offset correlations are only a proof device, and they have the same support size as the real one. Crucially, the left view is a function of $(L, L^+, M, \widetilde{M})$, the right view is a function of $(R, R^+, M, \widetilde{M})$, and the public

matrices are independent of both encodings. Hence, with the other encoding fixed, every leakage query during a refresh can be converted into a leakage query on one share of the target encoding.

Inverse multiplication. The multiplication gadget is handled in the same way. Fix its input encodings and its output encoding $(L^{(w)}, R^{(w)})$, put $x = L_1^{(w)}$, sample $U, V \leftarrow \mathbb{K}^{\nu^2-\nu}$, and set

$$L' = L^{(w)} \,\|\, U, \qquad R' = \Big(-R^{(w)} + e_1 x^{-1}(c - \langle U, V\rangle)\Big) \,\|\, V.$$

This is a pair of dimension ν^2 that compresses to $(L^{(w)}, R^{(w)})$: the front of L' is $L^{(w)}$, the backs of L' and R' are U and V, so that $t = \langle U, V\rangle$, and the compression formula returns $-\big(-R^{(w)} + e_1 x^{-1}(c - t)\big) + e_1 x^{-1}(c - t) = R^{(w)}$. Also, $\langle L', R'\rangle = c - \langle L^{(w)}, R^{(w)}\rangle$, as it should be for the refreshed tensor pair. The inverse refresh of dimension ν^2 then joins the tensor input $(L^{(0)} \otimes L^{(1)}, R^{(0)} \otimes R^{(1)})$ to (L', R'). The back U of L' and the matrices are independent of the encodings, and the back V of R' is local to Right. If the target is an input encoding, say $(L^{(0)}, R^{(0)})$, the left view is computed from $L^{(0)}$ and the fixed $L^{(1)}$, and similarly on the right. The only public value that depends on a target encoding is the single field element x of the output encoding (the same holds for translation by a public constant, which also uses $x = L_1$). The protocol publishes x tagged with the identifier of the encoding, and every later publication of it for the same encoding repeats the same value. Thus, in the reduction to LRS security, one additional leakage query of $\lceil N \rceil$ bits suffices to reconstruct the public data of all gadgets incident to an encoding.

Opening the output. The public output gadget does not fit this pattern: opening an encoding publishes its whole left share, not a single field element. If the opened encoding were a target of the hybrids, the reduction to LRS security would have to produce this share, which needs about $(\nu - 1)N$ bits of information about the target, more than the LRS bound allows. So we never open a target: the public output gadget first refreshes the output into a fresh pair, and opens only that pair. In the simulation, this pair is sampled directly from the public output y, and joined to its predecessor by the inverse refresh.

Some technicalities. To make the inverse refresh and inverse multiplication above give exactly the honest distribution of each gadget's view, given its input and output encodings, we make two small changes to the DF gadgets. First, every stored encoding is distributed as the *stable encoder* $\mathsf{IPEnc}_\nu^\dagger$, which samples a uniform encoding of its value subject to $L_1 \neq 0$ and a nonzero right tail $R_{2..\nu} \neq 0$. These two conditions are exactly the two public checks of the refresh in Section 2.3 – when a gadget commits, its output is a fresh sample of this distribution, independently of its input encodings, which is why we call it stable. Second, each gadget is run only once, and always for the same rounds in the same public order: if a check fails, the remaining rounds still take place but do nothing. A failed gadget is not retried, both components discard the new candidate and keep the encodings they had before the gadget. So the number and order of rounds never depend on secret data. Both changes add only a negligible error, since each condition or check fails with probability $O(1/Q)$ and $Q \geq 2^\kappa$. Also, since leakage functions see the complete view of the active component in a round, including its sampler coins, we use Lemma 4 to reconstruct the local randomness conditioned on the sampled algebraic values. Together with the gadget-specific algebraic reconstruction and deterministic replay, this yields the complete conditional activation views. Please see Section 4 and Appendix C for details.

2.6 Supporting Polynomially Many Adaptive Queries

The main challenge is to compose the above over polynomially many adaptively chosen queries, while the leakage accumulates across rounds. If a single physical encoding remained present for

polynomially many rounds, the small leakage from each round could add up and reveal it. To handle this, we follow each physical encoding across its lifetime.

Encodings and their lifetime. We distinguish physical encodings by their creation events instead of their values. Each successful refresh produces a fresh encoding of the same logical value, and each successful copy produces two fresh encodings. Each newly committed encoding receives a fresh identifier and an encoding retained across rounds or query boundaries keeps its identifier until it is consumed or replaced. We call a committed, stable, unopened encoding of dimension ν that is stored between gadgets a *closed encoding*; encodings of public values, the temporary encoding of dimension ν^2 inside a multiplication, and the opened pair of the public output gadget are not closed. The closed encodings are the targets of the hybrids. By the public schedule, every logical value that survives an instruction is refreshed into a new encoding, and the old encoding is deleted together with all the local coins, consumed messages and correlation outputs. A value that is needed later stays in its component, and can leak in every intervening round even while it is dormant. So we must count every round *incident* to an encoding: a round of a gadget that produces or consumes it, or a round in which it, or workspace that depends on it, is kept in memory.

We show that, except with negligible probability, every closed encoding has at most H incident rounds, as follows. We divide the program into *blocks*, one for each instruction, that is, each G_c gate, translation, copy or output step. A block consists of the gadgets that execute the instruction (including the copies of its operands), together with the refreshes that carry the other live registers. Between two consecutive blocks, every live value has exactly one committed encoding. We call this point a *stable boundary*. Each block has at most $B(W)$ gadgets, where $B(W)$ is a constant that depends only on the number W of live registers and on the fixed gadget interfaces, and each gadget takes at most $h_* = 6$ rounds, where a copy is counted as two refreshes. An encoding created in a block is consumed or refreshed by the end of the next block, so it is incident to rounds of at most two blocks, that is, to at most $2h_*B(W)$ rounds. Adding one query-entry and one query-exit round gives

$$H = 2h_*B(W) + 2 = O(1).$$

This bound is independent of κ, ν, f and the number of queries. In particular, it also covers the persistent key encoding that crosses from one query to the next (see Equation (13)).

Hybrid over encodings. We number the closed encodings of the whole execution, across all queries, in the order of their creation. These identifiers remain fixed across hybrids. In hybrid j, the encodings with the first j identifiers are random encodings of 0, and their incident gadgets are filled in, with offsets, by the inverse refresh and inverse multiplication of Section 2.5 (copy, translation and the public output gadget are refreshes plus deterministic steps). Two consecutive hybrids differ only in the encoding assigned to one identifier e and its incident gadget views. To show that they are indistinguishable, we build a reduction to the LRS security of a single encoding. The reduction gets leakage access to the two shares of an LRS challenge, which encodes either the value of e or 0, and uses this challenge as the encoding e. It samples all other values itself, including the other encodings, the public matrices and the backs U, V. For a leakage query f in a round incident to e, it hardwires these values, composes f with the same-side formulas above, and forwards the result as a leakage query on the corresponding share of the challenge. It answers leakage queries in all other rounds by itself. If the first left coordinate x of the challenge is needed, the reduction obtains it with one more leakage query of $\lceil N \rceil$ bits. Thus, the total leakage on e is at most $H\lambda + \lceil N \rceil$, which is allowed whenever

$$H\lambda + \lceil N \rceil \leq \Lambda_\nu,$$

where Λ_ν is the leakage bound of the LRS scheme. An encoding that crosses a query boundary keeps one identifier and is replaced in a single hybrid due to which it is never charged twice.

The public output gadget fits into this argument as explained in Section 2.5. Its input is a closed encoding, and hence a target and the sealing refresh is one of its incident gadgets. The opened pair is not a closed encoding, so it is never a target. In every hybrid, the opened pair is sampled directly from the public output, and the sealing refresh is filled in by the inverse refresh from the input (L, R) to this pair; once the input has been switched to an encoding of 0, this uses a correlation with offset $y - \langle L, R \rangle = y$. As for an ordinary refresh, the left view uses only the left shares and the public matrices, and similarly on the right, so the input still has local reconstruction.

The simulator. After all encodings have been replaced, all closed encodings, including the retained secret-state encoding, encode 0, every gadget view is given by the inverse refresh and inverse multiplication, and every opened pair carries its prescribed public output. This last hybrid uses neither s nor any value computed from s, except the outputs y_i. So it can be generated by a sampler that only uses the pairs (x_i, y_i), and this sampler is our simulator. We stress that this is not an honest evaluation under a dummy key: a dummy key s' would give the outputs $f(s', x_i)$, and not the prescribed answers $y_i = f(s, x_i)$. Further, the simulator keeps the exact secret-state encoding E_i that it returned for query i (whose marginal is an encoding of 0), and passes it verbatim to query $i + 1$. Sampling a new encoding of 0 instead would ignore the bias that earlier leakage introduces in its conditional distribution. Since the reconstruction randomness for future gadget invocations is sampled only after the released prefix, the next input can be any efficient function of this prefix.

The leakage-rate. An IP encoding over the full field tolerates leakage of a little less than half of one share: for any constant $c > \frac{5}{2}$, it tolerates $\Lambda_\nu = (\frac{\nu}{2} - c)N - 1$ bits [32, 33]. We take the largest λ with $H\lambda + \lceil N \rceil \leq \Lambda_\nu$, so that $\lambda \approx (\frac{\nu}{2} - c - 1)N/H$. Dividing by the size $\nu\lceil N \rceil$ of one persistent share, the leakage-rate tends to $\frac{\nu/2-c-1}{H\nu}$ as $N \to \infty$, and this tends to $\frac{1}{2H}$ as ν grows. The factor $\frac{1}{2}$ comes from the inner-product extractor, and the factor $\frac{1}{H}$ from accumulation, since each encoding is charged for every round incident to it. Since $N \geq \kappa$, for any constant $\rho < \frac{1}{2H}$, a large enough constant ν gives leakage-rate at least ρ for all large enough κ. The statistical errors from storage and from failed checks and those from the samplers are all negligible since $Q \geq 2^\kappa$. Please refer to Theorem 2 and Theorem 3 for the formal statement and proof.

Removing the leak-free sampler using iO. We can also replace the sampler by the two-party protocol of [22], using iO and LWE, but we must take care of leakage from the protocol itself. The simulated views must be consistent not only with the sampled outputs, but also with the receiver's coins for generating its encryption key. These coins remain in memory and may leak while the sender's randomness is replaced by an explanation. The explanation-security definition in [22] does not explicitly retain them. We use the concrete Sahai–Waters construction [87], whose explanation security holds even given the receiver's secret key, and include the key-generation coins as part of that key for the proof. This leaves the encryption algorithm and the real protocol unchanged. We also count the protocol's extra rounds in the lifetime of each encoding. Fortunately, there are only constantly many per gadget, so the leakage-rate remains constant. Please refer to Appendix E for the details.

2.7 Instantiations

All our applications are obtained from Theorem 3 by choosing the operation f and the base game G; please refer to Section 6 for the details. The compiler does not add any computational assumption, and H depends only on the fixed machine $\mathcal{U}$ and the public schedule. So every application gets leakage-rate at least ρ, for every constant $\rho < \frac{1}{2H}$. For each primitive, we only need a suitable base scheme. Its secret-key operation must be total (it returns an output, possibly $\perp$, on every input) and deterministic, with public polynomial bounds on its time, space and input and output lengths.

It must use a fixed secret, and the base security game must give the adversary an oracle for it, with public inputs and outputs. For PKE, we need an IND-CCA2 secure base scheme. PRFs, MACs and signatures only need one-way functions, which follow from LWE, or from constant-noise LPN with polynomial hardness.

PKE. As explained in Section 2.2, we take $f = \mathsf{Dec}_0$ and G the IND-CCA2 game of P_0. For P_0, we use known standard-model IND-CCA2 schemes: from LWE, the KDM-CCA2 secure scheme of [68], which is in particular IND-CCA2 secure, from low-noise LPN, the tag-based scheme of [63], and from constant-noise LPN with $2^{\omega(\sqrt{n})}$ hardness, the tag-based scheme of [95]. The two tag-based schemes are converted to IND-CCA2 PKE by the transformation of [62], which uses a strongly one-time signature. Such signatures follow from one-way functions, and hence from the same assumption. In each case, we fix canonical parsing and deterministic rounding or decoding, so that Dec_0 is total and deterministic, and maps every malformed ciphertext to $\perp$.

PRFs. For a PRF F, we take $f(k, x) = F_k(x)$ and G the real PRF oracle game. Our CML PRF notion (Definition 8) asks for one simulator Sim with two properties. First, given the actual outputs $F_k(x_i)$, Sim produces a view that is statistically close to the real one – this is output simulation for the compiled evaluator, as in the proof of Theorem 3. Second, the real view is indistinguishable from an ideal experiment, in which the outputs come from a random function R and Sim produces the leakage. For this, a PRF distinguisher runs the adversary with Sim, and gives Sim the answers of its own oracle. With oracle F_k, this is the simulated real experiment and with oracle R, it is exactly the ideal experiment. So Sim must also run in polynomial time on random outputs, but its statistical guarantee is needed only for the actual outputs.

MACs. We use the PRF-based MAC $\mathsf{Tag}_0(k, m) = F_k(\mathsf{encode}(m))$, for an injective encoding encode of messages of publicly bounded length, and protect both tagging and verification. Both operations use the same key, so we compile them together with a public selector, $f(k, (1, m)) = \mathsf{Tag}_0(k, m)$ and $f(k, (2, m, t)) = \mathsf{Vfy}_0(k, m, t)$. The tag recomputed during verification, and its comparison with t, stay inside the protected computation, and only the acceptance bit is published. Since verification queries also leak, the base game G must contain a verification oracle as well as a tagging oracle. An accepting verification query on a message that was not tagged counts as a forgery. Key generation and the challenger's final forgery check are not compiled, and do not leak. We show directly that the PRF-based MAC is secure in this game, up to a negligible extra term for guessing tags.

Signatures. One-way functions also give EUF-CMA signatures [84], and the standard PRF-based tree construction makes the signer stateless. If the resulting signer is randomized, we derandomize it by a standard PRF technique (see [17, footnote 5]). We then compile the complete deterministic signer, including the evaluation of the PRF with G the EUF-CMA game.

3 Definitions and Preliminaries

We use κ for the security parameter and $\mathsf{negl}(\kappa)$ for a negligible function. All adversaries are classical probabilistic polynomial-time algorithms unless stated otherwise. For a finite field $\mathbb{K}$, write $Q = |\mathbb{K}|$ and $N = \log_2 Q$. Vectors are column vectors, and $\langle u, v\rangle = u^T v$ is their standard inner product. $\mathrm{GL}_\nu(\mathbb{K})$ denotes the set of invertible $\nu \times \nu$ matrices over $\mathbb{K}$.

3.1 Leakage from Computation and Memory

The secret state is divided into ℓ private memory components $P_1, \ldots, P_\ell$, together with a common memory component, denoted Common, for public input and communication. In each protected

round, the computation activates one private component and accesses the common memory. The adversary may make adaptive leakage queries to any private component, including inactive components. The total output length of all leakage queries in that round is at most λ. Our constructions use $\ell = 2$, with private memory components Left and Right.

Definition 1 (λ-leakage game)**.** Let $P_1, \ldots, P_\ell$ be the private memory components and let Common be the common memory. In one round, an adaptive adversary submits polynomially many sequential queries, where the jth query is (f_j, I_j), for $j \in \text{poly}(\kappa)$, $I_j \in [\ell]$ and

$$f_j : \{0,1\}^* \times \{0,1\}^* \longrightarrow \{0,1\}^{\lambda_j}.$$

It receives $f_j(P_{I_j}, \text{Common})$. The queries can depend on all earlier answers, but $\sum_j \lambda_j \leq \lambda$.

Definition 2 (Leakage from computation)**.** Let Π be a computation divided into t protected rounds, with private memory components $P_1, \ldots, P_\ell$ and Common. In round i, Π activates one private component P_{j_i} and Common. Its *activation view* contains the input state, fresh randomness tape, received and sent messages, every intermediate value used by the local calculation, and the candidate output state.

The adversary's leakage is captured by the λ-leakage game in definition 1 s.t. during that activation, the active private component is the full activation view of P_{j_i}; and each inactive private component P_k, for $k \neq j_i$, consists of its complete current state, including its persistent secret state and any retained protocol workspace waiting for a later activation. All leakage queries in the round, across all private components, share one total λ-bit budget.

After these queries, Π performs the state transition and deletions prescribed by its protocol schedule. Values retained for later activations remain in their respective private components and continue to be leakable, including while those components are inactive. Deleted values are absent from subsequent states and cannot be queried in later rounds. The output of the interaction contains every leakage answer, all public messages, and the final public output.

3.2 Leakage rate

Between the rounds of a multi-activation computation, each component stores one persistent share of say $s(\kappa)$ bits. If the leakage bound in one protected round is $\lambda(\kappa)$ bits, the leakage rate is

$$\rho(\kappa) = \frac{\lambda(\kappa)}{s(\kappa)}.$$

This is the persistent-state convention used in independent-leakage literature. Temporary work space can be larger, but it does not remain between queries. We always state separately how many rounds one physical encoding remains present, because leakage can accumulate over those rounds.

This convention matters quantitatively. A multiplication gadget refreshes an encoding of dimension ν^2 and a local activation may contain $\Theta(\nu^4)$ field elements of matrix workspace, in addition to its full calculation trace. We do not prove a rate bound normalized by the largest active private memory. Every use of "rate" in the main theorems uses the between-query persistent-share denominator above.

3.3 Statistical simulation of protected computation from efficient adversaries

Let Π be a stateful protected computation with public input x, private persistent state, and public output y. Each invocation follows the protected-round schedule and leakage interaction of definition 2.

Definition 3 (Output simulation)**.** The computation Π is *output-simulatable* if there exists a stateful probabilistic polynomial-time online simulator Sim, depending only on Π and its public parameters, such that the view of every probabilistic polynomial-time interactive leakage adversary $\mathcal{A}$ in the following interactions is statistically close.

- In the real interaction, $\mathcal{A}$ chooses each public input x_i and interacts with Π according to definition 2, receiving the public messages, leakage answers, and public output y_i.
- In the simulated interaction, Sim receives the public input x_i and its actual output y_i before simulating invocation i. It releases public messages in the prescribed order and answers each leakage query online, using the same query interface and per-round total leakage budget as in definition 2. It retains private simulation state between invocations.

The real and simulated complete views of $\mathcal{A}$ have negligible statistical distance in κ. These views include $\mathcal{A}$'s random coins, chosen inputs, public messages, leakage queries and answers, public outputs, and stopping decision. The simulator receives no logical secret state or real private memory. Its private simulation state is not revealed to $\mathcal{A}$.

4 Full-Field Encodings and the LFH Functionality

This section recalls the part of the Dziembowski–Faust compiler that we use [32, 33]. We make two small changes that are useful below. First, we use one padded attempt of every gadget and take a public failure branch instead of restarting. Second, stable encodings are conditioned to have a nonzero right tail. These changes have negligible cost and make every conditional distribution in the reconstruction proof exact.

4.1 Inner-product based Leakage-Resilient Storage (LRS)

Let $\mathbb{K}$ be a finite field, let $Q = |\mathbb{K}|$, and put $N = \log_2 Q$; let d be a parameter. A value $z \in \mathbb{K}$ is stored as two vectors $(L, R) \in \mathbb{K}^d \times \mathbb{K}^d$ such that $\langle L, R\rangle = z$. The left component stores L and the right component stores R. The usual DF12 full-field encoder, denoted $\mathsf{IPEnc}^0_d(z)$, samples $L_1 \leftarrow \mathbb{K}^*$ and all coordinates $L_2, \ldots, L_d, R_2, \ldots, R_d$ uniformly, and sets

$$R_1 = L_1^{-1}\left(z - \sum_{i=2}^{d} L_i R_i\right).$$

Our stable encoder is

$$\begin{aligned} \mathcal{E}^{\dagger}_d(z) &= \Big\{(L, R) : L_1 \neq 0,\ R_{2..d} \neq 0,\ \langle L, R\rangle = z\Big\}, \\ (L, R) &\leftarrow \mathsf{IPEnc}^{\dagger}_d(z), \end{aligned} \tag{1}$$

where the pair is uniform in the displayed set. It is sampled by choosing $L_1 \in \mathbb{K}^*$, $L_{2..d} \in \mathbb{K}^{d-1}$, and $R_{2..d} \in \mathbb{K}^{d-1} \setminus \{0\}$ uniformly, and solving for R_1. One component share has $d\lceil N\rceil$ bits.

Lemma 1 (Full-field storage bound [32, 33])**.** *Let $Q = |\mathbb{K}|$, $N = \log_2 Q$, and let d be the encoding dimension. Fix constants $a > 1$ and $3/2 < b < d/2$, and assume that Q is in the range required by the inner-product extractor (in particular, $Q = \Omega(d)$).*

For any $z_0, z_1 \in \mathbb{K}$, consider the Λ_d-leakage game of definition 1, where the two private components contain the fixed shares $(L, R) \leftarrow \mathsf{IPEnc}^{\dagger}_d(z_\beta)$ for $\beta \in \{0, 1\}$, and the common memory is

empty. An adversary may adaptively query leakage from either share, making at most u queries and obtaining at most Λ_d bits of leakage in total, where

$$\Lambda_d = \left(\frac{d}{2} - a - b\right) N - 1. \tag{2}$$

Then the statistical distinguishing advantage between the experiments $\beta = 0$ and $\beta = 1$ is at most

$$\begin{aligned} \epsilon_d(u) &= 2u\left(Q^{3/2-b} + Q^{1-a}\right) + 2\tau_d \\ &\le 2u\left(Q^{3/2-b} + Q^{1-a}\right) + 2Q^{-1} + 2Q^{-(d-1)}, \end{aligned} \tag{3}$$

where τ_d is defined in eq. (4).

Proof: We reduce to the inner-product storage theorem of [32, 33]. Let $\mathsf{IPEnc}_d^{\mathsf{DF}}(z)$ sample L uniformly from $\mathbb{K}^d \setminus \{0\}$ and R uniformly subject to $\langle L, R\rangle = z$. Our encoder $\mathsf{IPEnc}_d^{\dagger}(z)$ is precisely this distribution conditioned on $L_1 \neq 0$ and $R_{2..d} \neq 0$. Put

$$\begin{aligned} \tau_d &= 1 - \frac{(Q-1)Q^{d-1}}{Q^d - 1}\left(1 - Q^{-(d-1)}\right), \\ \Delta\Big(\mathsf{IPEnc}_d^{\mathsf{DF}}(z), \mathsf{IPEnc}_d^{\dagger}(z)\Big) &= \tau_d \le Q^{-1} + Q^{-(d-1)}. \end{aligned} \tag{4}$$

The DF11 inner-product storage theorem implies, as a conservative consequence, leakage $((1/2 - \delta)d - a)N - 1$ and error $2u(Q^{3/2-d\delta} + Q^{1-a})$ for $\mathsf{IPEnc}_d^{\mathsf{DF}}$: compared with the bound stated in [32], this form counts the leakage on both shares in one total budget rather than per share, subtracts one bit, and includes the factor u in the error; the first two changes only weaken the guarantee, and so does the third for $u \ge 1$, while for $u = 0$ no leakage is released and the bound is trivial. Put $b = d\delta$ and use (4) at the two ends of the storage experiment. □

A useful concrete choice is $a = 5/4$ and $b = 7/4$. It gives

$$\Lambda_d = \left(\frac{d}{2} - 3\right) N - 1, \qquad \epsilon_d(u) \le 4uQ^{-1/4} + 2Q^{-1} + 2Q^{-(d-1)}. \tag{5}$$

The more general form will be used to optimize the rate.

4.2 The inputless Leak-Free Hardware (LFH)

For a field $\mathbb{K}$, $\delta \in \mathbb{K}$ and $d \in [\mathbb{N}]$, define:

$$\begin{aligned} \mathcal{O}'_{\delta,\mathbb{K},d} &:= \left\{ (A, \widetilde{A}), (B, \widetilde{B}) \;\middle|\; \begin{smallmatrix} ((A,\widetilde{A}),(B,\widetilde{B})) \in (\mathbb{K}^d)^2 \times (\mathbb{K}^d)^2 \\ \langle A,B\rangle + \langle \widetilde{A},\widetilde{B}\rangle = \delta,\ A \neq 0,\ \widetilde{B} \neq 0 \end{smallmatrix} \right\} \\ \mathcal{O}'_{\mathbb{K},d} &:= \mathcal{O}'_{0,\mathbb{K},d} \end{aligned}$$

Lemma 2 (Exact uniform sampling)**.** *For every $\delta \in \mathbb{K}$, there exists an exact polynomial-time sampler for the probability distribution $\mathcal{O}'_{\delta,\mathbb{K},d}$.*

Proof: Sample $A \leftarrow \mathbb{K}^d \setminus \{0\}$, $\widetilde{A} \leftarrow \mathbb{K}^d$, and $\widetilde{B} \leftarrow \mathbb{K}^d \setminus \{0\}$. Let i be the first nonzero coordinate of A. Sample B_j uniformly for $j \neq i$ and set

$$B_i = A_i^{-1}\left(\delta - \langle \widetilde{A}, \widetilde{B}\rangle - \sum_{j \neq i} A_j B_j\right). \tag{6}$$

The sampler chooses A, $\widetilde{A}$, and $\widetilde{B}$ with respective probabilities $(Q^d-1)^{-1}$, Q^{-d}, $(Q^d-1)^{-1}$, and chooses the prescribed coordinates $(B_j)_{j\neq i}$ with probability $Q^{-(d-1)}$. Since $A_i \neq 0$, it is invertible in $\mathbb{K}$, and the correlation constraint uniquely determines B_i through (6). Thus every allowed tuple has probability

$$\frac{1}{(Q^d-1)^2 Q^{2d-1}},$$

independent of the tuple and of δ. This proves uniformity. □

Remark 1. The real protocol uses only $\mathcal{O}'_{\mathbb{K},d} := \mathcal{O}'_{0,\mathbb{K},d}$. It writes $(A, \widetilde{A})$ privately to Left and $(B, \widetilde{B})$ privately to Right, and nothing to the common memory. Every call is fresh and inputless. The dimensions used in our constructions are $d = \nu$ and $d = \nu^2$.

4.3 Concrete DF gadgets

Recall that, all vectors are column vectors by notation. We now describe the gadgets that enable leakage-resilient computation. The randomized algorithms use sampling with conditional randomness reconstruction fixed in lemma 4. Thus "view" always means the complete activation view of definition 2, including sampler coins, rejected sampler proposals, and the deterministic arithmetic trace. It does not mean only the displayed algebraic variables. Please refer to appendix C for the full details.

1. **Refresh Gadget.** Let (L, R) be a stable encoding of secret z. A regular refresh from (L, R) to (L^+, R^+) uses a correlation from $\mathcal{O}'_{\mathbb{K},d}$ and locally sampled public matrices $M, \widetilde{M} \in \mathrm{GL}_d(\mathbb{K})$ s.t. $ML = A$ and $\widetilde{M}R^+ = \widetilde{B}$. Then:

$$\begin{aligned} A &= ML, & R^+ &= R + M^T B, \\ \widetilde{B} &= \widetilde{M}R^+, & L^+ &= L + \widetilde{M}^T \widetilde{A}. \end{aligned} \tag{7}$$

   We run one padded attempt. The refresh is a fixed sequence of activations: left, right, left, and then a commit step. Each activation runs for a fixed public time, and every activation takes place even after a failed check (as an empty, padded activation), so the number and order of rounds are fixed in advance. The right component announces in Common whether $R^+_{2..d} \neq 0$, and, if this check passed, the left component announces whether $L^+_1 \neq 0$; the candidate is committed only when both checks pass. Otherwise, both components take the same fixed failure branch, which is public since the announcements are: they delete the candidate and the correlation outputs, and keep the last committed encoding. (Inside the copy, translation and public output gadgets, this commit is part of the gadget's joint commit.) There is no retry. One fixed attempt keeps the schedule public and makes the conditional distributions in the reconstruction proof exact; unbounded retries would also keep the input encoding beyond the exposure bound H. A failure keeps the old encoding; this happens with negligible probability and is counted in the bad event BadExp_H of definition 4. Conditioned on commitment, the output is exactly $\mathsf{IPEnc}^{\dagger}_d(z)$ when the input encodes z. The failure probability of one refresh is

$$q_d + (1-q_d)Q^{-1}, \qquad q_d = Q^{-(d-1)}. \tag{8}$$

   In particular it is at most $2/Q$ for $d \geq 2$. Please see appendix C.2 for the exact protocol.
2. **Multiplication Gadget.** Let $(L^{(i)}, R^{(i)})$ be stable encodings of secrets s_i and $c \in \mathbb{K}$ be a public constant. The multiplication gadget for $G_c(s_0, s_1) = c - s_0 s_1$ first locally computes tensors:

$$(L^{(0)} \otimes L^{(1)}, R^{(0)} \otimes R^{(1)})$$

and refreshes them at dimension $D = d^2$, obtaining (L', R'). Put $x = L'_1$. The left party sends x and $L'_{d+1..D}$. The right party puts

$$t = \langle L'_{d+1..D}, R'_{d+1..D} \rangle, \quad L^{(w)} = L'_{1..d}, \quad R^{(w)} = -R'_{1..d} + e_1 x^{-1}(c - t).$$

It commits only if $R_{w,2..d} \neq 0$. Conditioned on this check and the internal refresh succeeding, $(L^{(w)}, R^{(w)})$ is exactly $\mathsf{IPEnc}_d^\dagger(c - s_0 s_1)$. The added compression failure has probability

$$\frac{Q^{D-d} - 1}{Q^{D-1} - 1} \leq Q^{-(d-1)}. \tag{9}$$

Please see Appendix C.3 for the exact protocol.

3. **Copy Gadget.** Given a stable encoding (L, R) of z, the copy gadget produces two fresh encodings $(L^{(0)}, R^{(0)})$ and $(L^{(1)}, R^{(1)})$ of z. It runs two refresh attempts, each starting from the same original pair (L, R), with independent correlations and independent local randomness. The original pair is retained until both attempts finish. If both succeed, the parties commit both output pairs and delete the original pair; otherwise they discard both candidates and retain (L, R). The attempts follow a fixed padded schedule, without retries. Conditioned on success, the two outputs are independent samples from $\mathsf{IPEnc}_d^\dagger(z)$. Ignoring the negligible sampler-failure probability, copy fails with probability

$$1 - \left((1 - Q^{-(d-1)})(1 - Q^{-1})\right)^2 \leq \frac{4}{Q}.$$

Please see Appendix C.4 for the exact protocol.

4. **Translation Gadget.** For a public constant $c \in \mathbb{K}$, the translation gadget transforms a stable encoding (L, R) of z into a fresh stable encoding of $z + c$. The left component sends $x = L_1$ to the right component, which computes

$$\widehat{R} = R + e_1 x^{-1} c,$$

where e_1 is the first standard basis vector. The inverse exists because $L_1 \neq 0$. Moreover,

$$\langle L, \widehat{R} \rangle = \langle L, R \rangle + L_1 x^{-1} c = z + c,$$

and $\widehat{R}_{2..d} = R_{2..d} \neq 0$. The parties then perform one refresh attempt on $(L, \widehat{R})$, retaining the original pair (L, R) until commitment. On success, they commit the refreshed output and delete the original pair and temporary data. On failure, they discard the temporary data and retain (L, R). Conditioned on success, the output is exactly $\mathsf{IPEnc}_d^\dagger(z + c)$. The failure probability is given by (8), apart from negligible sampler failures. Please see Appendix C.5 for the exact protocol.

5. **Public output gadget.** Given a closed encoding $E = (L, R)$ of z, this gadget makes z public. It runs one dedicated sealing refresh to a fresh pair $E_{\text{out}} = (L^{(\text{out})}, R^{(\text{out})})$. Only on commitment does the left component send $L^{(\text{out})}$ to the right component, which publishes $y = \langle L^{(\text{out})}, R^{(\text{out})} \rangle$. The pair is transparent gadget data, is deleted after opening, and is never reused as a closed encoding. The closed input is consumed on success; on failure it is retained and no candidate pair is opened. See Appendix C.6 for the exact protocol and reconstruction.

Commit and cleanup discipline. Each primitive computes its candidate shares in fresh private memory. Both components commit their new shares before either share is used by a later instruction.

At commitment, the parties delete the replaced shares, all remaining local coins, private copies of consumed common messages, and the consumed correlation outputs, as prescribed by definition 2. Values needed by a later activation remain in their component's private state and remain leakable until their scheduled deletion. When a logical value requires two subsequent uses, the compiler invokes the copy gadget to produce two fresh encodings. On a failure branch, the parties delete the uncommitted data and retain the last committed state.

4.4 Simulating Leakage of DF Gadgets

Recall that a stable encoding of z is a pair $(L, R) \in \mathcal{E}_d^\dagger(z)$, as defined in (1), with L and R stored in the left and right components, respectively. We call such an encoding *committed* once both components have installed their respective shares as the current encoding, following the checks and cleanup discipline of the preceding subsection. Here, commitment refers only to this protocol state update. If a gadget's candidate encoding fails its prescribed checks, the candidate is discarded and the previously committed encoding is retained.

Encoding Notations. We first state the semantics of some useful keywords for encodings.

1. **Identifier.** We distinguish physical encodings by their creation events, rather than by their share values. Each successful refresh produces a fresh encoding of the same logical value, and each successful copy produces two fresh encodings of that value. Each newly committed encoding has a fresh identifier, which may be reserved while the encoding is still a candidate. An encoding retained across activations or query boundaries keeps the same identifier until it is consumed or replaced. Distinct encodings retain distinct identifiers even if their share pairs happen to coincide. In a hybrid comparison, identifiers remain fixed while the share pairs assigned to them may change.
2. **Closed encoding.** It is a committed, stable d-dimensional encoding that remains unopened. Public-input encodings, the temporary d^2-dimensional encoding inside multiplication, and the fresh encoding used for a public output opening are not closed encodings.
3. **Incident bundle of an encoding.** For a closed encoding with identifier e, its *incident bundle* consists of its producing (or generating) and at most two consuming gadget views, all physical leakage answers involving e, and all public messages of those gadgets. This includes activations in which its shares or incident reconstruction-dependent workspace are retained in inactive memory, as well as activations across invocation boundaries.

We now explain how to simulate leakage from the DF gadget implementations described above. The key ingredient is *local reconstruction*: for a target closed encoding, the left component's incident gadget execution views can be sampled using only its left share, without knowing the target encoding's right share. Symmetrically, the right component's views can be sampled without knowing the target encoding's left share. These are complete activation views, including local randomness and intermediate computations, so leakage queries can be answered by evaluating the adversary's leakage functions on the simulated (reconstructed) views.

The public transcript is handled separately. Its potentially long matrices and tails are sampled independently of the target encoding's two shares; all remaining public data depending on those shares are obtained from a bounded number of field elements. Once these public data are supplied, local reconstruction reduces leakage during gadget execution to leakage on one share of the target encoding at a time. This allows us to apply the inner-product storage bound of lemma 1.

Definition 4 (Protected DF program)**.** A protected DF program implements a deterministic function $f(s, x)$ with fixed logical private state $s \in \mathbb{K}$, public input x, and public output $f(s, x)$, subject to the following conditions.

1. **Public schedule.** Each invocation is a polynomial-size, publicly clocked program of the DF gadgets of section 4. Its gadget types, wiring, and activation and deletion schedule are determined by public information, rather than secret-dependent branching. Gadgets follow their prescribed checks, fixed one-attempt schedules, failure branches, and cleanup rules. On successful execution, secret-dependent public outputs are released through the public output gadget; publicly determined outputs may be returned directly.
2. **Initialization and retained state.** Initialization samples $E_0 \leftarrow \mathsf{IPEnc}_d^{\dagger}(s)$ and deletes initialization workspace before leakage begins. Each successful invocation preserves the logical state s and retains only a fresh encoding of s, obtained through the prescribed copy and refresh gadgets, for the next invocation. On a prescribed failure branch, it retains the most recent valid encoding of s and deletes other private workspace.
3. **Encoding use.** Closed encodings have dimension d and unique identifiers. Each is consumed by at most two gadgets. An encoding retained across invocation boundaries keeps its identifier and exact share pair until consumed or replaced.

The program has *exposure bound* H if, for every PPT adversary interacting with the program over polynomially many adaptive invocations, the following event occurs with negligible probability:

$$\mathsf{BadExp}_H := \left\{ \begin{array}{l} \text{there exists a closed encoding whose complete} \\ \text{incident bundle meets more than } H \text{ protected} \\ \text{activations} \end{array} \right\}.$$

The negligible bound is over the randomness of the execution and may depend on the adversary and the polynomial invocation bound. In particular, a failed gadget may retain an existing encoding beyond its prescribed lifetime. Such executions are included in BadExp_H.

Theorem 2 (Output simulation for protected DF programs)**.** *Let $\mathbb{K}$ be an efficiently sampleable and computable finite field of size Q, with $N = \log_2 Q = \Omega(\kappa)$ and $N = \mathrm{poly}(\kappa)$. Let $d \geq 2$ be polynomially bounded in κ, and fix constants $a > 1$ and $3/2 < b < d/2$ as in lemma 1.*

Every protected DF program of definition 4 with exposure bound H is output-simulatable in the sense of definition 3 over polynomially many adaptive invocations, provided

$$H\lambda + \lceil N \rceil \leq \Lambda_d. \tag{10}$$

The statistical simulation error includes an additive $\Pr[\mathsf{BadExp}_H]$ term, which is negligible by the exposure-bound condition. The stateful PPT online simulator receives only each public input and its actual reachable public output. It preserves every previously released public and leakage prefix and passes the exact retained simulation-state encoding to the next invocation without resampling it.

Please refer to appendix C.7 for the full proof and to appendix C.8 for a comparison of the proof techniques with prior work.

5 A Universal Constant-Rate CML Compiler

This section shows how to implement any deterministic polynomial-time computation as a protected DF program in the sense of definition 4, with a constant exposure bound. The main challenge is to keep this bound independent of the computation's running time and space usage. We do so by packing machine configurations into field elements, evaluating each transition with a constant number of live encoded values, and refreshing every encoding that must survive a protected instruction.

We first establish this implementation in lemma 3. We then apply the output-simulation theorem of theorem 2 and derive a constant leakage rate over polynomially many adaptive invocations. The resulting compiler applies to any deterministic secret-key operation; we state the general security-transfer result in section 5.2 and present PKE as an application in section 6.1.

5.1 Compiling General Computations into Constant-Width DF Programs

We construct the program by composing four constant-width DF subprograms. The description below is at the logical register level: each repeated use is implemented with the copy and carry operations of definition 4. The exposure schedule is justified separately in appendix D.

Lemma 3 (Constant-width protected DF simulation)**.** *Let M be a total deterministic uniform polynomial-time machine with public time and space bounds. There is a polynomial-time choice of an extension field $\mathbb{K}$ and, for each polynomially bounded encoding dimension $d \geq 2$, a uniform polynomial-size protected DF program computing $f(S, x) = M(s, x)$, where S is the prescribed packing of s. The program has an absolute constant number W of live logical registers and an absolute constant physical-encoding exposure bound H. These constants are independent of M, the security parameter, d, and the number of invocations.*

Proof:
Extension-field packing of machine configurations. Fix a deterministic universal single-tape machine U with a fixed finite alphabet, finite control, and local transition rule. We use a standard clocked cellular simulation of U, in which the machine's tape contents, head position, and control state are represented by a one-dimensional configuration over a fixed alphabet. Each cell is encoded by a constant number of bits, and the next configuration is determined by applying a fixed local rule to a constant-radius neighbourhood of each cell.

Let T be a public polynomial bound on the number of simulated steps, and let D be a public bound on the number of bits in a padded configuration. We take $D \geq \kappa$ and choose a prime $D < p < 2D$. Set

$$\mathbb{K} = \mathbb{F}_{p^D}, \qquad Q = p^D.$$

The field, a public basis $(\beta_0, \ldots, \beta_{D-1})$, and its trace-dual basis $(\beta_0^\vee, \ldots, \beta_{D-1}^\vee)$ can be constructed uniformly in polynomial time [89].

A binary configuration $b = (b_0, \ldots, b_{D-1}) \in \{0,1\}^D$ is packed into the single field element

$$X = \mathsf{Pack}(b) := \sum_{i=0}^{D-1} b_i \beta_i \in \mathbb{K}.$$

By trace duality, every cell bit can be recovered as

$$b_i = \mathrm{Tr}_{\mathbb{K}/\mathbb{F}_p}(\beta_i^\vee X).$$

Thus, although a configuration may contain polynomially many cells, it occupies only one logical field register.

The key observation is that a complete machine transition can be implemented using a constant number of such registers. We scan the packed configuration in public order, extract the constant-size neighbourhood needed for each output cell, evaluate the fixed local transition rule, and append the resulting bit to a fresh packed configuration. The old and new configurations occupy two field registers; all other workspace is constant.

We now describe the DF subprograms implementing this procedure. All loops have public bounds and are unrolled. The permitted arithmetic gates are $G_c(u, v) = c - uv$. Multiplication, negation, and public-constant translations are implemented using these gates. No secret-addition gate is required. Repeated logical uses are implemented using the prescribed DF copy and carry operations.

The lookup program Lookup_F. For a fixed arity k and public function $F : \mathbb{F}_p^k \to \mathbb{F}_p$, enumerate all tuples $a \in \mathbb{F}_p^k$ in public order. For each tuple compute

$$I_a(z) = \prod_{j=1}^{k} \big(1 - (z_j - a_j)^{p-1}\big), \qquad C_a = 1 - I_a(z)(1 - F(a)).$$

Maintain a single accumulator A, initially 1, and update $A \leftarrow AC_a$. Since precisely one I_a equals 1, the final accumulator equals $F(z)$, including when $F(z) = 0$. This *multiplicative one-hot selection* is important: merging arbitrary weighted values by $1 - (1 - u)(1 - v)$ would not be correct. Only the k carried inputs, the accumulator, and constant scratch space are live. Each input is copied forward when used, and the previous physical encoding is retired. Since k is constant and p is polynomial, the program is uniform, polynomial-size, and constant-width. In particular, Lookup_+ evaluates addition on two elements of $\mathbb{F}_p$ without a secret-addition gate.

The cell-extraction program $\mathsf{Extract}_i$. On a packed configuration X, compute $Z = \beta_i^\vee X$ and set $A \leftarrow 0$. For each $t \in \mathbb{F}_p$ in public order, execute

$$V \leftarrow 1 + tZ, \quad N \leftarrow V^{(Q-1)/(p-1)}, \quad C \leftarrow -t^{p-2}N, \quad A \leftarrow \mathsf{Lookup}_+(A, C).$$

The norm exponent uses a fixed square-and-multiply program with constant scratch space. The sequential trace accumulator retains Z, A and a constant number of temporaries; it does not store all p summands. By the trace-dual identity and the finite-field moment identity (proved in section 2.4),

$$\mathrm{Tr}(Z) = -\sum_{t \in \mathbb{F}_p} t^{p-2}\, \mathrm{N}(1 + tZ), \qquad \mathsf{Extract}_i(X) = \mathrm{Tr}(\beta_i^\vee X) = b_i. \tag{11}$$

A carried encoding of X is returned for subsequent extractions.

The packed-append program Append_i. For a bit b and packed accumulator Y, execute

$$\begin{aligned} Y^+ &\leftarrow Y + \beta_i, \\ F_0 &\leftarrow 1 - (1 - b)(1 - Y), \\ F_1 &\leftarrow 1 - b(1 - Y^+), \\ Y' &\leftarrow F_0 F_1. \end{aligned} \tag{12}$$

The translation by β_i is public-constant arithmetic. If $b = 0$, then $Y' = Y$; if $b = 1$, then $Y' = Y + \beta_i$. This is a constant-size, constant-width DF subprogram computing $Y + b\beta_i$ without secret addition.

The transition program Step. Use the fixed-radius cellular simulation of U described in appendix B.1. On input X, initialize $Y = 0$ and scan the positions $i = 0, \ldots, D-1$. At position i, call $\mathsf{Extract}_j(X)$ for the constant number of neighbours j, compute the next bit b_i' by the fixed Boolean transition rule (using G_1 as NAND), and execute $Y \leftarrow \mathsf{Append}_i(Y, b_i')$. Discard the neighbourhood bits and proceed. Inductively, after position i the accumulator is $\sum_{j \le i} b_j' \beta_j$; thus $\mathsf{Step}(X)$ is the packed next configuration. The old X, the new Y, and the constant-width subprogram workspace are the only live registers.

The machine program $\mathcal{P}_M$. Its persistent logical state is the packed secret S; its public invocation input is x. Using the prescribed disjoint configuration layout, initialize the universal-machine configuration from the secret packing and the public configuration contribution P_x. Then apply Step for the public clock T, extract the prescribed output cells, and release their public decoding. Halted configurations self-loop. Preserve the unchanged logical secret S for the next invocation. The fixed universal machine and the four subprograms above give an absolute constant live width W, and all instructions are uniformly generated in polynomial time.

Protection and exposure. Compile the logical register operations using the protected DF interfaces of definition 4. Every live value, including S, the trace accumulator, and idle inputs, is carried through fresh physical encodings on the public schedule. By lemma 12, the fixed register schedule ensures that, in every execution without a gadget-check or sampler failure, every physical encoding has exposure at most

$$H_{\mathsf{cert}} \le 2h_* B(W) + 2 = 12B(W) + 2 = O(W) = O(1), \tag{13}$$

where $h_* = 6$ and $B(W) = O(W)$. By the same lemma, failures have negligible probability, so the program has exposure bound H_{cert} in the sense of definition 4. Here, the last equality follows because the universal compiler has constant live logical width $W = O(1)$. No particular optimized numerical value of H is needed. □

The degree D covers the whole padded work configuration, and can be larger than the original key length. Thus the compiler can have a polynomial secret-state expansion. The leakage rate below is measured against this compiled persistent state.

5.2 Expanding the Compiler to support Oracles

Let $f : \mathcal{S} \times \mathcal{X} \to \mathcal{Y}$ be computed by a total deterministic uniform machine with public polynomial bounds on its running time and space and on the lengths of s, x, and $f(s, x)$. Malformed inputs are parsed canonically and give $\perp$. Let Gen_f output public information pub and a secret $s \in \mathcal{S}$. Write $S \in \mathbb{K}$ for the prescribed packing of s in the reserved state cells of the padded configuration.

We use the constant $H = H_{\mathsf{cert}}$ from (13) below.

Definition 5 (f-oracle games and their CML extensions)**.** Fix a total deterministic function $f : \mathcal{S} \times \mathcal{X} \to \mathcal{Y}$ and a generator Gen_f. An *f-oracle game* G is a security experiment with a one-bit outcome in which the challenger samples $(\mathsf{pub}, s) \leftarrow \mathsf{Gen}_f(1^\kappa)$, keeps s fixed throughout the experiment, gives pub to the adversary, and provides a designated oracle $\mathcal{O}_f$. On input x, this oracle evaluates a public predicate $\mathsf{Allowed}(\tau, x)$, where τ is the game transcript so far. If the predicate is false, it returns $\perp$ without evaluating f; otherwise it records x and returns $f(s, x)$. The game may have additional oracles, challenges, and a winning condition, which may depend on s.

For a protected implementation Π_f, its *λ-CML extension* $\mathsf{G}^{\mathsf{CML}}_\lambda[\Pi_f]$ replaces the allowed $\mathcal{O}_f$ evaluations by $\Pi_f.\mathsf{Eval}(x)$ with the leakage interaction of definition 2. The adversary receives the public output and complete leakage transcript. Each allowed query is recorded even if its protected execution follows an exceptional branch; a disallowed query returns $\perp$ before any protected computation or leakage. All additional oracles, challenges, and the winning condition remain unchanged and leak-free. Leakage transcripts are not included in τ.

For an adversary $\mathcal{A}$ of $\mathsf{G}^{\mathsf{CML}}_\lambda$, let $\mathcal{B} = \mathcal{A}^{\mathsf{Sim}_f}$ be the following adversary in G. It runs $\mathcal{A}$ and forwards pub, every query to a further oracle, every challenge and the final output unchanged. It forwards every $\mathcal{O}_f$ query x of $\mathcal{A}$ to its own oracle. If the answer is $\perp$ because $\mathsf{Allowed}(\tau, x) = 0$, it returns $\perp$. Otherwise it receives $y = f(s, x)$, gives (x, y) to Sim_f, and answers the public messages and leakage queries of $\mathcal{A}$ using the simulator.

Table 2: Compiled protected DF implementation Π_f (using lemma 3).

Initialization. Sample $(\mathsf{pub}, s) \leftarrow \mathsf{Gen}_f(1^\kappa)$, pack s as $S \in \mathbb{K}$, and sample $(L^{(S)}, R^{(S)}) \leftarrow \mathsf{IPEnc}^\dagger_\nu(S)$. Publish pub, store $L^{(S)}$ in Left and $R^{(S)}$ in Right, and delete all other private initialization data before leakage begins.

$\Pi_f.\mathsf{Eval}(x)$**.** On a valid public input x, create two protected fresh copies of S. Carry one through fresh refreshes as the retained secret. On the other, run the protected DF program supplied by lemma 3 for $M(s, x) = f(s, x)$, with public input x and encoding dimension $d = \nu$.
Apply the prescribed sealed public-output procedure to its final closed output, decode, and return the result. Delete working data and retain a fresh valid encoding of S.
On malformed input, execute the fixed protected identity-copy path, refresh S, and return $\perp$.
On a prescribed protected-operation failure, delete uncommitted and working data, return $\perp$, and retain the most recent valid encoding of S.

Theorem 3 (Universal DF compiler and security transfer)**.** *Given a total deterministic uniform machine computing f, with public polynomial clock and space bounds, the compiler produces a protected DF implementation Π_f. Let H be the absolute constant of (13). For every constant $\rho < 1/(2H)$, choose the parameters as in the proof, so that $\lambda = \lambda_{\nu,\zeta}$ has rate at least ρ against one persistent share. In the $\mathcal{O}'_{\mathbb{K},\nu}, \mathcal{O}'_{\mathbb{K},\nu^2}$ hybrid model, the following hold without computational assumptions:*

1. ***Output simulation.*** *The compiled protected DF program Π_f computes f correctly over polynomially many invocations, except with negligible probability, and satisfies the output-simulation requirement of definition 3 for polynomially many adaptive invocations, with statistical error $\epsilon_{\mathsf{sim}}(\kappa) = \mathsf{negl}(\kappa)$.*
2. ***Security transfer for f-oracle games.*** *For every f-oracle game G satisfying definition 5 and every PPT adversary $\mathcal{A}$ in $\mathsf{G}^{\mathsf{CML}}_\lambda[\Pi_f]$, there is a PPT adversary $\mathcal{B} = \mathcal{A}^{\mathsf{Sim}_f}$ in G making exactly one $\mathcal{O}_f$ query for each such query by $\mathcal{A}$, on the same input and in the same order, such that*

$$\left|\Pr[\mathsf{G}^{\mathsf{CML}}_\lambda[\Pi_f](\mathcal{A}) = 1] - \Pr[\mathsf{G}(\mathcal{B}) = 1]\right| \leq \epsilon_{\mathsf{sim}}(\kappa).$$

Proof: **Construction.**

By lemma 3, the program in table 2 has the prescribed protected DF schedule and exposure bound H; its protected operations use fresh ideal-correlation calls as specified in that lemma and the DF-program definition.

Proof of Item 1: output simulation. The compiler is uniform and takes the description of the deterministic machine for f, together with its public clock and space bound. By lemma 3, the program used in table 2 has the required protected DF schedule, with constant exposure bound H. Consequently, theorem 2 applies directly whenever

$$H\lambda + \lceil N \rceil \leq \Lambda_\nu. \tag{14}$$

It supplies a stateful PPT online simulator Sim_f for polynomially many adaptive invocations, receiving only each public input and its actual reachable public output. Prefix preservation, retention of the exact simulation-state encoding across invocation boundaries, and leakage from retained workspace are already part of that theorem; no separate temporal reconstruction lemma is needed. Write ϵ_{sim} for the total statistical simulation error.

Leakage parameters and rate.

Fix a constant $\zeta > 0$ in lemma 1 and take

$$a = 1 + \zeta, \qquad b = \frac{3}{2} + \zeta.$$

Then

$$\Lambda_\nu = \left(\frac{\nu}{2} - \frac{5}{2} - 2\zeta\right) N - 1. \tag{15}$$

The quantitative bound in (26), together with (3), gives negligible total statistical error: there are polynomially many closed encodings and alternating leakage queries, and $N = \Omega(\kappa)$. The prescribed gadget failures and fixed-clock sampler failures also have negligible total probability. Set

$$\lambda_{\nu,\zeta} = \left\lfloor \frac{(\nu/2 - 5/2 - 2\zeta)N - 1 - \lceil N \rceil}{H} \right\rfloor. \tag{16}$$

Choose the fixed ν so that the numerator is positive for all sufficiently large N; in particular it is enough that $\nu/2 > 7/2 + 2\zeta$. The persistent share of one component has $\nu\lceil N \rceil$ bits. Hence

$$\lim_{N\to\infty} \frac{\lambda_{\nu,\zeta}}{\nu\lceil N \rceil} = \frac{\nu/2 - 7/2 - 2\zeta}{H\nu}. \tag{17}$$

For every constant $\rho < 1/(2H)$, first choose any fixed $\zeta > 0$ and then choose the fixed constant ν large enough that the limit in (17) exceeds ρ. The finite-N ratio is therefore at least ρ for all sufficiently large κ. Equivalently, the supremum over fixed ν and $\zeta > 0$ is $1/(2H)$.

The value $1/(2H)$ is the supremum of this DF storage analysis for the stated schedule. The leading storage bound is below one half of one share and one encoding may be charged in H rounds. The one reconstructed field element affects the finite value of ν, but not the supremum. This is not an impossibility bound for other compilers. In particular, no decimal value such as 0.10 is justified unless a complete implementation first fixes and proves the corresponding value of H.

Correctness. Logical correctness follows from lemma 3. The additional failure probability is $O(1/Q)$ per protected primitive. The field has $N = D \log_2 p = \Omega(\kappa)$ bits, and polynomially many calls make only polynomially many primitive calls, so the total additional error is negligible. The output-simulation argument above applies because the chosen leakage parameters satisfy (14).

Proof of Item 2: security transfer. For security, fix s and all coins of the challenger of G except those used inside Π_f. Combine $\mathcal{A}$ with the challenger's actions outside $\mathcal{O}_f$ into one interactive leakage adversary $\mathcal{A}'_s$ for Π_f. Its protected queries are the allowed $\mathcal{O}_f$ queries of $\mathcal{A}$. In $\mathsf{G}^{\mathsf{CML}}_\lambda$, $\mathcal{A}'_s$ interacts with the real protected computation. In G with $\mathcal{B}$, it interacts with Sim_f, which receives the actual outputs $f(s, x_i)$. By theorem 2 for Π_f, the complete views of $\mathcal{A}'_s$ in the two cases are within statistical distance ϵ_{sim}. This bound holds for every fixed s, because the DF and LRS arguments are pointwise in the logical private state (see the proof of theorem 2). It also holds although $\mathcal{A}'_s$ depends on s, because every step of that proof is statistical. The output bit of the game is a function of this view, s and the fixed coins, so the two probabilities differ by at most ϵ_{sim}; average over s and the coins.

Disallowed queries return $\perp$ before any protected computation in both games. Every allowed query, including one whose protected run fails, is recorded in both games, so the bookkeeping of G is identical. Finally, $\mathcal{B}$ is PPT because Sim_f is PPT and the leakage functions of $\mathcal{A}$ are efficient. □

Remark 2 (Scope of the transfer). The theorem requires that every protected computation is an $\mathcal{O}_f$ query of the base game, with a public input chosen by the adversary, an output given to the adversary, and a fixed secret. These conditions cannot simply be dropped.

1. *No oracle in the base game.* Let f be the decryption of Regev's IND-CPA PKE. The compiled decryption publishes its output, so leaky decryption of adversarial ciphertexts gives a decryption oracle, and Regev's scheme admits key recovery with a decryption oracle. So the base game must already contain the oracle; IND-CPA security of the base scheme does not suffice.
2. *Secret inputs.* A challenger computation on a secret input, such as the challenge encryption of IND-CPA symmetric encryption, is not an $\mathcal{O}_f$ query. The theorem leaves it unchanged and leak-free. It cannot be compiled in our model, because the input of Π_f is placed in common memory; compiling it would publish m_b.
3. *Fresh randomness.* f must be deterministic. Derandomizing with a PRF works for signatures (section 6.3) but not in general: for symmetric encryption, $\mathsf{Enc}_k(m; F_{k'}(m))$ is deterministic and is not IND-CPA secure.
4. *Evolving secret state.* s must be fixed, so stateful signatures and forward-secure schemes are not covered. Removing session state by derandomization can be insecure: if a Σ-protocol prover uses commitment randomness $r = F_{k'}(x)$, an adversary that obtains responses to two different challenges for the same x gets two accepting transcripts with one commitment, and special soundness gives the witness.

Several operations on the same secret are covered together. For $f_1, \ldots, f_k$ on the same s, put $f(s, (j, x)) = f_j(s, x)$ with a public selector j; the universal machine and hence H are unchanged.

6 Instantiations in the LFH Model

6.1 Public-Key Encryption

We instantiate the compiler with ordinary IND-CCA2-secure encryption schemes to obtain leakage-resilient PKE in the sense of definition 7. Let $H = H_{\mathsf{cert}}$ be the constant from the generic compiler's protected schedule.

Let $P_0 = (\mathsf{Gen}_0, \mathsf{Enc}_0, \mathsf{Dec}_0)$ be an ordinary IND-CCA2-secure PKE scheme with deterministic decryption and the public polynomial resource bounds required by the compiler. Instantiate table 2 with $\mathsf{Gen}_f = \mathsf{Gen}_0$ and $f(s, c) = \mathsf{Dec}_0(s, c)$.

Corollary 4 (PKE instantiations from LWE and LPN)**.** *Assume any one of the following:*

1. *The polynomial-modulus LWE assumptions used by [68].*
2. *Polynomial hardness of decisional low-noise LPN with noise $p = \Theta(1/\sqrt{n})$ and polynomially many samples.*
3. *For some fixed $0 < \mu \leq 1/10$, decisional constant-noise $\mathsf{LPN}_{\mu,n}$ is $2^{\omega(\sqrt{n})}$-hard in the quantitative sense of [95].*

For every fixed $\rho < 1/(2H)$, there exists an IND-CCLA2 secure PKE with leakage rate at least ρ for polynomially many adaptive decryption queries in the $\mathcal{O}'_{\mathbb{K},\nu}, \mathcal{O}'_{\mathbb{K},\nu^2}$ hybrid model. Its correctness error is negligible.

Proof: We first obtain an ordinary IND-CCA2 secure base scheme P_0 under each alternative assumption.

LWE. Use the standard-model KDM-CCA2 PKE of Libert, Nguyen, Passelègue and Titiu [68]. KDM-CCA2 security implies ordinary IND-CCA2 security by restricting the KDM functions to constants. Fix canonical parsing and deterministic rounding with least-representative tie breaking.

The resulting base scheme has polynomial-time deterministic decryption and negligible correctness error.

Low-noise LPN. Use the selective-tag weak-CCA tag-based encryption of Kiltz, Masny and Pietrzak [63, Thm. 2], followed by the TBE-to-PKE transformation of [62]. A fresh strongly one-time signature verification key serves as the tag, and the resulting ciphertext is signed. Low-noise LPN implies one-way functions and hence the required signatures. Choose the tag length to accommodate the verification key, and fix canonical parsing and deterministic code decoding. The transformation yields an IND-CCA2 secure base scheme with polynomial-time deterministic decryption and negligible correctness error.

Constant-noise LPN. Apply the TBE-to-PKE transformation of [62] to the selective-tag CCA tag-based encryption of [95]. The stated quantitative LPN hardness supplies the security of the tag-based scheme, as well as one-way functions and the required strongly one-time signatures. Choose the tag length to accommodate the verification key, and fix canonical parsing and deterministic decoding. The resulting base scheme is IND-CCA2 secure and has polynomial-time deterministic decryption and negligible correctness error.

Application of the transfer theorem. For each base scheme, supply public polynomial clock and space bounds for the complete decryption algorithm, with malformed inputs parsed canonically and mapped to $\perp$. Apply theorem 3 with

$$s = \mathsf{sk}_0, \qquad \mathsf{pub} = \mathsf{pk}_0, \qquad f(s, c) = \mathsf{Dec}_0(s, c),$$

and let G be the IND-CCA2 game of P_0. The logical secret key stays fixed, and f is total and deterministic with the required public resource bounds.

The public predicate $\mathsf{Allowed}(\tau, c)$ rejects exactly the challenge ciphertext after the challenge is issued. Such rejection occurs before any protected computation or leakage. Encryption remains unchanged; in particular, challenge encryption uses only the public key, the challenge messages, and fresh randomness. Thus $\mathsf{G}^{\mathsf{CML}}_{\lambda}$ is the IND-CCLA2 experiment of definition 7 for the compiled scheme.

By theorem 3, the compiled experiment is statistically close to the ordinary IND-CCA2 experiment with simulated leakage. The reduction makes the same decryption queries, in the same order, and never queries the challenge ciphertext. Ordinary IND-CCA2 security therefore gives IND-CCLA2 security with negligible statistical loss. The theorem also supplies leakage rate at least ρ and negligible additional correctness error over polynomially many calls. Combining this with the base scheme's negligible correctness error proves the claim. No additional computational hardness assumption is introduced by compilation. □

6.2 PRFs and MACs

We apply theorem 3 to deterministic PRF evaluation and MAC tagging, using the security definitions in definitions 8 and 9. Let $H = H_{\mathsf{cert}}$ be the constant from the generic compiler's protected schedule.

Assumptions for the instantiations. In this subsection and section 6.3, LWE means decisional LWE in a standard polynomial-modulus parameter regime sufficient for one-way functions. LPN means polynomial hardness of decisional constant-noise LPN with a fixed $\tau \in (0, 1/2)$ and polynomially many samples. Either assumption supplies one-way functions. The constructions of [51, 42] therefore supply ordinary polynomial-time PRFs.

No low-noise LPN assumption or subexponential hardness is needed for these symmetric-key and signature applications; the stronger hypotheses in items 2 and 3 of corollary 4 concern the public-key encryption instantiations.

6.2.1 Pseudorandom Functions Let

$$F : \{0,1\}^{\ell_k} \times \{0,1\}^{\ell_x} \longrightarrow \{0,1\}^{\ell_y}$$

be an ordinary PRF family with uniform keys and deterministic polynomial-time evaluation. We compile its evaluator to obtain a simulation-based CML PRF as defined in definition 8.

Corollary 5 (Two-state CML PRFs from LWE or LPN)**.** *Assume either LWE or LPN as specified above. For every fixed $\rho < 1/(2H)$, there exists a simulation-based λ-CML PRF in the sense of definition 8, with a stateful two-component evaluator supporting polynomially many adaptive queries. Its leakage rate is at least ρ per persistent share in the $\mathcal{O}'_{\mathbb{K},\nu}, \mathcal{O}'_{\mathbb{K},\nu^2}$ hybrid model, and its outputs equal $F_k(x)$ except with negligible total failure probability. The only computational assumption is the selected LWE or LPN assumption.*

Proof: Under either assumption, obtain one-way functions, then a PRG by [51], and an ordinary PRF by [42]. Polynomially bounded input and output lengths are supported by the standard tree construction and polynomial PRG stretch.

Apply theorem 3 with $s = k$, $f(k, x) = F_k(x)$, and G the real PRF oracle experiment, with every well-formed query allowed. The key is fixed, and PRF evaluation is total and deterministic with public polynomial resource bounds. The theorem gives an online simulator Sim_f such that the real compiled view is statistically close to the view obtained by supplying actual oracle outputs $F_k(x_i)$ to this simulator.

An ordinary PRF distinguisher $\mathcal{B}$ runs the adversary and Sim_f. On each query x_i, it asks its function oracle for y_i and supplies (x_i, y_i) to the simulator. With oracle F_k, this is the simulated real experiment. With a random-function oracle, it is exactly the ideal experiment. Therefore

$$\mathrm{Adv}_F^{\mathsf{sim\text{-}CML\text{-}PRF}}(\mathcal{A}) \le \mathrm{Adv}_F^{\mathsf{PRF}}(\mathcal{B}) + \epsilon_{\mathsf{sim}}.$$

The simulation error is negligible over polynomially many queries. Repeated inputs cause no difficulty: the oracle outputs repeat, while the compiled executions use fresh correlations and shares. The correctness and leakage-rate claims follow from theorem 3. □

We protect both tagging and online verification, using definition 9. The two operations share one protected representation of a fixed logical key. A public operation selector allows us to compile them together using remark 2.

Corollary 6 (Two-state CML MACs from LWE or LPN)**.** *Assume either LWE or LPN as specified above, and let $H = H_{\mathsf{cert}}$ be the constant of the generic compiler's protected schedule. For every fixed $\rho < 1/(2H)$, there exists a λ-CML-secure MAC with tagging and verification leakage in the sense of definition 9. It has deterministic logical tagging and verification and a stateful two-component implementation supporting polynomially many adaptive, interleaved tagging and verification requests. Its leakage rate is at least ρ per persistent share, and its total correctness error is negligible in the $\mathcal{O}'_{\mathbb{K},\nu}, \mathcal{O}'_{\mathbb{K},\nu^2}$ hybrid model.*

Proof: Under either assumption, obtain an ordinary PRF F as in the proof of corollary 5. Fix a publicly bounded message domain and an injective fixed-length encoding encode into the PRF input domain. Let $\ell_y = \Omega(\kappa)$ be the PRF output length. Define the base MAC by

$$\mathsf{Tag}_0(k, m) = F_k(\mathsf{encode}(m)), \qquad \mathsf{Vfy}_0(k, m, t) = [t = F_k(\mathsf{encode}(m))].$$

Verification rejects malformed inputs, including tags of length other than ℓ_y.

Security of the base game with verification queries. Let G be the ordinary game with tagging and verification oracles and the winning condition of definition 9, with no protected computations or leakage. In particular, an accepting verification query on a message not previously submitted for tagging already counts as a forgery. A PRF distinguisher simulates both oracles using its function oracle: it returns the function value for tagging, and only the equality bit for verification. It also uses this oracle to evaluate the final forgery. The logical key is never needed by this reduction.

Replace F_k by a lazy-sampled random function R. We claim that an adversary making at most q_{vfy} verification queries wins with probability at most $(q_{\mathsf{vfy}} + 1)2^{-\ell_y}$. To handle adaptive queries and repeated guesses, consider an auxiliary experiment that always answers 0 to verification queries on messages not yet tagged. It answers queries on already tagged messages by comparison with their known tags and answers tagging requests using R. Couple this experiment with the actual random-function game using the same R and adversary coins. Their transcripts coincide until the first accepting verification query on an untagged message.

In the auxiliary experiment, each candidate tag guessed for a currently untagged message is independent of that message's unrevealed random-function value: all earlier verification answers on that message were fixed to 0. Injectivity of encode ensures that tagging other messages does not reveal this value. Each such guess therefore matches with probability $2^{-\ell_y}$. The same statement holds for a final candidate on a message that remains untagged. A union bound over at most $q_{\mathsf{vfy}}+1$ candidates bounds the first accepting fresh-message verification query or a valid final forgery. Thus, for every PPT adversary $\mathcal{B}$ in G, there is a PPT PRF distinguisher $\mathcal{D}$ such that

$$\Pr[\mathsf{G}(\mathcal{B}) = 1] \le \mathrm{Adv}_F^{\mathsf{PRF}}(\mathcal{D}) + (q_{\mathsf{vfy}} + 1)2^{-\ell_y}.$$

Compiling both operations. Use a canonical, publicly bounded encoding of operation requests and define the total deterministic function

$$f(k, x) = \begin{cases} \mathsf{Tag}_0(k, m), & x = (1, m), \\ \mathsf{Vfy}_0(k, m, t), & x = (2, m, t), \\ \bot, & \text{otherwise.} \end{cases}$$

The outputs have a canonical encoding distinguishing tags, verification bits, and failure. Compile this single function using theorem 3; define protected tagging and verification by calling $\Pi_f.\mathsf{Eval}$ with the corresponding selector. Both operations use the same continuing state and fresh compiler randomness. In the base game, the allowed-query predicate accepts exactly the well-formed operation requests, using public parsing. The challenger maintains $\mathcal{Q}$ and win as in definition 9.

The hypotheses of theorem 3 hold: the key is fixed, inputs are public, both operations are total and deterministic with public polynomial resource bounds, and each operation's output is returned by an oracle already present in G. In particular, the simulator for a verification query receives only $((2, m, t), \mathsf{Vfy}_0(k, m, t))$, rather than the correct tag. Equality comparison is part of the protected machine; its recomputed tag is never released as an intermediate public output.

For a CML adversary $\mathcal{A}$, the transfer theorem produces a PPT adversary $\mathcal{B} = \mathcal{A}^{\mathsf{Sim}_f}$ in G and gives

$$\mathrm{Adv}_{\mathsf{MAC},\lambda}^{\mathsf{EUF\text{-}CML}}(\mathcal{A}) \le \mathrm{Adv}_F^{\mathsf{PRF}}(\mathcal{D}) + (q_{\mathsf{vfy}} + 1)2^{-\ell_y} + \epsilon_{\mathsf{sim}}.$$

The theorem preserves query inputs and order and records tagging requests even when a protected execution fails. It therefore preserves the freshness bookkeeping needed by the game. The first term is negligible by PRF security, the second because q_{vfy} is polynomial and $\ell_y = \Omega(\kappa)$, and the last by the transfer theorem. Correctness and the leakage rate also follow from that theorem. The public selector does not change the universal machine's constant W or the schedule constant H. □

Scope. Both accepted and rejected well-formed verification requests are protected and receive the prescribed leakage interaction. The leakage budget is that of definition 2 per round, rather than a single budget for an entire oracle call; an operation may require polynomially many refreshes. Messages must have a public polynomial length bound and an injective encoding into the PRF domain, or be handled by a separately justified variable-input PRF or base MAC. Initialization and the challenger's final forgery check remain leak-free. Leakage from operational verification is covered by $\mathcal{V}_\lambda$, including when that query itself constitutes a forgery.

6.3 Digital Signatures

We apply theorem 3 to deterministic signing to obtain CML-secure signatures as defined in definition 10. Randomized base signers are first derandomized as described below.

6.3.1 Signatures from LWE or LPN

Corollary 7 (Two-state CML signatures from LWE or LPN)**.** *Assume either LWE or LPN as specified in section 6.2. Let $H = H_{\mathsf{cert}}$ be the constant of the packed fixed-machine schedule. For every fixed $\rho < 1/(2H)$, there exists a λ-CML-secure signature scheme in the sense of definition 10, with stateful two-component signing supporting every polynomial number of adaptive signing requests, in the $\mathcal{O}'_{\mathbb{K},\nu}, \mathcal{O}'_{\mathbb{K},\nu^2}$ hybrid model. Its verification key stays fixed, verification is public and unchanged by compilation, its leakage rate is at least ρ per persistent share, and its correctness error is negligible. The only computational assumption is the selected LWE or LPN assumption.*

Proof: Either assumption supplies one-way functions and hence ordinary EUF-CMA secure signatures by [84]. Use the standard PRF-based tree implementation to obtain a stateless polynomial-time base signer with public polynomial resource bounds; the node-generation PRF is also obtained from the same one-way functions. If it is randomized, derandomize it with an independent ordinary PRF key using the argument below; this PRF follows from the same assumption by [51, 42]. Let Σ_0 denote the resulting deterministic base scheme and s its full secret, including the PRF key. No additional assumption is introduced by derandomization.

Apply lemma 3 to $M(s, m) = \mathsf{Sign}_0(s, m)$. Encode the packed signing key as in table 2. On each request, run the fixed-clock protected program, carry a separately refreshed encoding of the signing key, seal the signature with a fresh refresh, open only the resulting transparent pair, and retain a fresh encoding of the unchanged logical key. All surviving registers are carried and erased under the same public schedule. Consequently eq. (13) bounds each encoding's lifetime by H, except with negligible probability (lemma 12), including request boundaries. Choose the field and encoding parameters as in section 5.2; the storage budget $H\lambda + \lceil \log_2 |\mathbb{K}| \rceil \le \Lambda_\nu$ then holds. Correctness follows from the packed evaluation and the negligible total probability of primitive failure over polynomially many requests.

For security, this is theorem 3 with $f = \mathsf{Sign}_0$ and G the EUF-CMA game of Σ_0, with every request allowed; the request list of G contains every allowed request, including failed ones. We spell out the reduction. It receives the base verification key and a signing oracle, but not the signing key. On an adaptive request m_i, it obtains $\sigma_i \leftarrow \mathsf{Sign}_0(s, m_i)$ from that oracle and supplies (m_i, σ_i) to the online simulator of theorem 2. That lemma concerns deterministic protected programs and their actual outputs; its reconstruction does not require a PKE challenge or a decryption-specific property. It therefore simulates this signing program, including the shares of its retained secret-state encoding, without s. The total statistical simulation error ϵ_{sim}, including exceptional paths, is negligible. Verification is unchanged, and an unqueried-message forgery in the simulated experiment is a base EUF-CMA forgery. Thus, for the constructed reduction $\mathcal{B}$,

$$\mathrm{Adv}^{\mathsf{EUF\text{-}CML}}_{\Sigma}(\mathcal{A}) \le \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Sigma_0}(\mathcal{B}) + \epsilon_{\mathsf{sim}}.$$

This proves the claim for the defined post-initialization leakage model. □

Randomized base signers. The deterministic hypothesis can be met before compilation using a standard PRF derandomization, also discussed in [17, footnote 5]. Fix a public message-length bound and an injective padded message encoding. Add an independent PRF key k to the logical secret key and define

$$\mathsf{Sign}_{\mathsf{det}}((s,k),m) = \mathsf{Sign}_0(s,m;F_k(\mathsf{encode}(m))),$$

where the PRF output supplies the publicly bounded signing randomness tape. Ordinary EUF-CMA security follows by replacing the PRF with a random function: distinct messages receive independent randomness tapes, and repeated messages receive the same signature. A base signing-oracle reduction implements this distribution by caching one signature per message. Compile the entire deterministic algorithm, including PRF evaluation and both keys. The signature corollary applies with the PRF key and all intermediate values protected. It does not assume that a PRF seed or signing coins are leak-free, nor that an ordinary PRF alone resists leakage. The resulting bound adds the ordinary PRF distinguishing advantage to the base EUF-CMA advantage and the simulation error.

7 Acknowledgement of AI usage

The first version of the paper was entirely conceived and written by the human authors. Subsequently, we used ChatGPT Astra 6 as an auxiliary tool to proof-read, verify and generalize the results beyond PKE, which was our original focus. We also used Claude Opus 5.5 to make editorial changes such as improving the presentation and clarity of the manuscript, and for identifying potential gaps in technical arguments. All suggested revisions were independently reviewed and verified by the authors, who take full responsibility for the contents of this work.

References

[1] Miklos Ajtai. Secure computation with information leaking to an adversary. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 715–724, New York, NY, USA, 2011. Association for Computing Machinery.

[2] Adi Akavia, Shafi Goldwasser, and Carmit Hazay. Distributed public key schemes secure against continual leakage. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, page 155–164, New York, NY, USA, 2012. Association for Computing Machinery.

[3] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *Theory of Cryptography*, pages 474–495, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[4] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 113–134, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 427–455. Springer, Cham, August 2018.

[6] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with o(1/log n) leakage rate. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, page 586–615, 2016.

[7] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, Clara Paglialonga, and François-Xavier Standaert. Consolidating inner product masking. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 724–754. Springer, Cham, December 2017.

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

[9] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the glp lattice-based signature scheme at any order. *J. Cryptol.*, 37(1), November 2023.

[10] Guy Barwell, Daniel P. Martin, Elisabeth Oswald, and Martijn Stam. Authenticated encryption in the face of protocol and side channel leakage. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 693–723. Springer, Cham, December 2017.

[11] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, page 616–648, Berlin, Heidelberg, 2016. Springer-Verlag.

[12] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 397–426. Springer, Cham, August 2017.

[13] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 722–739. Springer, Berlin, Heidelberg, December 2011.

[14] Nir Bitansky, Dana Dachman-Soled, and Huijia Lin. Leakage-tolerant computation with input-independent preprocessing. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 146–163, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[15] Andrej Bogdanov, Yuval Ishai, and Akshayaram Srinivasan. Correction to: Unconditionally secure computation against low-complexity leakage. *Journal of Cryptology*, 35(2):7, Jan 2022.

[16] Elette Boyle, Shafi Goldwasser, Abhishek Jain, and Yael Tauman Kalai. Multiparty computation secure against continual memory leakage. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1235–1254, New York, NY, USA, 2012. Association for Computing Machinery.

[17] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 89–108. Springer, Berlin, Heidelberg, May 2011.

[18] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, page 501–510, USA, 2010. IEEE Computer Society.

[19] Suvradip Chakraborty, Goutam Paul, and C. Pandu Rangan. Efficient compilers for after-the-fact leakage: from CPA to CCA-2 secure PKE to AKE. Cryptology ePrint Archive, Paper 2017/451, 2017.

[20] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999.

[21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 2021, Part I*, volume 12726 of *LNCS*, pages 460–479. Springer, Cham, June 2021.

[22] Dana Dachman-Soled, Feng-Hao Liu, and Hong-Sheng Zhou. Leakage-resilient circuits revisited – optimal number of computing components without leak-free hardware. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 131–158, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[23] Ivan Damgård, Frédéric Dupuis, and Jesper Buus Nielsen. On the orthogonal vector problem and the feasibility of unconditionally secure leakage resilient computation. Cryptology ePrint Archive, Report 2014/282, 2014.

[24] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. Cryptology ePrint Archive, Paper 2009/399, 2009.

[25] Giovanni Di Crescenzo, Richard Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 225–244, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[26] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. *IACR Cryptol. ePrint Arch.*, page 196, 2010.

[27] Yevgeniy Dodis, Allison Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, page 688–697, USA, 2011. IEEE Computer Society.

[28] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 21–40. Springer, Berlin, Heidelberg, August 2010.

[29] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, page 21–40, Berlin, Heidelberg, 2010. Springer-Verlag.

[30] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 423–440, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[31] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 207–224, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[32] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In *Advances in Cryptology – ASIACRYPT 2011*, ASIACRYPT'11, pages 702–720, Berlin, Heidelberg, 2011. Springer-Verlag.

[33] Stefan Dziembowski and Sebastian Faust. Leakage-resilient circuits without computational assumptions. In *Proceedings of the 9th International Conference on Theory of Cryptography*, TCC'12, page 230–247, Berlin, Heidelberg, 2012. Springer-Verlag.

[34] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, page 293–302, USA, 2008. IEEE Computer Society.

[35] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *Theory of Cryptography*, pages 343–360, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[36] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 781–810. Springer, Cham, December 2017.

[37] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from computationally bounded and noisy leakage. *SIAM Journal on Computing*, 43(5):1564–1614, 2014.

[38] Eiichiro Fujisaki, Akinori Kawachi, Ryo Nishimaki, Keisuke Tanaka, and Kenji Yasunaga. Post-challenge leakage resilient public-key cryptosystem in split state model. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(3):853–862, 2015.

[39] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.

[40] Daniel Genkin, Yuval Ishai, and Mor Weiss. How to construct a leakage-resilient (stateless) trusted party. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 209–244, Cham, 2017. Springer International Publishing.

[41] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Berlin, Heidelberg, April 2012.

[42] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[43] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 59–79. Springer, Berlin, Heidelberg, August 2010.

[44] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. *SIAM J. Comput.*, 44(5):1480–1549, January 2015.

[45] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 567–597, Cham, 2017. Springer International Publishing.

[46] Vipul Goyal, Yuval Ishai, Hemanta K. Maji, Amit Sahai, and Alexander A. Sherstov. Bounded-communication leakage resilience via parity-resilient circuits. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 2016.

[47] Hannes Groß and Stefan Mangard. Reconciling d+1 masking in hardware and software. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 115–136. Springer, Cham, September 2017.

[48] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. Ls-designs: Bitslice encryption for efficient masked software implementations. In *FSE*, pages 18–37. Springer, 2014.

[49] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, May 2009.

[50] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *Proceedings of the 8th Conference on Theory of Cryptography*, TCC'11, page 107–124, Berlin, Heidelberg, 2011. Springer-Verlag.

[51] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[52] Carmit Hazay, Adriana López-Alt, Hoeteck Wee, and Daniel Wichs. Leakage-resilient cryptography from minimal assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 160–176. Springer, Berlin, Heidelberg, May 2013.

[53] Brett Hemenway, Rafail Ostrovsky, Silas Richelson, and Alon Rosen. Adaptive security with quasi-optimal rate. In *Proceedings, Part I, of the 13th International Conference on Theory of Cryptography - Volume 9562*, TCC 2016-A, page 525–541, Berlin, Heidelberg, 2016. Springer-Verlag.

[54] Clément Hoffmann, Benoît Libert, Charles Momin, Thomas Peters, and François-Xavier Standaert. Polka: Towards leakage-resistant post-quantum cca-secure public key encryption. In *Public-Key Cryptography – PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part I*, page 114–144, Berlin, Heidelberg, 2023. Springer-Verlag.

[55] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, page 219–235, Berlin, Heidelberg, 2013. Springer-Verlag.

[56] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.

[57] Yuval Ishai and Yifan Song. Protecting computations against continuous bounded-communication leakage. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, page 1887–1897, New York, NY, USA, 2025. Association for Computing Machinery.

[58] Anthony Journault and François-Xavier Standaert. Very high order masking: Efficient implementation and security evaluation. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 623–643. Springer, Cham, September 2017.

[59] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 41–58. Springer, Berlin, Heidelberg, August 2010.

[60] Yael Tauman Kalai and Leonid Reyzin. *A survey of leakage-resilient cryptography*, page 727–794. Association for Computing Machinery, New York, NY, USA, 2019.

[61] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 703–720. Springer, Berlin, Heidelberg, December 2009.

[62] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 581–600. Springer, Berlin, Heidelberg, March 2006.

[63] Eike Kiltz, Daniel Masny, and Krzysztof Pietrzak. Simple chosen-ciphertext security from low-noise LPN. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2014.

[64] Eike Kiltz and Krzysztof Pietrzak. Leakage resilient ElGamal encryption. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 595–612. Springer, Berlin, Heidelberg, December 2010.

[65] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Berlin, Heidelberg, August 1996.

[66] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999.

[67] Allison B. Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 725–734. ACM Press, June 2011.

[68] Benoît Libert, Khoa Nguyen, Alain Passelègue, and Radu Titiu. Simulation-sound arguments for lwe and applications to kdm-cca2 security. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 128–158, Cham, 2020. Springer International Publishing.

[69] Yehuda Lindell. A simpler construction of cca2-secure public-key encryption under general assumptions. In *Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, page 241–254, Berlin, Heidelberg, 2003. Springer-Verlag.

[70] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In Yuval Ishai, editor, *Theory of Cryptography*, pages 89–106, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[71] Daniel P. Martin, Elisabeth Oswald, Martijn Stam, and Marcin Wójcik. A leakage resilient mac. In *Proceedings of the 15th IMA International Conference on Cryptography and Coding - Volume 9496*, IMACC 2015, page 295–310, Berlin, Heidelberg, 2015. Springer-Verlag.

[72] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.

[73] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 427–437, New York, NY, USA, 1990. Association for Computing Machinery.

[74] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, page 18–35, Berlin, Heidelberg, 2009. Springer-Verlag.

[75] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked ind-cca secure saber kem implementation. In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 04 2021.

[76] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 96–108. ACM Press, October 2015.

[77] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology - EUROCRYPT 2009 - Volume 5479*, page 462–482, Berlin, Heidelberg, 2009. Springer-Verlag.

[78] Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - a block cipher allowing efficient higher-order side-channel resistance. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 2012*, volume 7341 of *LNCS*, pages 311–328. Springer, Berlin, Heidelberg, June 2012.

[79] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 142–159, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[80] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*, E-SMART '01, page 200–210, Berlin, Heidelberg, 2001. Springer-Verlag.

[81] Anup Rao. An exposition of bourgain's 2-source extractor. *Electronic Colloquium on Computational Complexity (ECCC)*, 14, 01 2007.

[82] Prasanna Ravi, Sujoy Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 307–335, 06 2020.

[83] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[84] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd ACM STOC*, pages 387–394. ACM Press, May 1990.

[85] Guy N. Rothblum. How to compute under AC^0 leakage without secure hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *LNCS*, pages 552–569. Springer, 2012.

[86] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 543–553. IEEE Computer Society, 1999.

[87] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013.

[88] Joachim H. Schipper. Leakage-resilient authentication. Master's thesis, Utrecht University, 2010. Available at https://ir.cwi.nl/pub/16862.

[89] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. In *29th FOCS*, pages 283–290. IEEE Computer Society Press, October 1988.

[90] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *DCC*, 71(1):57–81, 2014.

[91] Alvy Ray Smith, III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, 1971.

[92] François-Xavier Standaert, Olivier Pereira, and Yu Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 335–352, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[93] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. *Leakage Resilient Cryptography in Practice*, pages 99–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[94] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. Cryptology ePrint Archive, Paper 2021/849, 2021.

[95] Yu Yu and Jiang Zhang. Cryptography with auxiliary input and trapdoor from constant-noise LPN. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 214–243. Springer, Berlin, Heidelberg, August 2016.

[96] Zongyang Zhang, Sherman S. M. Chow, and Zhenfu Cao. Post-challenge leakage in public-key encryption. *Theoretical Computer Science*, 572:25–49, 2015.

A Discussion on the CML Leakage Model

This section discusses the leakage model of section 3 and the reasons for its restrictions.

A.1 The Continual Memory Leakage (CML) model

The CML model considers side-channel information leakage from the intermediate values obtained during computation in addition to the secret key. As suggested by the name, the adversary is allowed continuous (and therefore unbounded) leakage – with the restriction that leakage is bounded in a given time interval (or round). The secret key needs to be necessarily refreshed between intervals, to make the leakages across time intervals difficult for the adversary to correlate. The public key, however, is never updated.

To avoid running into impossibilities, the model makes a crucial assumption: the required computation is divided into sub-components, each such sub-component uses its own distinct memory (see below for the relevance of sub-components). Then, in each round of the computation, the adversary is allowed to get bounded leakage on the current state of computation restricted as follows: each leakage function only takes as input a single sub-component (together with the public communication) and leaks from it. We note that the state of a sub-component includes not only its share of the secret key, but also the local randomness and the intermediate values of the current computation, as well as every value that it keeps for later use. More precisely, the leakage class is restricted as follows:

1. **Bounded Leakage:** leakage in each round is bounded to λ bits.
2. **Independent Leakage:** each sub-component leaks independently.
3. **Efficient Leakage:** the leakage function must be efficiently computable.

It is easy to justify the first restriction – most side-channel attacks in practice only leak a polylogarithmic amount of information per observation and typically need thousands of observations to recover a secret key. This is, for instance, true for differential-power based attacks. Thus, most works on leakage consider this a reasonable assumption. The other two restrictions are justified below.

On the Independent Leakage assumption. Due to the impossibility results by [8] (and explicitly stated in [44]), if a leakage function acts on the entire state of the secret, then it is not possible to get leakage-resilience of even 1 bit. Thus, the minimal number of components required for even feasibility is at least 2. While one may object that leakage is in general a global phenomena, it has been shown [72, 59, 43, 32, 33, 44] that many relevant global leakages are in fact computable from just local leakages. This is true for the prominent hamming weight based leakages (in power analysis) and more generally for any affine leakage. Hence, independent leakages still allows for a broad class of useful leakages. Further, the model also captures memory attacks (for example, the cold-boot attacks of [49]).

For the specific case of PKE, the natural way to model leakage-resilience is via an augmented security experiment where the adversary gets oracle access to leakage queries (in addition to decryption queries). If the adversary is only allowed oracle queries before the challenge ciphertext (called IND-CCLA1 in the literature), then secure schemes in the 1-component model are feasible. The works of [18, 27] are in this model, and give constructions from the k-lin assumption. However, we note that this is a significantly weaker definition – typical real-world attackers continue to have side-channel access to the decryption key after seeing ciphertexts, thus security in their sense is only meaningful for ephemeral keys, which is extremely inefficient in the public-key setting. Hence we do not consider this setting in this work.

Impossibility of Information-theoretic Leakages. The work of [27] shows that if unbounded (as opposed to efficient) leakage functions are allowed, then there exist adversaries that can recover the entire secret key with probability 1. We note that, in our work, the only computational assumption is the one underlying the primitive that we start from: the protection of the computation itself is information-theoretic (given the leak-free hardware discussed below), and the efficiency of the leakage functions is only used when we reduce to the security of the underlying primitive.

A.2 Related Work for Leakage-Resilient PKE, MAC, Signatures

Related work for PKE. Table 3 compares our PKE with prior PKE schemes in the CML model. As noted by [14, 22], the compilers of [33] and [44] are secure in this model and when instantianted with any PKE decryption, gives leakage-resilience. Juma and Vahlis [59] give a continuous leakage-resilient PKE in the 2-component OCL model from any CCA2 secure PKE and FHE, which is also secure in the CML model [22]. With an FHE scheme of hardness $2^{\ell(\kappa)}$, it tolerates $\ell(\kappa)$ bits of leakage per round, while one component stores an FHE ciphertext of size $\text{poly}(\kappa)$; so its leakage-rate is $\frac{1}{\text{poly}(\kappa)}$.

Other related work. In the CML model with a single leaking component, Brakerski et al. [18] and Dodis et al. [26] give public-key primitives, such as encryption and signatures, that tolerate a constant fraction of leakage between key updates, and Lewko, Lewko and Waters [67] and Dodis et al. [27] also tolerate a constant fraction during updates. With one component, leakage after the challenge ciphertext is impossible, so these schemes only allow leakage before the challenge. Post-challenge leakage was first studied by Halevi and Lin [50] for CPA security with bounded leakage from a split key; CCA2 security with bounded post-challenge split-state leakage is achieved

[3] post-challenge leakage is impossible with a single leaking component, please see [8, 44].

Work	Assumption	Leakage Rate	CCLA2	#Comp	Quantum Broken?
[59]	FHE + LFH	$\frac{1}{\text{poly}}$	Yes	2	No
[33]	PKE + LFH	$\frac{1}{\text{poly}}$	Yes	2	(same as PKE)
[44]	PKE	$\frac{1}{\text{poly}}$	Yes	poly	(same as PKE)
[18, 27]	k-lin	constant	No[3]	1	Yes
[67]	composite-order bilinear groups	constant	No[3]	1	Yes
This	PKE + LFH	constant	Yes	2	(same as PKE)

Table 3: PKE in the CML model. The work of [32] (constant rate and post-challenge leakage from DDH) is in the only-computation-leaks model and is not listed.

in [96, 38, 19], with rate $1-o(1)$ in [96, 19]. These works consider a bounded total amount of leakage on the stored key, with no refresh and no leakage from the decryption computation. Dziembowski and Faust [32] give CCA2 encryption with continual post-challenge leakage and constant rate, from DDH, in the only-computation-leaks model, where values that are not being computed on do not leak; our model also lets the inactive component leak.

Related work for PRF and MAC. Table 4 compares our PRF and MAC instantiations with prior work in the CML model. PRFs and MACs in the bounded-leakage model such as [52] are not shown; see appendix A.3 for a more detailed related work.

Work	Assumption	Leakage Rate	#Comp	Quantum Broken?
[33]	PRF/MAC + LFH	$\frac{1}{\text{poly}}$	2	(same as PRF/MAC)
[44]	PRF/MAC	$\frac{1}{\text{poly}}$	poly	(same as PRF/MAC)
This	PRF/MAC + LFH	constant	2	(same as PRF/MAC)

Table 4: PRF and MAC in the CML model.

Other related work. Dodis and Pietrzak [28] construct leakage-resilient PRFs from ordinary PRFs under fixed, non-adaptively selected leakage functions and independently leaking computation steps, with disjoint leakage and test-query inputs. Martin, Oswald, Stam and Wójcik [71] give a practical continual OCL MAC using bilinear groups in the random oracle model and give a generic-group analysis of leakage. An early OCL authentication construction of Schipper combines a leakage-resilient generator with an ordinary one-time MAC [88]. Pereira, Standaert and Vivek [76] give specialized MAC and encryption constructions from symmetric primitives using a limited leak-free component. Their protected-component requirement must be distinguished from the assumption that inactive memory does not leak; they do not establish our dormant-memory CML guarantee. Barwell, Martin, Oswald and Stam [10] study continuous leakage for nonce-based authenticated encryption, including composition with protocol leakage and misuse resistance. Their construction uses a sufficiently leakage-resilient PRF, with a concrete pairing-based instantiation. This also supplies related work for PRFs and authentication, although its authenticated-encryption interface differs from our PRF and tagging-only MAC experiments.

A.2.1 Related work for Digital Signatures. Table 5 compares our signature with prior work in the CML model. Signatures in the bounded-leakage models such as [61] are not shown. Malkin, Teranishi, Vahlis and Yung [70] give a standard-model signature scheme from SXDH with $1-o(1)$ signing-key leakage between updates, including signing computation. Key generation and each update have separate $O(\log \kappa)$ leakage budgets. Boyle, Segev and Wichs [17] give fully leakage-resilient

signatures and a continual-leakage extension from a continual leakage-resilient one-way relation. The verification key stays fixed, while the signing key is refreshed. Their continual construction inherits the leakage guarantees of that relation, both between updates and during refresh; its signing-leakage interface also protects signing coins and intermediate values. The relation constructions of Dodis, Haralambiev, López-Alt and Wichs [26], and the storage and relation constructions of Dodis, Lewko, Waters and Wichs [27], are relevant instantiations and building blocks from bilinear assumptions. For this instantiation, the resultant scheme only supports $O(\log \kappa)$ leakage during key-updates.

To the best of our knowledge, there are no instantiations for the required relation from post-quantum assumptions.

Work	**Assumption**	**Leakage Rate**	**#Comp**	**Quantum Broken?**
[33]	Base signature + LFH	$\frac{1}{\text{poly}}$	2	(same as base)
[44]	Base signature	$\frac{1}{\text{poly}}$	poly	(same as base)
[17]	CML one-way relation [26, 27] + proof-system machinery	constant *	1	Depends on instantiation
[70]	SXDH	$1 - o(1)$ *	1	Yes
This	Base signature + LFH	constant	2	(same as base)

Table 5: Digital Signatures in the CML model.
* both these schemes only support $O(\log \kappa)$ leakage during key updates.

Other related work. Faust, Kiltz, Pietrzak and Rothblum [35] obtain tree-based signatures in the OCL model from bounded-leakage three-time signatures; their per-signing budget is one third of the underlying total bound. Dziembowski and Faust [32] give specialized inner-product-based protection for identification and signing using refreshed encodings and independently leaking parts. This signing application, with the assumptions needed for its identification-to-signature transformation, is distinct from the later generic compiler [33].

A.3 A comparison of other leakage models with CML

In this section, we briefly describe other popular leakage-models in the literature and compare them with the CML model. We focus on PKE schemes, PRFs, MACs, and signatures, together with the generic compilers used to protect their evaluation. Please refer to [60] for a more comprehensive survey.

The Wire-Probing model. Introduced by [56], this was one of the first formal leakage-models and has been very influential both in theoretical and practical settings. Briefly, they model computation as being performed by a clocked circuit with an updatable secret state. The adversary is allowed to provide arbitrary inputs and observe their outputs and get leakage – the intermediate values of t wires chosen adaptively. Security requires this to be simulatable given only input/output queries. [56] give a general compiler that transforms any circuit to be leakage-resilient with perfect security.

Many subsequent papers build on this work: masking-friendly block ciphers [78, 48], efficient masking techniques [47, 45, 58], reducing randomness [11, 12, 36], etc. A series of works [7] (see references within) consider "inner-product" masking (similar to boolean masking). Because the circuit size in [56] increases by a factor of t^2, the leakage-rate is low (about $\frac{1}{t}$). If the choice of leaking wires is not completely upto the adversary, then this can be amplified to constant leakage-rate [1, 6].

In the post-quantum setting, the work of [9] give a signature scheme from lattice-based assumptions. We note that, the CML model completely subsumes wire-probing leakages. In particular,

consider the circuit for the decryption algorithm with sk as the secret state. This circuit is composed of sub-circuits corresponding to the computation of Left and Right, with the communication between them as the wires connecting the sub-circuits. Then, any wire-probing leakage to this circuit has a corresponding leakage-function query to the current state of either Left or Right for the appropriate round.

Bounded-Leakage and Bounded-Retrieval model. [3] introduced the notion of bounded memory leakage and construct public-key encryptions from exponential hardness. In this model, the adversary can request for an apriori bounded leakage on the secret key adaptively one bit at a time. [74] improve on this by giving a PKE scheme from any hash-proof system, thus getting security from polynomial-time assumptions such as DDH, Quadratic Residuosity, etc. Their scheme is optimally leakage-resilient to $1 - o(1)$ leakage-rate. Unfortunately, these schemes only support before-the-fact leakage. The work of [50] considers after-the-fact leakage by defining the notion of entropic leakage-resilience for PKE; this captures the intuition that as long as the entropy of the encrypted message is higher than the amount of leakage, the message still has some (pseudo) entropy left. They show that this notion is realized by the Naor-Segev constructions. Unfortunately, the theoretical restrictions on the leakage function are unsupported by the bitter reality that the key may eventually leak completely over time.

The bounded-retrieval model of [25, 31] assumes that the bound on the overall leakage is extremely large (polynomially in the security parameter). The secret key is necessarily larger than this bound, however the goal of constructions in this model is to make sure that the efficiency of the system does not degrade with the leakage bound. This allows the users to increase their secret key size flexibly, so as to protect against large amounts of leakage, without degrading other efficiency parameters. This model is motivated by various malware attacks, in which a persistent virus may transmit a large amount of private data to a remote attacker. The work of [4] give a leakage-resilient PKE scheme in this model. However, the key-sizes and the supported leakage-rates are quite bad. It is also unclear how secure the scheme of [4] is, in particular their IBE construction seems to have a bug (as noted in [60]).

For symmetric-key primitives, Hazay, López-Alt, Wee and Wichs [52] construct bounded-leakage-resilient weak PRFs and MACs from any one-way function. Their leakage bound can be any prescribed polynomial by enlarging the key; this is a bound on total key leakage, not a fresh allowance for each evaluation. Moreover, weak PRF security concerns random inputs and should be distinguished from security for adversarially chosen evaluation inputs.

For signatures, Katz and Vaikuntanathan [61] give bounded-leakage-resilient signatures, including one-time and few-time variants that tolerate leakage from the signer's entire state. Boyle, Segev and Wichs [17] obtain fully leakage-resilient signatures in the bounded-leakage setting, including leakage from signing randomness and intermediate values; their continual-leakage extension is discussed below. These bounded-leakage results do not provide an unbounded sequence of fresh leakage budgets through key refresh.

The basic key-leakage formulations described above restrict leakage to the stored key. This restriction should not be attributed to every bounded-leakage signature model: the full-state variants just mentioned also protect intermediate computation and signing randomness, subject to a bounded total leakage budget.

Variants of Only-Computation Leaks (OCL). An influential model of leakage introduced by [72], the OCL captures leakage during computation (including local randomness) and has a key-assumption – the existence of leak-free memory. The model allows for intermediate variables to be moved to the leak-free memory when not needed by the current computation. Alternatively, this could be thought of as a weaker notion of CML, where the adversary can only get leakage from the

sub-component currently being computed on.

The OCL model has been very successful, with a variety of primitives instantiated in this model: PRGs [72], stream ciphers [34, 77], PRFs [29], MACs [71], signatures [72, 35, 32], and PKE [64, 32]. The leakage restrictions and security interfaces differ across these results.

For PRFs, Dodis and Pietrzak [29] give a construction with non-adaptively selected leakage functions and independently leaking computation steps. Their leakage and test queries use disjoint input sets. Thus, "non-adaptive" here qualifies the leakage functions; it should not be read as a blanket restriction on all PRF evaluation queries. Barwell, Martin, Oswald and Stam [10] also use a sufficiently leakage-resilient PRF to obtain misuse-resistant, leakage-resilient authenticated encryption, and give a concrete pairing-based instantiation. Their security interfaces account for both protocol and side-channel leakage.

For MACs, an early OCL construction of Schipper combines a leakage-resilient generator with an ordinary one-time MAC [88]. Martin, Oswald, Stam and Wójcik [71] give a practical MAC with continual leakage under the only-computation-leaks assumption. Pereira, Standaert and Vivek [76] give specialized MAC and encryption constructions from symmetric primitives using a limited leak-free component. This protected-component assumption is separate from the OCL assumption on inactive memory, and these constructions should not be interpreted as supporting arbitrary leakage from all stored state. The authenticated-encryption composition results of [10] provide further related work on authentication with continuous leakage.

For signatures, Micali and Reyzin [72] provide the foundational physically observable framework. Faust, Kiltz, Pietrzak and Rothblum [35] construct stateful tree-based signatures from bounded-leakage-resilient three-time signatures, with a leakage budget for each signing operation on the accessed state. Dziembowski and Faust [32] give specialized inner-product-based protection for identification and signing with continually refreshed encodings and independently leaking parts. Their signing application is distinct from the later generic circuit compiler [33]; the security assumptions of the identification-to-signature transformation must also be taken into account.

The works of [34, 59, 33] also note that the OCL model can be naturally interpreted as a distributed protocol between independently leaking parties. This is made more formal and explicit in follow-up works; for example, [2] give a PKE scheme (with no post-challenge leakage) in this distributed leakage model. General-purpose protection is provided by the compilers of [59, 43, 33, 44], under their respective computational and leak-free-component assumptions.

Unfortunately, the limitations of the OCL model is demonstrated by its in-ability to capture memory-attacks such as cold-boot attacks [49]. [34] notes that the "only-computation leaks" assumption can be removed, independent leakages assumption is sufficient to get security. Variants of OCL with this relaxation were defined in strong-OCL, OCL+ [14] or Leaky-distributed system (LDS) [13, 16]. Indeed, this was formalized as the CML model in [18, 26] and extended by [70]. Thus, the CML model completely subsumes the OCL model. It was observed by [14, 22] that the compilers of [33] and [44] are also secure in the CML (or strong-OCL) model.

For signatures in continual memory-leakage models, the one-way relations of Dodis, Haralambiev, López-Alt and Wichs [26] and the storage and relation constructions of Dodis, Lewko, Waters and Wichs [27] are relevant building blocks. Boyle, Segev and Wichs [17] construct fully leakage-resilient signatures whose continual-leakage guarantees inherit the permitted leakage between updates and during refresh from the underlying relation. Malkin, Teranishi, Vahlis and Yung [70] give signatures with separate budgets for leakage between updates and during key generation and updates. These whole-state continual-leakage guarantees should be distinguished from the OCL restriction to accessed state and from our two-component, per-activation leakage model.

Noisy Leakage Model. First proposed by [20] and extended by [37]; their model allows each wire

value of a circuit computation to be provided to the adversary, but each wire value is flipped with some independent probability p. This captures the inherent noise present in the measurements used in some wire-probing based real-world side-channel attacks. [37] give a generic compiler, however it is prohibitively inefficient. Subsequent works consider more general leakages in the OCL model with bounded leakage. However, as noted by [93, 92], arbitrary polynomial (bounded) noisy leakages does not quite capture real-world attacks, while also adding unnecessary complexity and inefficiency. The work of [79, 30] give generic compilers in this noisy OCL model. In particular, [30] show an equivalence between noisy and wire-probing leakages, thus showing that the original compiler of [72] is also secure in the noisy leakage model. Subsequent works improve the efficiency, quantitive bounds, leakage-rates etc. However, this model suffers from the same limitations as the wire-probing model, in that it captures a very limited class of leakage functions.

Bounded Communication Leakage (BCL). The work of [37] extends the wire-probing model of [56] and gives a clean formalism of information-theoretic Leakage-Resilient Circuits (LRC) against global leakages. This model uses a leak-free trusted input encoding phase that takes a circuit, secret and outputs an encoded circuit (with encoded secret state). Similar to [56], the adversary is allowed to feed an arbitrary input and get as leakage a t-bit function (from the supported class) of the wire values of the encoded circuit evaluation. The stateful LRC version aims to capture a general computation device with updatable memory that is subject to continuous global leakage; here the encoded secret state is refreshed after each circuit evaluation. The work of [37] give stateful-LRC against AC^0 leakage and uses leak-free components (size in the security parameter) for refresh. Subsequent work of [85, 15] get rid of the leak-free components.

The work of [46, 40, 57] builds on the information-theoretic LRC model by borrowing ideas from OCL. In particular, the circuit wires are now divided into two partitions. A leakage function is allowed if it can be computed as an interactive two-party protocol between the two partitions using at most t-bits of bounded communication. Fundamentally differing from OCL and CML, in this model a partition of wires need not correspond to a valid sub-computation or sub-circuit. The information-theoretic LRC compiler of [57] gives continuous leakage-resilience against t-BCL leakage for any $t \in \mathbb{N}$.

Note that, the encoded circuit evaluation itself can also be modeled as a two-party protocol between parties P_0, P_1 (with each party getting as input a share of the encoded secret state). [57] defines it explicitly as t-BCL protocol – here, leakage is modeled by a two-party protocol between semi-honest adversaries $\mathbf{A}_0$, $\mathbf{A}_1$ s.t. each $\mathbf{A}_i$ sees the entire computation of party P_i. The t-BCL protocol model also has the following extra constraint – the split-state adversaries $\mathbf{A}_i$ have bounded memory of t-bits. In this model, [57] build leakage-resilient protocols from the DDH or Factoring assumptions. Note that, this model is completely subsumed by CML (which allows the split-state adversaries to have unbounded polynomial memory).

B Additional Preliminaries

B.1 Clocked cellular simulation

We describe how to simulate a polynomial-time Turing machine using a cellular machine with a fixed public schedule, and how to implement this simulation using the arithmetic operations supported by our compiler.

Definition 6 (Clocked cellular machine)**.** A clocked cellular machine is specified by a finite alphabet Σ containing a distinguished blank symbol, a local transition function $\phi : \Sigma^3 \to \Sigma$, and public bounds $S, T \in \mathbb{N}$.

A configuration is a vector $C \in \Sigma^S$. Starting from an initial configuration C_0, the machine evolves for exactly T steps according to

$$C_{t+1}[i] = \phi(C_t[i-1], C_t[i], C_t[i+1]), \qquad t = 0, \ldots, T-1,$$

where cells outside $\{1, \ldots, S\}$ are assigned the blank symbol. The machine's output is obtained by applying a fixed decoding function to C_T.

Simulating Turing machines. A polynomial-time multitape Turing machine is first converted, with polynomial overhead, into a single-tape Turing machine. A single-tape machine is then simulated step by step by a clocked cellular machine with polynomially bounded S, T and, for a fixed Turing machine, a constant-size alphabet and local transition rule. Each cellular symbol records the tape symbol at its position and, at the head position, a head marker with the control state. Since the head moves by at most one cell per step, the local rule updates these symbols using only the neighbourhood of radius one, as in the definition above.

Halting configurations are made stationary by extending the local transition rule appropriately. Thus, the cellular machine runs for exactly T steps regardless of the actual halting time, and its update schedule depends only on public parameters.

B.2 Continuous leakage-resilient public-key encryption

A stateful public-key encryption scheme $P = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ maintains a secret state $\mathsf{st} = (L, R)$ encoding a fixed logical secret key. Decryption returns a message or $\perp$ and an updated secret state. Key generation and initial encoding are leak-free; leakage begins only after initialization and cleanup.

Every protected computation uses the interaction of definition 2. In each protected round, the adversary may adaptively query either private component, with a single total λ-bit budget across all queries in that round. The resulting transcript contains all public communication and leakage answers. The activation views, retained workspace, state updates, and erasures are exactly those specified in definition 2.

Definition 7 (λ-IND-CCLA2 PKE)**.** A stateful PKE scheme is λ-IND-CCLA2 secure if it satisfies the correctness and security requirements below.

The requirements are:

1. **Correctness.** For every polynomial number of sequential honest encryptions and protected decryptions, all returned plaintexts are correct except with negligible probability.
2. **IND-CCLA2 security.** For every PPT adversary $\mathcal{A}$ making polynomially many adaptive oracle queries,
$$\mathrm{Adv}^{\mathsf{IND\text{-}CCLA2}}_{P,\lambda}(\mathcal{A}) := \left| \Pr[\mathsf{IND\text{-}CCLA2}_{P,\mathcal{A},\lambda}(1^\kappa) = 1] - \frac{1}{2} \right| \le \mathsf{negl}(\kappa).$$

The security experiment is defined as follows.

Experiment $\mathsf{IND\text{-}CCLA2}_{P,\mathcal{A},\lambda}(1^\kappa)$

1. Sample $(\mathsf{pk}, \mathsf{st}) \leftarrow \mathsf{Gen}(1^\kappa)$. Set $c^* \leftarrow \perp$.

2. Give $\mathcal{A}$ access to the following mutable oracle $\mathcal{D}_\lambda(c)$:

(a) If a challenge has been issued and the canonical string of c is exactly c^*, return $\perp$ before any protected computation, leakage, or state update.

(b) Otherwise, run the complete protected computation $\mathsf{Dec}(\mathsf{st}, c)$, interacting with $\mathcal{A}$ according to definition 2. Let (y, st', τ) denote its output, updated state, and complete public and leakage transcript. Malformed or rejected inputs return $y = \perp$ and execute the protected identity-refresh path.

(c) Atomically set $\mathsf{st} \leftarrow \mathsf{st}'$ before serving another query. Return (y, τ).

3. Run $(\sigma, m_0, m_1) \leftarrow \mathcal{A}_0^{\mathcal{D}_\lambda}(\mathsf{pk})$, where m_0, m_1 are valid messages of equal length.

4. Sample $b \leftarrow \{0,1\}$ and $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.

5. Run $b' \leftarrow \mathcal{A}_1^{\mathcal{D}_\lambda}(\sigma, c^*)$, using the same mutable state and oracle.

6. Output 1 if and only if $b' = b$.

Public keys and ciphertexts use canonical, injective encodings. If each private component stores a persistent share of $s(\kappa)$ bits between queries, the leakage rate is $\lambda(\kappa)/s(\kappa)$.

B.3 Continuous leakage resilient PRF and MAC

PRF. Let $F : \{0,1\}^{\ell_k} \times \{0,1\}^{\ell_x} \longrightarrow \{0,1\}^{\ell_y}$ be a deterministic polynomial-time function family. A two-component protected implementation consists of an initialization algorithm Init_F and a stateful evaluator Eval_F.

Definition 8 (Simulation-based CML PRF)**.** A scheme $(\mathsf{Init}_F, \mathsf{Eval}_F)$ is a simulation-based λ-CML PRF if it is correct and there exists a stateful efficient online simulator Sim s.t. for every PPT interactive adversary $\mathcal{A}$:

1. **Actual-output simulation.** Replacing the protected executions in the real experiment by Sim supplied with the actual pairs $(x_i, F_k(x_i))$ changes the adversary's complete view by at most negligible statistical distance.
2. **Pseudorandomness.**

$$\left|\Pr\left[\mathsf{Real}^{\mathsf{CML\text{-}PRF}}_{F,\mathcal{A},\lambda}(1^\kappa) = 1\right] - \Pr\left[\mathsf{Ideal}^{\mathsf{CML\text{-}PRF}}_{R,\mathsf{Sim},\mathcal{A},\lambda}(1^\kappa) = 1\right]\right| \leq \mathsf{negl}(\kappa).$$

For an interactive adversary $\mathcal{A}$ and simulator Sim, define the following experiments.

Real experiment. $\mathsf{Real}^{\mathsf{CML\text{-}PRF}}_{F,\mathcal{A},\lambda}(1^\kappa)$. Sample $k \leftarrow \{0,1\}^{\ell_k}$ and initialize $\mathsf{st} \leftarrow \mathsf{Init}_F(k)$ without leakage. Give $\mathcal{A}$ the public parameters. On each adaptive input x, run $\mathsf{Eval}_F(\mathsf{st}, x)$ with the CML interaction. Let (y, st', τ) be the resulting output, updated state, and transcript. Set $\mathsf{st} \leftarrow \mathsf{st}'$ and return (y, τ), with leakage answers delivered online. Output the final bit of $\mathcal{A}$.

Ideal experiment $\mathsf{Ideal}^{\mathsf{CML\text{-}PRF}}_{R,\mathsf{Sim},\mathcal{A},\lambda}(1^\kappa)$**.** Give $\mathcal{A}$ identically distributed public parameters. Initialize one stateful online simulator Sim with no logical secret key. Maintain a lazy-sampled random function

$$R : \{0,1\}^{\ell_x} \longrightarrow \{0,1\}^{\ell_y}.$$

On each adaptive input x, obtain $y = R(x)$, consistently on repeated inputs. Supply (x, y) to Sim. The simulator generates the public communication and answers the same adaptive leakage requests online, using the prescribed interface and per-round budget. Return y and the simulated transcript, and retain the simulator's private continuation state. Output the final bit of $\mathcal{A}$.

The same simulator is used for actual-output simulation and in the ideal experiment. It must run in polynomial time on every well-formed input/output pair, including pairs containing random-function outputs. Its statistical simulation guarantee is required only for actual outputs. Its private continuation state is not revealed to $\mathcal{A}$.

MAC. Let $\mathsf{MAC}_0 = (\mathsf{Kg}_0, \mathsf{Tag}_0, \mathsf{Vfy}_0)$ be a MAC with deterministic tagging and verification, computed by total uniform polynomial-time machines with public resource bounds. Messages and candidate tags have publicly bounded lengths; malformed messages or tags are rejected canonically. A two-component protected implementation consists of $\mathsf{Init}_{\mathsf{MAC}}$ and the stateful algorithms $\mathsf{Tag}_{\mathsf{CML}}$ and $\mathsf{Vfy}_{\mathsf{CML}}$. Both algorithms operate on the same mutable two-component state, which represents one fixed logical key.

Definition 9 (CML-secure MAC with tagging and verification leakage)**.** The protected implementation of MAC_0 is a λ-CML-secure MAC with tagging and verification leakage if it satisfies the following properties.

1. **Correctness.** Over any polynomial number of adaptive, interleaved well-formed calls, every tagging output equals $\mathsf{Tag}_0(k, m)$ and every verification output equals $\mathsf{Vfy}_0(k, m, t)$, except with negligible total probability.
2. **EUF-CML security.** For every PPT adversary $\mathcal{A}$ making polynomially many adaptive oracle queries,

$$\mathrm{Adv}^{\mathsf{EUF\text{-}CML}}_{\mathsf{MAC},\lambda}(\mathcal{A}) := \Pr\left[\mathsf{EUF\text{-}CML}_{\mathsf{MAC},\mathcal{A},\lambda}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa).$$

The security experiment is defined as follows.

Experiment $\mathsf{EUF\text{-}CML}_{\mathsf{MAC},\mathcal{A},\lambda}(1^\kappa)$:

1. Sample $k \leftarrow \mathsf{Kg}_0(1^\kappa)$ and initialize $\mathsf{st} \leftarrow \mathsf{Init}_{\mathsf{MAC}}(k)$ without leakage. Set $\mathcal{Q} \leftarrow \emptyset$ and $\mathsf{win} \leftarrow 0$.

2. Give $\mathcal{A}$ the public parameters and adaptive access to the following two oracles, in any interleaving. Oracle executions are sequential, and each execution uses the state left by the preceding one.

(a) **Tagging oracle $\mathcal{T}_\lambda(m)$.** For a well-formed message m, record m in $\mathcal{Q}$ before executing the protected operation, including requests whose execution subsequently fails. Run $\mathsf{Tag}_{\mathsf{CML}}(\mathsf{st}, m)$ with the CML interaction of definition 2, obtaining (t, st', τ). Set $\mathsf{st} \leftarrow \mathsf{st}'$ and return (t, τ), delivering leakage answers online.

(b) **Verification oracle $\mathcal{V}_\lambda(m, t)$.** For a well-formed pair (m, t), run $\mathsf{Vfy}_{\mathsf{CML}}(\mathsf{st}, m, t)$ with the same CML interaction, obtaining (b, st', τ), where $b \in \{0, 1, \perp\}$ and $\perp$ denotes exceptional failure. Set $\mathsf{st} \leftarrow \mathsf{st}'$ and return (b, τ), delivering leakage answers online. If $b = 1$ and $m \notin \mathcal{Q}$ at this query, set $\mathsf{win} \leftarrow 1$. Verification queries do not add messages to $\mathcal{Q}$.

Malformed requests return $\perp$ before any protected execution or leakage and leave the state unchanged. All well-formed verification requests, including rejected candidates, receive the prescribed leakage interaction.

3. Obtain a final purported forgery (m^*, t^*) from $\mathcal{A}$. Output 1 if and only if $\mathsf{win} = 1$ or

$$m^* \notin \mathcal{Q} \quad \text{and} \quad \mathsf{Vfy}_0(k, m^*, t^*) = 1,$$

where the latter condition requires a well-formed pair.

Every protected tagging or verification call uses the leakage interaction of definition 2. Initialization and the challenger's final verification check are leak-free. An accepting verification query on a message not previously submitted for tagging also counts as a forgery.

B.4 Continuous leakage-resilient digital signatures

Definition 10 (CML-secure digital signatures)**.** Let $\Sigma_0 = (\mathsf{Kg}_0, \mathsf{Sign}_0, \mathsf{Vfy}_0)$ be a signature scheme with deterministic signing. Let Init_Σ and $\mathsf{Sign}_{\mathsf{CML}}$ be its two-component initialization and stateful protected signing algorithms. The logical signing key includes any auxiliary secret used to derandomize a randomized base signer.

The implementation is a λ-CML-secure signature scheme if it is correct and, for every PPT adversary $\mathcal{A}$,

$$\mathrm{Adv}^{\mathsf{EUF\text{-}CML}}_{\Sigma,\lambda}(\mathcal{A}) := \Pr\left[\mathsf{EUF\text{-}CML}_{\Sigma,\mathcal{A},\lambda}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa).$$

The security experiment is defined as follows.

Experiment $\mathsf{EUF\text{-}CML}_{\Sigma,\mathcal{A},\lambda}(1^\kappa)$:

1. Sample $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}_0(1^\kappa)$ and initialize $\mathsf{st} \leftarrow \mathsf{Init}_\Sigma(\mathsf{sk})$ without leakage. Set $\mathcal{Q} \leftarrow \emptyset$.

2. Give $\mathcal{A}$ the verification key vk, the public parameters, and access to the mutable signing oracle $\mathcal{S}_\lambda(m)$:

1. Record the request in $\mathcal{Q}$ before executing the protected operation, including requests whose execution subsequently fails.
2. Run $\mathsf{Sign}_{\mathsf{CML}}(\mathsf{st}, m)$ with the CML interaction, obtaining $(\sigma, \mathsf{st}', \tau)$.
3. Set $\mathsf{st} \leftarrow \mathsf{st}'$ and return (σ, τ), delivering leakage answers online.

3. Obtain a purported forgery (m^*, σ^*) from $\mathcal{A}$.

4. Output 1 if and only if

$$m^* \notin \mathcal{Q} \quad \text{and} \quad \mathsf{Vfy}_0(\mathsf{vk}, m^*, \sigma^*) = 1.$$

The verification key stays fixed, and verification is public and unchanged. This definition requires a forgery on an unqueried message; it does not require strong unforgeability on previously queried messages.

C Concrete Protocols and Reconstruction of the DF Gadgets

This appendix proves theorem 2. We first formalize the reconstruction properties needed of the concrete DF gadgets. lemma 4 supplies conditional reconstruction of local random tapes; appendices C.2 to C.6 verify the properties for each gadget. The final subsection uses these properties to reconstruct complete executions and prove output simulation via the LRS hybrid argument.

Throughout, gadgets use the fixed one-attempt serialization of section 4. Our reconstruction argument builds on the techniques of [32, 23].

Definition 11 (Reconstructible DF gadget)**.** Consider one invocation of any of the concrete DF gadgets (refresh, multiplication, copy, translation, or public output), with its incident regular encodings and prescribed public data. The gadget is *reconstructible* if it has the following properties.

1. *Correctness and output distribution.* On successful prescribed checks, the gadget implements its stated operation. Each newly created closed encoding has the specified uniform regular distribution, jointly with any other outputs as specified by the gadget. The probability of a failed check is bounded as stated for that gadget, apart from negligible sampler failure.
2. *Conditional reconstruction.* For incident encodings compatible with the gadget's prescribed operation, an efficient kernel samples the joint conditional distribution of the gadget's algebraic execution, given those encodings, its prescribed public inputs, and successful checks. Together with lemma 4, this reconstructs the complete component activation views, including local random tapes and deterministic intermediate values, up to negligible fixed-clock error.
3. *One-sided programmed reconstruction.* For any choice of one incident closed encoding $E = (L, R)$, the kernel admits a programmed version for arbitrary regular incident endpoints, including endpoints that need not satisfy the gadget's prescribed input/output relation. Its left and right reconstructed component traces can be generated using, respectively, only L and only R of the designated encoding, the corresponding same-side shares of the other incident encodings, and the specified public reconstruction information and randomness. The resulting traces and public data have a jointly specified sampling procedure; in particular, the two component traces are not asserted to be independently sampled. The programmed kernel agrees with the exact conditional reconstruction in item 2 whenever the endpoints are compatible.

   Any public reconstruction information depending on the designated encoding must be explicitly identified. For these gadgets, it is either absent or determined by the first left coordinate of an incident encoding; all other public reconstruction randomness is independent of the designated encoding. This condition concerns the reconstruction interface, not disclosure of an entire encoding.

C.1 Reconstructing local randomness

The activation views of definition 2 contain not only the sampled algebraic values, but also local random coins and deterministic intermediate values. We first give a common sampling convention that permits reconstruction of the random coins conditioned on the sampled value. We then verify that it applies to all the algebraic sets used by the gadgets.

Lemma 4 (Conditional randomness reconstruction)**.** *Let $T_{\mathsf{gad}} \geq 1$ be a public upper bound on the number of sampling operations performed during one execution of a gadget.*

Consider a nonempty finite set S of size m, for which there is an efficiently computable bijection

$$\mathrm{rank} : S \longrightarrow \{0, \ldots, m-1\}$$

with an efficiently computable inverse unrank.

There is an efficient fixed-block sampler for S with the following properties:

1. *Conditioned on success, its output is uniform over S.*
2. *The probability that any sampling operation fails during one gadget execution is at most $2^{-2\kappa-1}$.*
3. *Given any successful output $s \in S$, the sampler's random block conditioned on producing s can be reconstructed exactly in expected polynomial time.*
4. *The conditional reconstruction admits a fixed-clock implementation such that the statistical error accumulated over all sampling operations in one gadget execution is at most $2^{-2\kappa}$.*

Specifically, set

$$\ell = \lceil \log_2 m \rceil + 2\kappa + \lceil \log_2 T_{\mathsf{gad}} \rceil + 1, \qquad a = \left\lfloor \frac{2^\ell}{m} \right\rfloor m.$$

The forward sampler reads an entire block $r \leftarrow \{0,1\}^\ell$, interpreted as an integer. If $r \geq a$, it takes the prescribed sampler-failure branch; otherwise, it outputs unrank$(r \bmod m)$.

To reconstruct the random block conditioned on a successful output $s \in S$, sample

$$j \leftarrow \{0, \ldots, a/m - 1\}, \qquad r = jm + \mathrm{rank}(s).$$

Applying this reconstruction to each local sampling operation, followed by deterministic replay of the activation code, reconstructs the complete conditional activation view. Over a polynomial execution, the fixed-clock reconstruction incurs only negligible statistical error.

Proof: Each element $s \in S$ has exactly a/m accepted random blocks, namely

$$\big\{jm + \mathrm{rank}(s) : 0 \leq j < a/m\big\}.$$

Consequently, the forward sampler is uniform over S conditioned on success, and the reconstruction procedure samples exactly the conditional distribution of its random block given s.

The probability that one forward sampling operation fails is

$$\frac{2^\ell - a}{2^\ell} < \frac{m}{2^\ell} \leq \frac{2^{-2\kappa-1}}{T_{\mathsf{gad}}}.$$

Since a gadget performs at most T_{gad} sampling operations, a union bound gives the claimed failure probability.

For fixed-clock conditional reconstruction, write

$$k = a/m, \qquad b = \lceil \log_2 k \rceil.$$

To sample j uniformly from $\{0, \ldots, k-1\}$, repeatedly sample a uniform b-bit integer and accept it if it is smaller than k. Each trial succeeds with probability at least $1/2$. Thus, the expected number of trials is at most two.

Cap the number of trials at

$$L = 2\kappa + \lceil \log_2 T_{\mathsf{gad}} \rceil + 1.$$

The probability that all L trials fail is at most

$$2^{-L} \le \frac{2^{-2\kappa-1}}{T_{\mathsf{gad}}}.$$

On this event, the conditional sampler takes a prescribed failure branch. Hence, its statistical distance from the exact conditional distribution is at most 2^{-L}. A union bound over all sampling operations in one gadget gives statistical error at most $2^{-2\kappa-1}$, and therefore at most $2^{-2\kappa}$.

Finally, once the input state, incoming messages, and local random blocks are fixed, the activation code determines all intermediate values and outgoing messages. Thus, reconstructing the random blocks and replaying the deterministic computation yields the complete conditional activation view. Over polynomially many gadget executions, the total statistical error remains negligible. □

Efficient sampling domains. The sets used below admit efficient public ranking and unranking: field vectors, nonzero vectors, affine hyperplanes, and the sets

$$\{M \in \mathrm{GL}_d(\mathbb{K}) : Mu = v\}, \qquad u, v \neq 0.$$

For the last set, choose canonical invertible matrices P_u, P_v such that $P_u e_1 = u$ and $P_v e_1 = v$. Every matrix in the set is uniquely expressible as

$$M = P_v \begin{pmatrix} 1 & b^T \\ 0 & B \end{pmatrix} P_u^{-1}, \qquad b \in \mathbb{K}^{d-1}, \quad B \in \mathrm{GL}_{d-1}(\mathbb{K}).$$

This parameterization recursively provides ranking and unranking for these matrix sets and for general linear groups. Gaussian elimination constructs the canonical basis extensions. All computations are padded to their public clocks.

We use the replay procedure of lemma 4 for every local random choice below. The correlation functionality contributes received local values, but no party-local randomness tape for generating them.

C.2 Exact refresh protocol and reconstruction

Refresh protocol. The left and right components initially hold L and R, respectively, where $(L, R) \in \mathcal{E}_d^\dagger(z)$. The protocol makes one padded attempt as follows.

Refresh:

- **Setup.** Invoke $\mathcal{O}'_{\mathbb{K},d}$, which writes $(A, \widetilde{A})$ privately to the left component and $(B, \widetilde{B})$ privately to the right component.
- **Round 1 – Left.** Sample M uniformly from $\{M \in \mathrm{GL}_d(\mathbb{K}) : ML = A\}$ and send M through common memory. This set is nonempty because L and A are nonzero.
- **Round 2 – Right.** Compute the candidate $R^+ = R + M^T B$ and announce whether $R^+_{2..d} \neq 0$. If this check passes, sample $\widetilde{M}$ uniformly from $\{\widetilde{M} \in \mathrm{GL}_d(\mathbb{K}) : \widetilde{M}R^+ = \widetilde{B}\}$ and send $\widetilde{M}$ through common memory. This set is nonempty because the check implies $R^+ \neq 0$ and the correlation guarantees $\widetilde{B} \neq 0$.
- **Round 3 – Left.** If the right check passed, compute $L^+ = L + \widetilde{M}^T \widetilde{A}$ and announce whether $L^+_1 \neq 0$.
- **Commit / rollback.** If both checks pass, both components commit (L^+, R^+) before either new share is used by a later instruction. Otherwise, they retain the last committed encoding (L, R). In either case, delete all unneeded candidates, local coins, consumed private message copies, and correlation outputs; on commitment, also delete the replaced shares.

Table 11: Refresh protocol

Lemma 5 (Refresh reconstructibility)**.** *The refresh gadget of table 11 satisfies definition 11. Its successful output is a fresh uniform regular encoding of the input value, and its success probability is* $(1 - Q^{-(d-1)})(1 - Q^{-1})$*, apart from negligible sampler failures. It requires no encoding-dependent public reconstruction information.*

Proof:

- **Reconstruction kernel (used in items (ii) and (iii)).** Fix regular encodings

$$E = (L, R) \in \mathcal{E}^\dagger_d(z), \qquad E^+ = (L^+, R^+) \in \mathcal{E}^\dagger_d(z^+).$$

Sample $M, \widetilde{M}$ independently and uniformly from $\mathrm{GL}_d(\mathbb{K})$, and set

$$\begin{aligned} A &= ML, & B &= M^{-T}(R^+ - R), \\ \widetilde{B} &= \widetilde{M}R^+, & \widetilde{A} &= \widetilde{M}^{-T}(L^+ - L). \end{aligned} \tag{18}$$

These equations replay the share updates:

$$\begin{aligned} R + M^T B &= R^+, \\ L + \widetilde{M}^T \widetilde{A} &= L^+. \end{aligned}$$

Moreover,

$$\langle A, B\rangle + \langle \widetilde{A}, \widetilde{B}\rangle = \langle L^+, R^+\rangle - \langle L, R\rangle. \tag{19}$$

Indeed, the two terms equal $\langle L, R^+ - R\rangle$ and $\langle L^+ - L, R^+\rangle$. The reconstructed correlation therefore has offset $\delta = z^+ - z$. Its required nonzero coordinates follow from $L \neq 0$ and $R^+ \neq 0$.

- **(ii) Exact conditional reconstruction.** The transitive action of $\mathrm{GL}_d(\mathbb{K})$ on nonzero vectors gives the same number of matrices mapping u to v for every nonzero u, v:

$$h_d = |\{M \in \mathrm{GL}_d(\mathbb{K}) : Mu = v\}| = \frac{|\mathrm{GL}_d(\mathbb{K})|}{Q^d - 1}, \qquad u, v \neq 0. \tag{20}$$

For fixed nonzero $A, \widetilde{B}$, the offset equation has Q^{2d-1} solutions $(B, \widetilde{A})$. Consequently,

$$|\operatorname{supp}(\mathcal{O}'_{\delta,\mathbb{K},d})| = (Q^d - 1)^2 Q^{2d-1}, \tag{21}$$

independently of δ.

For fixed encodings and public matrices, (18) determines a unique correlation tuple and protocol path. Conversely, every successful path with these encodings determines that tuple from its matrices. By eqs. (20) and (21), all such paths have equal probability. Uniform independent matrices therefore give the exact conditional algebraic path law. For compatible encodings, $\delta = 0$, this is the honest conditional law.

- **(iii) One-sided programmed reconstruction.** The left trace is determined by

$$L, L^+, M, \widetilde{M}, \quad A = ML, \quad \widetilde{A} = \widetilde{M}^{-T}(L^+ - L),$$

and the right trace by

$$R, R^+, M, \widetilde{M}, \quad B = M^{-T}(R^+ - R), \quad \widetilde{B} = \widetilde{M}R^+.$$

Each trace uses only same-side shares of the encodings. The matrices are encoding-independent public random data. Applying lemma 4 reconstructs the complete local randomness tapes and intermediate values.

- **(i) Correctness and output distribution.** In a zero-offset refresh, B is uniform in $\mathbb{K}^d$ after fixing A. The sampler for M uses only L, A, so $R^+ = R + M^T B$ is uniform in $\mathbb{K}^d$. Hence

$$\Pr[R^+_{2..d} = 0] = q_d = Q^{-(d-1)}.$$

Conditioned on $A, B, \widetilde{B}$, $\widetilde{A}$ is uniform subject to

$$\langle \widetilde{A}, \widetilde{B} \rangle = -\langle A, B \rangle.$$

For a fixed R^+ with nonzero tail, $\widetilde{B} = \widetilde{M}R^+$. The invertible change of variables $Y = \widetilde{M}^T \widetilde{A}$ therefore makes Y uniform subject to

$$\langle Y, R^+ \rangle = -\langle A, B \rangle.$$

Since $\langle L, R^+ \rangle = z + \langle A, B \rangle$, the candidate $L^+ = L + Y$ is uniform subject to $\langle L^+, R^+ \rangle = z$. The nonzero right tail makes L^+_1 uniform in $\mathbb{K}$. Consequently,

$$\Pr[\mathsf{Good}_{\mathsf{ref}}] = (1 - q_d)(1 - Q^{-1}),$$

giving (8).

Conditioned on success, every pair in $\mathcal{E}^\dagger_d(z)$ has equal probability. Thus the output is exactly $\mathsf{IPEnc}^\dagger_d(z)$, independently of the particular input encoding of z.

□

C.3 Exact multiplication reconstruction

Multiplication protocol. For the gate $G_c(s_0, s_1) = c - s_0 s_1$, where $c \in \mathbb{K}$ is public, the left component holds $L^{(0)}, L^{(1)}$ and the right component holds $R^{(0)}, R^{(1)}$, with $(L^{(i)}, R^{(i)}) \in \mathcal{E}^\dagger_d(s_i)$. Put $D = d^2$. The protocol makes one padded attempt as follows.

Multiplication:

- **Form the tensor encoding.** Each component locally computes its tensor share: $L^{(\otimes)} = L^{(0)} \otimes L^{(1)}, \qquad R^{(\otimes)} = R^{(0)} \otimes R^{(1)}$. The identity $\langle L^{(\otimes)}, R^{(\otimes)} \rangle = \langle L^{(0)}, R^{(0)} \rangle \langle L^{(1)}, R^{(1)} \rangle = s_0 s_1$ shows that these shares encode the product.

- **Refresh at dimension D.** Run the refresh protocol on $(L^{(\otimes)}, R^{(\otimes)})$, obtaining a candidate pair (L', R'). If the refresh fails, take the padded failure branch below. Otherwise, $L'_1 \neq 0$ and $R'_{2..D} \neq 0$.

- **Left activation: expose the compression data.** Set $x = L'_1, \qquad L^{(w)} = L'_{1..d}, \qquad U = L'_{d+1..D}$. We call U the back of L'. Reserve a fresh identifier e for the candidate output encoding; on successful commitment, this identifier is retained. Publish the tagged cache entry (e, x) and send U through common memory. The left component retains $L^{(w)}$ as its candidate output share.

- **Right activation: compress.** Using the received x, U, compute $t = \langle U, R'_{d+1..D} \rangle, \qquad R^{(w)} = -R'_{1..d} + e_1 x^{-1}(c - t)$, where $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{K}^d$. The inverse x^{-1} exists because the internal refresh succeeded. Announce whether $R_{w,2..d} \neq 0$.

- **Commit / rollback.** If the internal refresh and the compression check both succeed, both components commit $(L^{(w)}, R^{(w)})$ before either new share is used by a later instruction. Otherwise, delete all uncommitted gadget data and retain the last committed compiler state. At gadget completion, delete all temporary tensor shares, internal refresh workspace, remaining local coins, consumed private message copies, and correlation outputs. Consumed input encodings are deleted according to the public lifetime schedule.

Table 12: Multiplication protocol

Lemma 6 (Multiplication reconstructibility)**.** *The multiplication gadget of table 12 satisfies definition 11. Its failure probability is at most $3/Q$ for $d \geq 2$, apart from negligible sampler failures. Its only encoding-dependent public reconstruction information is the first left coordinate of its output encoding.*

Proof:

- **Reconstruction kernel (used in items (ii) and (iii)).** Fix stable input pairs and a stable output $(L^{(w)}, R^{(w)}) \in \mathcal{E}_d^{\dagger}(w)$. Write

$$P = L^{(0)} \otimes L^{(1)}, \qquad S = R^{(0)} \otimes R^{(1)}, \qquad x = L_1^{(w)}.$$

Sample $U, V \leftarrow \mathbb{K}^{D-d}$ independently, and define

$$\begin{aligned} L' &= L^{(w)} \parallel U, \\ R' &= \Big(-R^{(w)} + e_1 x^{-1}(c - \langle U, V \rangle) \Big) \parallel V. \end{aligned} \tag{22}$$

Compression returns exactly $(L^{(w)}, R^{(w)})$, and

$$\langle L', R' \rangle = c - \langle L^{(w)}, R^{(w)} \rangle. \tag{23}$$

Indeed,

$$\langle L', R' \rangle = \Big\langle L^{(w)}, -R^{(w)} + e_1 x^{-1}(c - \langle U, V \rangle) \Big\rangle + \langle U, V \rangle$$

$$= c - \langle L^{(w)}, R^{(w)} \rangle.$$

Apply (18) at dimension D from (P, S) to (L', R'). The offset is

$$c - \langle L^{(w)}, R^{(w)} \rangle - \langle L^{(0)}, R^{(0)} \rangle \langle L^{(1)}, R^{(1)} \rangle. \tag{24}$$

It vanishes for compatible encodings.

Both internal encodings are regular. For (P, S), the first left coordinate is nonzero, and the nonzero right tails give a nonzero tensor coordinate outside the first position. For (L', R'), the first left coordinate is $x \neq 0$, and right coordinates $2, \ldots, d$ equal $-R_{w,2..d}$.

- **(i) Correctness and output distribution; (ii) exact conditional reconstruction.** Put $p = \langle P, S \rangle$ and $w = c - p$. For each fixed outer pair $(L^{(w)}, R^{(w)}) \in \mathcal{E}_d^\dagger(w)$, (22) gives a bijection between its internal preimages and $\mathbb{K}^{D-d} \times \mathbb{K}^{D-d}$. Every regular outer pair therefore has $Q^{2(D-d)}$ preimages. The uniform internal refresh output consequently gives a uniform outer output conditioned on the compression check.

  Conditioned on the outer encodings, the backs U, V of L', R' are independent and uniform. Together with the exact conditional refresh law, this proves the exact conditional algebraic multiplication law.

  For a uniform nonzero internal right tail, the compression check fails with probability

$$\frac{Q^{D-d} - 1}{Q^{D-1} - 1},$$

  as in (9). Together with internal refresh failure, the probability is at most $3/Q$ for $d \geq 2$, apart from negligible sampler failures.

- **(iii) One-sided programmed reconstruction and public data.** The public back U and refresh matrices are uniform, encoding-independent public random data; V is right-local. The only encoding-dependent public value is $x = L_1^{(w)}$. We verify the one-sided property for each possible designated closed encoding. If the first input encoding is designated, the tensor shares $P = L^{(0)} \otimes L^{(1)}$ and $S = R^{(0)} \otimes R^{(1)}$ use its left and right shares, respectively, only on their corresponding sides; the second input case is symmetric. The outer output encoding and its coordinate x are then fixed non-target data. If instead the output encoding is designated, the inverse-compression formula reconstructs $L' = L^{(w)} \parallel U$ from its left share, and reconstructs $R' = (-R^{(w)} + e_1 x^{-1}(c - \langle U, V \rangle)) \| V$ from its right share and the single public coordinate $x = L_1^{(w)}$. In every case the subsequent inverse-refresh formulas use the tensor input and internal output only through their respective same-side shares, with the jointly sampled public reconstruction randomness. These formulas also define the programmed traces when the incident encoding values are incompatible; their LFH offset is exactly (24). No other public reconstruction datum depends on the designated encoding. Conditional randomness reconstruction followed by deterministic replay supplies the complete activation views.

  The tagged cache entry records the output encoding's first left coordinate. Any later translation of that encoding uses the same coordinate. Refresh, copy, and use as a multiplication input expose no additional encoding-dependent public value.

□

C.4 Exact copy protocol and reconstruction

Copy protocol. The left and right components initially hold L and R, respectively, where $(L, R) \in \mathcal{E}_d^\dagger(z)$. The input encoding is consumed only upon successful commitment of both output encodings.

Copy:
• **First attempt.** Run the refresh protocol on (L, R) using a fresh call to $\mathcal{O}'_{\mathbb{K},d}$ and fresh local randomness. Store its successful candidate as $(L^{(0)}, R^{(0)})$, without replacing or deleting (L, R).
• **Second attempt.** Run another refresh attempt on the same original pair (L, R), using an independent call to $\mathcal{O}'_{\mathbb{K},d}$ and independent local randomness. Store its successful candidate as $(L^{(1)}, R^{(1)})$. This attempt occupies its prescribed schedule even if the first attempt failed. A candidate produced by the first attempt remains in its component's private, leakable memory until commitment or rollback.
• **Commit / rollback.** If both attempts succeed, both components commit the two candidate pairs under distinct fresh encoding identifiers before either output is used. They then delete the original shares and all unneeded temporary data. If either attempt fails, both components discard both candidates and retain the original encoding (L, R). In either case, delete consumed correlation outputs, local coins, and private message copies according to the prescribed cleanup schedule.

Table 13: Copy protocol

Each constituent attempt uses the refresh protocol with its encoding-replacement step deferred to the joint commitment above. No attempt is retried.

Lemma 7 (Copy reconstructibility)**.** *The copy gadget satisfies definition 11. On success its two outputs are independent uniform regular encodings of the input value. Its failure probability is at most $4/Q$ for $d \geq 2$, apart from negligible sampler failures. No encoding-dependent public reconstruction information is needed.*

Proof:

- **(i) Correctness and output distribution.** Each successful refresh preserves the encoded value. Conditional on the original pair $E = (L, R)$, the attempts are independent. Each succeeds with probability
$$p_d = (1 - Q^{-(d-1)})(1 - Q^{-1})$$
and returns a uniform regular encoding of z. Hence
$$\mathcal{L}\big(E^{(0)}, E^{(1)} \mid E, \mathsf{Good}_{\mathsf{copy}}\big) = \mathsf{IPEnc}_d^{\dagger}(z) \otimes \mathsf{IPEnc}_d^{\dagger}(z).$$
The failure probability $1 - p_d^2$ is at most $4/Q$ for $d \geq 2$, apart from negligible sampler failures.
- **(ii) Exact conditional reconstruction and programmed kernel.** Fix the input and both output encodings. Apply (18) independently from E to each $E^{(j)}$. The offsets are
$$\langle L^{(j)}, R^{(j)} \rangle - \langle L, R \rangle, \qquad j \in \{0, 1\}.$$
For compatible encodings they vanish. The refresh conditional-path law and independence of the attempts give the exact algebraic copy law.
- **(iii) One-sided programmed reconstruction and public data.** Each left trace depends only on $L, L^{(0)}, L^{(1)}$ and the public matrices; each right trace depends only on $R, R^{(0)}, R^{(1)}$ and those matrices. Copy therefore exposes no encoding-dependent public field element. Apply lemma 4 to both attempts and replay their joint commitment and cleanup, including the first candidate retained during the second attempt.

□

C.5 Exact translation protocol and reconstruction

Translation protocol. Let $c \in \mathbb{K}$ be public. The left and right components initially hold L and R, respectively, where $E = (L, R) \in \mathcal{E}_d^\dagger(z)$.

Translation:
• **Publish the first coordinate.** The left component sends $x = L_1$ through common memory.
• **Translate the right share.** The right component computes, in fresh private memory, $\widehat{R} = R + e_1 x^{-1} c$. where e_1 is the first standard basis vector. It retains the original right share R for rollback. The left share remains L.
• **Refresh the translated pair.** Run one refresh attempt on $(L, \widehat{R})$, using a fresh call to $\mathcal{O}'_{\mathbb{K},d}$ and fresh local randomness. Denote the candidate output by (L^+, R^+). Defer the refresh protocol's encoding replacement and cleanup to the joint commitment below. The original shares remain in their respective private, leakable memories throughout the attempt.
• **Commit / rollback.** If the refresh succeeds, both components commit (L^+, R^+) before either new share is used by a later instruction. They then delete the original shares and the temporary translated share. Otherwise, they discard the candidates and the temporary translated share, retaining (L, R). In either case, delete consumed correlation outputs, local coins, and private message copies according to the prescribed cleanup schedule. The published value x remains part of the public transcript even on failure.

Table 14: Translation protocol

All failure decisions follow the fixed padded schedule. There are no retries.

Lemma 8 (Translation reconstructibility)**.** *The translation gadget of table 14 satisfies definition 11. Its successful output is a fresh uniform regular encoding of the input value plus c; its failure probability is* $Q^{-(d-1)} + (1 - Q^{-(d-1)})Q^{-1}$*, apart from negligible sampler failures. Its only encoding-dependent public reconstruction information is the first left coordinate of its input encoding.*

Proof:

- **(i) Correctness and output distribution.** Since $x = L_1 \neq 0$,

$$\begin{aligned} \langle L, \widehat{R} \rangle &= \langle L, R \rangle + \langle L, e_1 \rangle x^{-1} c \\ &= z + c. \end{aligned}$$

  Moreover, $\widehat{R}_{2..d} = R_{2..d} \neq 0$. Thus $(L, \widehat{R})$ is a regular encoding of $z + c$. The refresh output law gives a uniform regular encoding of $z + c$ on success, independently of the particular input encoding of z. Its failure probability, ignoring negligible sampler failures, is

$$q_d + (1 - q_d)Q^{-1}, \qquad q_d = Q^{-(d-1)}.$$

- **(ii) Exact conditional reconstruction and programmed kernel.** Fix regular encodings $E = (L, R)$ and $E^+ = (L^+, R^+)$ encoding z, z^+. Set $x = L_1$ and $\widehat{R} = R + e_1 x^{-1} c$. Sample $M, \widetilde{M}$ independently and uniformly from $\mathrm{GL}_d(\mathbb{K})$, and set

$$A = ML,$$

$$
\begin{aligned}
B &= M^{-T}(R^+ - \widehat{R}),\\
\widetilde{B} &= \widetilde{M} R^+,\\
\widetilde{A} &= \widetilde{M}^{-T}(L^+ - L).
\end{aligned}
$$

These are the inverse refresh equations from $(L, \widehat{R})$ to E^+. Their offset is

$$
\begin{aligned}
\langle A, B\rangle + \langle \widetilde{A}, \widetilde{B}\rangle &= \langle L^+, R^+\rangle - \langle L, \widehat{R}\rangle\\
&= z^+ - z - c.
\end{aligned}
$$

It vanishes for compatible encodings. The exact conditional refresh law therefore gives the exact conditional algebraic translation law.

- **(iii) One-sided programmed reconstruction and public data.** Given the public x, the left trace depends only on L, L^+ and the public matrices; the right trace depends only on R, R^+, c, x, and those matrices. Canonical replay reconstructs the complete views, including retained original shares and the temporary translated share.
  The sole encoding-dependent public datum is $x = L_1$. If this coordinate was already published when the encoding was produced, its transmission here is a deterministic repetition. Every publication of this coordinate during the same encoding's lifetime is determined by its first publication. Thus these transmissions require only one field element for the complete incident bundle.

□

C.6 Exact public output gadget and reconstruction

Public output protocol. The left and right components initially hold L and R, respectively, where $E = (L, R) \in \mathcal{E}_d^\dagger(z)$ is a closed encoding. The input encoding is consumed only upon successful commitment of a fresh pair for opening.

Public output:

- **Refresh the closed encoding.** Run one refresh attempt on (L, R) using the protocol of appendix C.2, with a fresh inputless call to $\mathcal{O}'_{\mathbb{K},d}$ and fresh local randomness. Denote the candidate output by $(L^{(\mathrm{out})}, R^{(\mathrm{out})})$. Defer the refresh protocol's encoding replacement and cleanup to the commitment below. The original shares remain in their respective private, leakable memories throughout the attempt.

- **Commit / rollback.** If both refresh checks pass, both components commit $E_{\mathrm{out}} = (L^{(\mathrm{out})}, R^{(\mathrm{out})})$ before either new share is used for opening. They then delete the original shares. Otherwise, they discard the candidates and retain (L, R). In either case, delete consumed correlation outputs, local coins, and private message copies according to the prescribed cleanup schedule.

- **Opening – Left.** On success, the left component sends $L^{(\mathrm{out})}$ through common memory. On failure, it executes the prescribed padded activation without sending a candidate share.

- **Opening – Right.** On success, the right component receives $L^{(\mathrm{out})}$, computes $y = \langle L^{(\mathrm{out})}, R^{(\mathrm{out})}\rangle$, and publishes y through common memory. Both components return this public value. On failure, execute the prescribed padded activation and take the public failure branch.

- **Cleanup.** On success, delete the transparent shares, received private copies, and opening workspace after their last use. On failure, retain the original closed encoding E. The public transcript remains in either case.

Table 15: Public output protocol

All failure decisions follow the fixed padded schedule. There are no retries, and no transparent pair is reused as a closed encoding. Failure branches are charged to the existing negligible bad-event bound.

Lemma 9 (Public output reconstructibility)**.** *The public-output gadget satisfies definition 11 with respect to its closed input encoding. On success it opens the correct value through a fresh uniform regular transparent pair. Its failure probability is at most* $2/Q$*, apart from negligible sampler failures. It requires no additional encoding-dependent public reconstruction information about the closed input.*

Proof:

- **(i) Correctness and output distribution.** A successful refresh preserves the encoded value, so
$$y = \langle L^{(\mathrm{out})}, R^{(\mathrm{out})} \rangle = \langle L, R \rangle = z.$$
Conditional on success, E_{out} is a uniform regular encoding of z, independently of the particular closed input encoding. The opening is deterministic given this fresh pair. Ignoring negligible sampler failures, the gadget's failure probability is exactly the refresh failure probability
$$q_d + (1 - q_d)Q^{-1}, \qquad q_d = Q^{-(d-1)},$$
which is at most $2/Q$.
- **(ii) Exact conditional reconstruction and programmed kernel.** Sample independent uniform $M, \widetilde{M} \in \mathrm{GL}_d(\mathbb{K})$ and set
$$A = ML, \quad B = M^{-T}(R^{(\mathrm{out})} - R), \quad \widetilde{B} = \widetilde{M}R^{(\mathrm{out})}, \quad \widetilde{A} = \widetilde{M}^{-T}(L^{(\mathrm{out})} - L).$$
By (19), the reconstructed LFH tuple has offset $y - \langle L, R \rangle$. For compatible endpoints, the exact conditional law established for (18) applies; the fresh committed pair has the stated uniform encoding law. Conditional randomness reconstruction (lemma 4) supplies local coins; deterministic replay supplies intermediate values. The opening then deterministically sends $L^{(\mathrm{out})}$ and publishes y. For incompatible endpoints these same equations define the hybrid trace; they do not represent a call to a new real LFH oracle.
- **(iii) One-sided programmed reconstruction and public data.** Fix the sampled transparent pair and the public matrices as reconstruction randomness. Their distribution is independent of both shares of E. The left refresh trace uses only $L, L^{(\mathrm{out})}, M, \widetilde{M}$; the right refresh trace uses only $R, R^{(\mathrm{out})}, M, \widetilde{M}$. Both opening traces are determined by the fixed transparent pair. Thus no opposite share of E is needed, and publishing the transparent left share adds no encoding-dependent public data about E. All retained inactive state follows the same deletion schedule. The sole refresh fails with probability at most $2/Q$, apart from negligible sampler failures, already covered by the refresh error accounting.

□

C.7 Proof of theorem 2

We first isolate the two global consequences of the local reconstructibility statements. Throughout this subsection an *encoding identifier* refers only to a closed, regular, d-dimensional encoding. Public-input encodings, the internal d^2-dimensional pair used in multiplication, and the transparent pair used for a public opening are not assigned identifiers. Every closed encoding has one producer (or is the initial encoding), and its identifier is retained until its last use, including across program boundaries. The public schedule determines the wiring, creation, and deletion of identifiers. In particular, encodings are the only persistent private state shared between invocations; all gadget-local randomness and LFH correlations are fresh.

Lemma 10 (Joint reconstruction)**.** *For any fixed finite, successful, publicly scheduled DF execution skeleton, fix the regular incident encodings and prescribed public data so that all gadget relations are compatible. The joint conditional law of the complete gadget activation views is sampled by the product of the local conditional reconstruction kernels of lemmas 5 to 9, followed by the conditional random-tape reconstruction of lemma 4, up to negligible fixed-clock error. For arbitrary regular endpoint assignments, the corresponding programmed kernels define a joint distribution of complete views; this distribution agrees with the preceding conditional law when the assignments are compatible. The sampling procedure admits a causal, lazy implementation which retains every already sampled trace and every released prefix.*

Proof: Fix first a predetermined skeleton. Write $\mathbf{E}$ for the incident encodings and $\mathbf{P}$ for the prescribed public data. The honest generative law is a product of gadget transition laws: all dependencies between distinct invocations pass through the incident encodings and fixed shared public data, since no other private state persists. The unique-producer property prevents two independent transitions from imposing competing output-generation laws on the same encoding. Conditioning on $\mathbf{E}, \mathbf{P}$ and on the success checks consequently leaves precisely the local conditional laws specified by the reconstruction lemmas. Thus, writing V_g for the algebraic trace of gadget g,

$$\mathsf{Law}((V_g)_g \mid \mathbf{E}, \mathbf{P}, \mathsf{Succ}) = \bigotimes_g \mathsf{Rec}_g^{\mathrm{cond}}(\,\cdot \mid \mathbf{E}_{\partial g}, \mathbf{P}_g).$$

Here public data which are outputs of a gadget are conditioned on consistently, rather than independently resampled. The local kernels reconstruct the joint left/right trace, not merely its marginals. Conditional random-tape replay from lemma 4 then reconstructs the complete activation views, including deterministic intermediate values.

For arbitrary regular endpoints, execute the same kernels with the offset prescriptions of the local lemmas. This is a definition of a programmed joint law; no claim is made that incompatible endpoints occur in an honest execution. All shared encodings and public values are fixed consistently, and each gadget's auxiliary randomness is sampled only once. For the adaptive extension, fix the public one-attempt schedule of each of the five concrete gadgets, including its padded checks, commit/rollback point, and deletion rules. We prove prefix consistency by induction over the publicly scheduled activation and disclosure events. Initially the retained encodings and released transcript are fixed. Suppose the reconstructed prefix agrees with the prescribed conditional (or programmed) law and all earlier traces are retained. The adversary's next choice of public input, leakage function, or stopping decision is determined by that prefix and its own coins. At a newly reached gadget, use its already fixed incident endpoints and public inputs to sample the appropriate local reconstruction kernel once; replay its local random tapes by lemma 4. Release only the messages and leakage answers prescribed at the next scheduled event, evaluating each leakage function on the retained trace rather than resampling it. At a commit, register the uniquely produced output identifier(s) before

any subsequent use; at a rollback, retain the prior committed encodings and delete uncommitted candidates. Public-output openings are handled by the prescribed transparent pair, not by creating a new closed identifier. Since the only persistent private state consists of committed encodings, the inductive state contains every private value that can influence a later gadget. The local conditional law establishes the next-step distribution for compatible endpoints, while the programmed kernel defines it for arbitrary endpoints. Hence the induction preserves the entire released prefix, including across program boundaries. This is an online sampling argument for the actual fixed gadget schedules, not an assertion that a retrospectively conditioned adaptive schedule has independent local traces.

□

Lemma 11 (One-sided incident-bundle reconstruction)**.** *In the programmed joint reconstruction, designate any closed encoding $e = (L, R)$, and sample every other closed encoding and all auxiliary reconstruction randomness independently of the choice of its shares. The entire collection $\mathcal{B}(e)$ of its producing, consuming, and retaining activation views, including their public messages, can be generated through separate access to L and R. The only additional encoding-dependent public reconstruction datum is the single field element L_1, if needed. All traces and public data are sampled once and reused. If $\mathcal{B}(e)$ meets at most H protected activations, every admissible physical leakage interaction with this bundle is therefore implementable by an adaptive two-share leakage adversary of total output length at most*

$$H\lambda + \lceil N \rceil \leq \Lambda_d. \tag{25}$$

Proof: Apply the one-sided programmed reconstruction property to each incident gadget, designating e in that local invocation. For a left activation, substitute the left reconstruction into the adversary's requested leakage function; this yields a function of L alone, with the already sampled non-target state, public prefix, and reconstruction randomness hardwired. Do the same with R for right activations. Retaining an encoding simply retains the same share in the component's state. The same sampled gadget trace is used for all queries involving that activation, preserving their joint law and the correlations between the two components.

Refresh and copy have only encoding-independent public reconstruction randomness. In multiplication the extra public coordinate is the output encoding's first left coordinate; in translation it is the input encoding's first left coordinate. The public-output gadget uses a separate transparent opening pair sampled from its prescribed public value and the sealing-refresh kernel for its closed predecessor. Consequently, whenever public reconstruction data depend on e, that dependence is through $x_e = L_1$ alone. Query x_e from the left-share leakage oracle once and cache it; repeated uses are post-processing, not new leakage. All other public reconstruction randomness is independent of the challenge shares. Offset correlations need not be honest zero-offset correlations when the endpoints are incompatible; their programmed joint law is precisely the one fixed in lemma 10.

At most H protected activations involve e, each releasing at most λ physical leakage bits. The coordinate x_e costs at most $\lceil N \rceil$ additional bits. This proves (25), including across invocation boundaries: the same identifier, share pair, cached coordinate, and previously sampled traces are retained throughout its lifetime. □

We now prove theorem 2. We first restate the theorem for convenience.

Theorem 2 (Output simulation for protected DF programs)**.** *Let $\mathbb{K}$ be an efficiently sampleable and computable finite field of size Q, with $N = \log_2 Q = \Omega(\kappa)$ and $N = \mathrm{poly}(\kappa)$. Let $d \geq 2$ be polynomially bounded in κ, and fix constants $a > 1$ and $3/2 < b < d/2$ as in lemma 1.*

Every protected DF program of definition 4 with exposure bound H is output-simulatable in the sense of definition 3 over polynomially many adaptive invocations, provided

$$H\lambda + \lceil N \rceil \leq \Lambda_d. \tag{10}$$

The statistical simulation error includes an additive $\Pr[\mathsf{BadExp}_H]$ *term, which is negligible by the exposure-bound condition. The stateful PPT online simulator receives only each public input and its actual reachable public output. It preserves every previously released public and leakage prefix and passes the exact retained simulation-state encoding to the next invocation without resampling it.*

Proof: Fix the logical initial state and an admissible PPT adversary. All statements below are pointwise in that logical state; the simulator itself does not receive it. Fix public polynomial bounds M on the number of closed encoding identifiers and u_e on the number of alternating LRS leakage queries for identifier e, including the optional coordinate query. Pad unused identifiers with dummy slots.

Bad Events. We first account for the bad events in the execution. Let $\mathsf{BadCheck}$ denote the event that some prescribed gadget check fails, and let BadExp_H denote the event that the exposure bound H is violated.

By the correctness guarantees of lemmas 5 to 9, each reached gadget fails with probability at most $O(1/Q)$, apart from negligible sampling errors. Since the execution contains at most polynomially many gadget invocations, a union bound gives

$$\Pr[\mathsf{BadCheck}] \leq \mathrm{poly}(\kappa)/Q + \mathsf{negl}(\kappa).$$

Moreover, $\Pr[\mathsf{BadExp}_H]$ is negligible by the hypothesis of the theorem.

Stopping rule. The hypothesis bounds BadExp_H only in the real interaction, so it cannot serve directly as a leakage budget inside the programmed hybrids. We therefore build the bound into every experiment below. For an experiment X, let $\mathsf{X}^{\mathsf{stop}}$ be the same experiment, except that, immediately before the first protected activation that would make the incident bundle of some closed encoding meet more than H protected activations, it switches to a dummy continuation: it performs no further protected activation, answers every further leakage query with the all-zero string of the prescribed length, and releases a fixed symbol stop in place of every further public message and public output, while $\mathcal{A}$ keeps running. Whether an activation is incident to a given identifier is determined by the public schedule, the public inputs and the publicly announced check outcomes, that is, by the public identifier-retention history. So the stopping decision is a function of the public prefix, and it is made before any leakage of that activation is released. Hence, in every stopped experiment, on every execution path, the incident bundle of every closed encoding meets at most H protected activations. Since $\mathcal{A}$'s coins are part of its view, the view in $\mathsf{X}^{\mathsf{stop}}$ is a fixed function of the view in X, so $\Delta(\mathsf{X}^{\mathsf{stop}}, \mathsf{Y}^{\mathsf{stop}}) \leq \Delta(\mathsf{X}, \mathsf{Y})$ for any two experiments X, Y.

In the real interaction, the stopping rule fires only on BadExp_H, so

$$\Delta\big(\mathsf{View}_{\mathrm{real}}, \mathsf{View}_{\mathrm{real}}^{\mathsf{stop}}\big) \leq \Pr[\mathsf{BadExp}_H].$$

Let Succ denote the successful execution, which samples each reached gadget conditioned on its own prescribed checks passing, rather than conditioning on the success of future gadgets. The real and successful executions can be coupled so that they agree until the first failed check, and the stopping rule acts identically on agreeing prefixes. Therefore

$$\Delta\big(\mathsf{View}_{\mathrm{real}}^{\mathsf{stop}}, \mathsf{View}_{\mathsf{Succ}}^{\mathsf{stop}}\big) \leq \Pr[\mathsf{BadCheck}] + \mathsf{negl}(\kappa).$$

We never condition on the absence of an exposure violation. In particular, the LRS challenge used below remains an unconditioned sample from $\mathsf{IPEnc}_d^\dagger$.

Programmed hybrids. Number closed encoding identifiers in public creation order, including the initial encoding. In hybrid H_j, the first j identifiers are independently sampled from $\mathsf{IPEnc}_d^\dagger(0)$ and the remaining identifiers from $\mathsf{IPEnc}_d^\dagger(z_e)$, where z_e denotes their intended logical values under the actual public inputs. All incident activations use the programmed joint reconstruction of lemma 10. Public-input pairs are sampled for their known public values. A transparent opening pair is sampled for the actual reachable public output and attached by the public-output gadget's sealing-refresh reconstruction; it is not itself a storage-hybrid target. Every sampled encoding, public reconstruction datum, and trace is retained and reused throughout its lifetime. Every hybrid applies the stopping rule; to keep the notation light, we write H_j for the stopped hybrid. The $j = 0$ hybrid has compatible endpoints and is therefore the successful auxiliary execution $\mathsf{View}_{\mathsf{Succ}}^{\mathsf{stop}}$, up to the negligible replay error.

Adjacent hybrids and adaptive inputs. Fix j, and let e be the closed encoding that differs between H_{j-1} and H_j. We show that these hybrids are statistically close by constructing an adversary for the fixed-message LRS experiment of lemma 1.

The reduction first simulates the execution until the encoding e is needed. Since the hybrids replace encodings in their order of creation, all previously created closed encodings have already been replaced by encodings of zero. The reduction can therefore generate the preceding execution, including all leakage answers and adaptively chosen public inputs, without accessing the LRS challenge.

At this point, the prescribed logical value z_e is determined by the initial logical state and the public inputs chosen so far. The reduction submits $(z_e, 0)$ as its two challenge secrets and uses the resulting LRS encoding as e.

More precisely, the reduction introduces the challenge before reconstructing any part of the gadget execution that depends on e. This is possible because the schedules of the DF gadgets are fixed: each output encoding has a unique producer, and no information depending on that encoding needs to be released before its prescribed logical value is determined. Thus, the preceding simulation is independent of the LRS challenge bit. In particular, the reduction makes no leakage queries to the challenge shares before choosing $(z_e, 0)$.

The reduction then continues the execution using the programmed reconstruction kernels. It generates every other encoding itself, according to the corresponding hybrid, and uses lemma 11 to simulate all observations involving e, including those arising in its producer gadget and subsequent uses. The reduction also applies the stopping rule, which it can evaluate since it sees the entire public prefix. Hence, on every path, the incident bundle of e meets at most H protected activations before the experiment stops, and by lemma 11 the total leakage from the challenge encoding is at most

$$H\lambda + \lceil N \rceil \le \Lambda_d.$$

This bound holds pathwise, as lemma 1 requires, and involves no conditioning on the challenge shares. The reduction retains all sampled encodings and reconstructed views, so subsequent adaptive leakage queries and public inputs are handled consistently, including across invocation boundaries.

Since the preceding execution is independent of the challenge bit, the fixed-message LRS guarantee applies even though z_e is determined during this simulation. Indeed, we may fix the randomness of the preceding simulation, apply lemma 1 to the resulting fixed challenge pair and continuation, and then average over that randomness.

The two challenge bits produce precisely H_{j-1} and H_j, respectively. Therefore, by lemma 1,

$$\Delta(\mathsf{H}_{j-1}, \mathsf{H}_j) \leq \epsilon_d(u_e).$$

For a dummy identifier, the adjacent hybrids are identical.

The final hybrid and online simulation. In H_M every closed encoding represents zero. This hybrid is implementable by a stateful online simulator receiving only the public inputs and their actual reachable public outputs: it samples zero encodings for all closed intermediate and retained state, reconstructs every gadget using the programmed kernels, and samples transparent openings from the prescribed public outputs. It also applies the stopping rule, which depends only on public data. The simulator does not perform an honest evaluation on secret zero; the offset kernels bridge its zero encodings to the prescribed public behavior. At an invocation boundary it passes the exact realized retained encoding and cached traces to the next invocation, without resampling them from their marginals. This preserves every previously released public and leakage prefix, including the conditional bias induced by earlier leakage.

Combining the hybrid argument with the two bounds of the stopping-rule paragraph, we obtain

$$\begin{aligned}\Delta\big(\mathsf{View}_{\text{real}}, \mathsf{View}_{\text{sim}}\big) \leq \sum_{e=1}^{M} \epsilon_d(u_e) + \Pr[\mathsf{BadCheck}] \\ + \Pr[\mathsf{BadExp}_H] + \mathsf{negl}(\kappa).\end{aligned} \tag{26}$$

By the parameter choices in the theorem and lemma 1, the hybrid sum is negligible. The probabilities of $\mathsf{BadCheck}$ and BadExp_H are negligible by the preceding analysis and the theorem's hypotheses, respectively. Hence the total simulation error is negligible, as required. This is the output simulation required by definition 3. □

C.8 Relation to Prior DF Reconstruction Techniques

Our proof of theorem 2 builds on the reconstruction techniques of [32, 33], developed in the only-computation-leaks (OCL) model. In particular, these works establish reconstruction of gadget views conditioned on their incident encodings, together with one-sided reconstruction using bounded communication. The one-field-element reconstruction for generalized multiplication is also established in [33].

The main distinction is that our leakage model is not restricted to the OCL setting. In addition to leakage from complete activation views, including local randomness, received messages, and intermediate values, we allow leakage from retained inactive memory. Consequently, our reconstruction argument must account for leakage across all activations involving a given encoding, including leakage obtained while that encoding is inactive.

Furthermore, theorem 2 establishes output simulation for stateful DF executions with adaptive public inputs and polynomially many invocations. Our proof therefore reconstructs complete activation traces, accounts for encoding-dependent public information across all incident gadgets, and preserves the exact retained encoding and previously released transcript across invocations. Public openings are handled separately from closed encodings.

D Register Scheduling and Exposure

We establish the register-scheduling and exposure bounds used by the universal compiler. Recall that a *block* implements one primitive operation, together with the operations required to carry

the remaining live logical registers to the next *stable boundary*. At each stable boundary, every live logical register has exactly one committed encoding.

Lemma 12 (Register scheduling and exposure)**.** *Consider the fixed register schedule of the universal compiler, using refresh, multiplication, copy, translation, and public output gadgets. For a computation with live logical width at most W, there exist an absolute constant $h_* = 6$ and a function $B(W) = O(W)$ such that:*

1. *Each block contains at most $B(W)$ constituent gadget invocations, each taking at most h_* party turns.*
2. *In every execution in which no gadget check and no sampler fails, every physical encoding is present in at most two consecutive blocks.*
3. *In such executions, the incident bundle of every closed encoding meets at most*

$$H_{\mathsf{cert}} \le 2h_*B(W) + 2 = 12B(W) + 2 = O(W)$$

protected activations. In arbitrary executions, $\Pr[\mathsf{BadExp}_{H_{\mathsf{cert}}}] \le \mathrm{poly}(\kappa)/Q + \mathsf{negl}(\kappa)$, so the compiler has exposure bound H_{cert} in the sense of definition 4.

In particular, the constant-width universal compiler has $H_{\mathsf{cert}} = O(1)$.

Proof: We verify the register bound, the encoding-lifetime invariant, and the exposure bound in turn.

Register allocation. At a stable boundary, at most W logical registers are live, each represented by one committed encoding. During the following block, operands are copied when both their use and a retained successor are needed. Registers not used by the primitive operation are carried through refreshes. Since a primitive arithmetic operation has at most two operands, at most two additional copy branches are needed. A copy is implemented by two constituent refresh invocations, rather than counted as one unit-cost gadget. Only a constant number of further temporary encodings are needed for the primitive result and its commitment.

It follows that the number of constituent gadget invocations in a block is bounded by

$$B(W) \le c_0(W + 1) + c_1$$

for absolute constants c_0, c_1 determined by the fixed gadget interfaces and register-allocation convention. In particular, $B(W) = O(W)$; no exact numerical value of $B(W)$ is needed.

Encoding lifetimes. Consider first an execution in which no gadget check and no sampler fails. The schedule maintains the following invariant: at every stable boundary, each live logical value has one committed encoding, and all predecessors present at the previous stable boundary have been retired. Indeed, each such predecessor is consumed, copied to a fresh successor, or carried through a refresh in the next block. Temporary branches are consumed or retired before that block ends.

Inductively, a physical encoding can be present only in its producing block and the following block, where it is consumed or retired. The same invariant applies to nested lookup procedures, sequential trace accumulation, norm exponentiation, and the refreshed persistent secret. At an invocation boundary, temporary encodings have been retired and only the prescribed persistent encoding is retained; its next use follows the same carry-and-retire schedule.

Exposure accounting. The fixed schedules of the five DF gadgets give the following conservative party-turn bounds. Commitment and cleanup are included.

Gadget	Party turns	Bound
Full-field refresh	Left, right, left, then commit	4
Generalized multiplication/compression	At most five turns, then commit	6
Two-output copy	Two constituent refresh invocations	$2h_*$ in total
Translation	Publish coordinate, refresh, then commit	At most 6
Public output	Refresh, open, and synchronize	At most 6

A copy is charged as two constituent refresh invocations. Hence every constituent gadget invocation takes at most $h_* = 6$ party turns. Since an encoding is present in at most two consecutive blocks, each containing at most $B(W)$ constituent gadget invocations, its exposure is bounded by

$$H_{\mathsf{cert}} \leq 2h_*B(W) + 2 = 12B(W) + 2.$$

The additional two turns conservatively account for query entry and exit. The count includes leakage from dormant encodings in the inactive component and from temporary shares and masks until their prescribed cleanup.

Failures. Failed padded attempts follow their fixed public failure schedules, and there are no retries within an invocation. However, a rollback retains the most recent committed encoding, so a failure can extend the lifetime of that encoding, possibly across invocations; for example, the initial copy of the secret may fail in several consecutive invocations. This is why items 2 and 3 are stated for failure-free executions. Now consider an arbitrary execution in which some closed encoding's incident bundle meets more than H_{cert} protected activations, and the first activation at which this happens. If no gadget check or sampler had failed before that activation, the execution up to that activation would coincide with a prefix of a failure-free execution with the same public inputs, since failures are the only events that change the public schedule and retention history for given public inputs. In that execution the bound H_{cert} holds, a contradiction. Hence $\mathsf{BadExp}_{H_{\mathsf{cert}}}$ is contained in the event that some gadget check or sampler fails. Each reached gadget fails with probability $O(1/Q)$, and each fixed-clock sampler fails with negligible probability (lemma 4). A union bound over the polynomially many gadget invocations gives

$$\Pr[\mathsf{BadExp}_{H_{\mathsf{cert}}}] \leq \mathrm{poly}(\kappa)/Q + \mathsf{negl}(\kappa),$$

which is negligible since $Q \geq 2^\kappa$.

Since $B(W) = O(W)$, we obtain $H_{\mathsf{cert}} = O(W)$. For the universal compiler, $W = O(1)$, and therefore $H_{\mathsf{cert}} = O(1)$, as claimed. □

E Hardware Replacement from Indistinguishability Obfuscation

Our compiler uses leak-free hardware only to sample fresh, inputless correlations. For fixed public parameters $\mathbb{K}, d$, a call to $\mathcal{O}'_{\mathbb{K},d}$ samples a publicly specified pair (Y_L, Y_R) and delivers the two coordinates privately to their respective components. Calls are independent and the oracle retains no state between calls. The sampler in (6) shows that this functionality is efficiently samplable. These properties permit the deniable-encryption and non-committing-encryption approach of [22, Section 3]. The algebraic relation between the outputs is needed by our gadgets, but is not a restriction imposed by this replacement.

For composition, we retain the correlation outputs in the simulation guarantee: a simulated public transcript is generated independently of those outputs, after which each component's local

execution is reconstructed using only its own output and output-independent simulator state. The proof below establishes this guarantee and accounts for leakage during the additional activations.

Corollary 8 (Hardware replacement from LWE and iO)**.** *Assume iO for polynomial-size circuits and the decisional LWE assumptions required by the non-committing encryption of [53]. Under the other hypotheses of theorem 3, in the honestly generated public-CRS model, all calls to $\mathcal{O}'_{\mathbb{K},d}$ can be replaced by a two-party protocol, preserving the transfer guarantee up to negligible error against PPT adversaries with efficient leakage functions. With fresh keys, fixed-clock execution, and the five-activation schedule below, the encoding-lifetime bound becomes*

$$H_{\mathrm{io}} = 2(h_* + 5)B(W) + 2 = 22B(W) + 2 = O(W) = O(1), \qquad h_* = 6.$$

Consequently every fixed persistent-share leakage rate $\rho < 1/(2H_{\mathrm{io}})$ is achievable. The replacement removes online leak-free hardware; leak-free initialization, erasures, and an honest public CRS remain required.

Proof: Write $Y_L = (A, \widetilde{A})$ and $Y_R = (B, \widetilde{B})$. Truncating the rejection procedures in (6) gives a fixed-clock sampler $\widehat{\Delta}_d$ negligibly close to the oracle distribution. Let $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be the NCE of [53]. Apply the concrete universal deniable-encryption transformation of Sahai and Waters [87, Section 4.2], with the interface described in [22, Definition 3.2], to the auxiliary bit-encryption scheme with key generation Gen and encryption

$$(Y_L, Y_R) \leftarrow \widehat{\Delta}_d; \qquad E^\star.\mathsf{Enc}(pk, b) = (Y_L, \mathsf{Enc}(pk, Y_R \| b)).$$

Its decryption ignores the first coordinate and returns the last bit. The resulting public programs $C_{\mathrm{enc}}, C_{\mathrm{exp}}$ form the CRS, for each public dimension used by the compiler. LWE supplies the NCE and one-way functions; iO supplies the deniable transformation together with those one-way functions.

Explanation security with receiver coins. The first hybrid below must retain the receiver's key-generation coins, since they are part of its leakable view. We justify this for the concrete Sahai–Waters transformation, rather than infer it from the statement of [22, Definition 3.2(b)], which does not include these coins. Write $(pk, sk) = \mathsf{Gen}(1^\kappa; \rho_R)$, and consider the auxiliary encryption scheme with key generation

$$E^{\star,+}.\mathsf{Gen}(1^\kappa; \rho_R) = \big(pk, (sk, \rho_R)\big).$$

Its encryption is exactly $E^\star.\mathsf{Enc}$, and its decryption ignores the added ρ_R. Thus its public-key and ciphertext distributions, correctness, and ordinary encryption security are unchanged. Sahai and Waters observe that their explanation security holds even given the underlying encryption scheme's secret key [87, footnote 2 and Section 4.1.2]. Applying this property to $E^{\star,+}$ gives security even given ρ_R. The universal transformation uses only the public encryption program, so the programs $C_{\mathrm{enc}}, C_{\mathrm{exp}}$ are unchanged. The augmentation is only a proof device; it changes neither the real protocol nor its storage and deletion schedule.

More explicitly, let $\mathsf{crs} = (C_{\mathrm{enc}}, C_{\mathrm{exp}})$ be generated independently of the honest receiver coins ρ_R. For fresh uniform r_L, set $\widehat{c} = (Y_L, c) = C_{\mathrm{enc}}(pk, 0; r_L)$, and let $r'_L \leftarrow C_{\mathrm{exp}}(pk, \widehat{c}, 0)$ use fresh explanation coins. The resulting guarantee is

$$\big(\mathsf{crs}, pk, \rho_R, \widehat{c}, r_L\big) \approx_c \big(\mathsf{crs}, pk, \rho_R, \widehat{c}, r'_L\big). \tag{27}$$

For completeness, the explanation-security hybrids of [87, Section 4.1.2] also justify this directly. In each iO step, the reduction samples ρ_R, computes its matching pk, and includes ρ_R in the

auxiliary information; the two obfuscated circuits remain functionally equivalent except on the same negligible exceptional events. In each puncturable-PRF or PRG step, the reduction likewise generates ρ_R itself, independently of the primitive's challenge randomness. None of these steps uses secrecy of the underlying encryption key. In the final hybrid, the true and explained sender coins are treated symmetrically even conditional on ρ_R. For the universal construction, the public encryption-program description is included in the PRF inputs [87, Section 4.2]; the same argument applies. This compares explanations of the same message, not encryptions of different messages after revealing a decryption key. It does not reveal the CRS-generation coins or the secret keys inside the obfuscated programs. Thus (27) uses only the same iO and one-way-function assumptions, with the latter already supplied by LWE.

Each call uses five protected activations:

1. The right component generates fresh (pk, sk) and sends pk.
2. The left evaluates $(Y_L, c) = C_{\mathrm{enc}}(pk, 0; r_L)$, retains Y_L, and sends only c.
3. The right decrypts c and retains Y_R from $Y_R\|0$.
4. The left commits its output and erases its temporary workspace.
5. The right commits its output and erases its key, coins, and remaining temporary workspace.

Every activation, including evaluation of the obfuscated program, leaks according to definition 2. The outputs remain until the consuming gadget deletes them; failures follow a fixed-clock exceptional branch and contribute negligible error.

We use the DLZ hybrids while explicitly retaining both outputs. By (27), first replace r_L by $C_{\mathrm{exp}}(pk, (Y_L, c), 0)$. Since pk and Y_R are computed from ρ_R and c, this step also retains their joint distribution with the receiver's coins and the sender's output Y_L. Next sample $(Y_L, Y_R) \leftarrow \widehat{\Delta}_d$ and $c \leftarrow \mathsf{Enc}(pk, Y_R\|0)$, retaining the explained left randomness tape. The deniable transformation's statistical output-distribution property and encryption correctness justify this step. Finally use the NCE simulator:

$$(\widetilde{pk}, \widetilde{c}, t) \leftarrow \mathsf{Sim}_1(1^\kappa), \qquad \widetilde{u}_R \leftarrow \mathsf{Sim}_{2,R}(Y_R\|0, t),$$

where t is its private first-stage state and $\mathsf{Sim}_{2,R}$ returns receiver key-generation coins. Reconstruct the left randomness tape as $C_{\mathrm{exp}}(\widetilde{pk}, (Y_L, \widetilde{c}), 0)$, discarding any NCE sender randomness tape. The NCE reduction retains the sampled Y_L as auxiliary information when submitting $Y_R\|0$. Thus the transcript is generated independently of the outputs; left reconstruction uses only Y_L and the transcript, and right reconstruction uses only Y_R, the transcript, and output-independent t. The state t is simulator-only and is not given to the left.

Replay these randomness tapes to reconstruct all local intermediate values and dormant workspace, reusing them across adaptive leakage queries. Composition with same-side compiler state introduces no opposite-side input. A sequential hybrid over polynomially many calls incurs only negligible computational and statistical error. For the explanation-security step with a reused CRS, the reduction generates the other calls using the public programs and samples their fresh receiver coins itself. Thus the single-call argument above applies while retaining the preceding transcript.

Each gadget has at most one correlation call; the two refreshes of a copy are separate gadgets. Calls are not interleaved with compiler gadgets, and temporary sampling workspace is erased before proceeding. Since each stable block contains at most $B(W)$ gadget slots, adding five activations per gadget to the producing and following stable blocks, plus the two boundary activations, gives the stated H_{io}. The temporal reconstruction argument therefore applies with

$$H_{\mathrm{io}}\lambda + \lceil N \rceil \le \Lambda_\nu, \qquad N = \log_2 |\mathbb{K}|.$$

The existing parameter choice then achieves every fixed $\rho < 1/(2H_{\mathrm{io}})$ for sufficiently large constant ν. $\square$