How to Compute under $\mathcal{AC}^0$ Leakage without Secure Hardware

Abstract

We study the problem of computing securely in the presence of leakage on the computation's internals. Our main result is a general compiler that compiles any algorithm P, viewed as a boolean circuit, into a functionally equivalent algorithm P'. The compiled P' can then be run repeatedly on adversarially chosen inputs in the presence of leakage on its internals. In particular, in each execution of P', an adversary can (adaptively) choose any leakage function that can be computed in $\mathcal{AC}^0$ and has bounded output length, apply it to the values on P''s internal wires in that execution, and view its output. We show that no such leakage adversary can learn more than P's input-output behavior. In particular, the internals of P are protected.

Security does not rely on any secure hardware, and is proved under a computational intractability assumption regarding the hardness of computing inner products for $\mathcal{AC}^0$ circuits with pre-processing. This new assumption has connections to long-standing open problems in complexity theory.

1 Introduction

Modern cryptographic algorithms are often modeled as "black boxes". It is commonly assumed that their keys, internal computations, and randomness are opaque to external adversaries. In practice, however, these algorithms might be run in adversarial settings where keys are compromised and computations are not be fully opaque, e.g. because of side channel attacks.

Side channel attacks exploit the physical implementation of cryptographic algorithms. The physical implementation might enable observations and measurements on the cryptographic algorithm's internals. Attacks such as these can and have broken systems with a mathematical security proof, without violating any of the underlying mathematical principles. (see [KJJ99, RCL] for just two examples). A growing body of recent research on *cryptography resilient to leakage or side-channel attacks* aims to build robust mathematical models of realistic side channel attacks, and to develop methods grounded in modern cryptography to provably resist such attacks.

One line of research considers constructing specific primitives (e.g. encryption schemes) resilient to side-channel attacks, e.g. [AGV09, BKKV10]. A different line, which we pursue in this work, considers leakage-resilience compilers for transforming general algorithms (represented as stateful circuits) into functionally equivalent stateful circuits that are resilient to side-channel attacks on any polynomial number of executions (the polynomial is not fixed in advance). Typically, these results consider a family $\mathcal{L}$ of leakage attacks, and show (via a simulation argument) that even an adversary who can adaptively choose inputs to each execution, and launch a leakage attack from the family $\mathcal{L}$ on each execution, learns no more about the underlying functionality than it would from black-box (i.e. input-output) access. It is usually assumed that the initial circuit compilation is done (once only) without any leakage. Afterwards, every execution of this stateful circuit—including any and all state updates—is subject to leakage attacks from the family $\mathcal{L}$.

An imperative goal for research on leakage-resilience compilers, from both a foundational and a practical perspective, is handling leakage from families $\mathcal{L}$ that are as rich and expressive as possible. There are, however, inherent limitations on the leakage functions that can be tolerated in each execution. For example, they must be of bounded length—otherwise the entire circuit internals can leak, and we enter the more challenging domain of code obfuscation. In particular, the impossibility results of Barak *et al.* [BGI$^+$01] imply that there does not exist a leakage-resilience compiler that can protect against leakage of unbounded length. In fact, the impossibility result of [BGI$^+$01] can be used to show that there is no leakage-resilience compiler that protects against even *a single bit* of polynomial-time leakage.

Given this impossibility, work on leakage-resilience compilers has considered various additional restrictions on the leakage functions (on top of bounded output length):

Wire-Probe/Bit Leakage. Ishai Sahai and Wagner [ISW03] considered leakage functions that expose the values on a bounded number of wires in each circuit evaluation (the computation is viewed as a boolean circuit). For this class of leakage functions they show (unconditionally) a leakage-resilience compiler for general circuits. Ajtai [Ajt11] considered the RAM model. He divided each RAM computation into sub-computations, and considered leakage functions that exposed a constant fraction of the memory words involved in each sub-computation. He obtained a leakage-resilience compiler for general RAM computations. The leakage model is qualitatively similar to that of [ISW03], in the sense that the (length bounded) leakage operates separately on each bit of the computation—either exposing it in its entirety or revealing nothing at all. We view the main (and important) improvement in [Ajt11] versus [ISW03] as the quantitative improvement to fraction of leaked bits that can be tolerated in each execution of the transformed computation.

Computationally Bounded Leakage. Faust, Rabin, Reyzin, Tromer and Vaikuntanathan [FRR+10] considered leakage functions with two restrictions: (*i*) the functions are *computationally bounded*, e.g. capable of computing only $\mathcal{AC}^0$ functions of the values on the circuit's wires,[1] and (*ii*) the computation can use *perfectly secure hardware components*, whose internals never leak. We view this second assumption as a strong restriction on the leakage functions: they cannot even compute a single bit of information about the internals of the secure hardware components, which are used multiple times throughout the execution. For this family of leakage functions, Faust *et al.* showed (unconditionally) a general leakage-resilience compiler for transforming any circuit.

$\mathcal{AC}^0$ leakage is a qualitatively richer class than wire-probe leakage. In particular, for a fixed leakage bound λ, $\mathcal{AC}^0$ leakage can output the values of λ wires, but potentially also much more (e.g. the AND of all circuit wire values). The results of [FRR+10], however, are incomparable to [ISW03] because of the secure hardware restriction.

Only-Computation (OC) Leakage. Goldwasser and Rothblum [GR10] and Juma and Vhalis [JV10] considered leakage under the restriction that *"only computation leaks information"*, as pioneered by Micali and Reyzin [MR04]. Here, each execution of the algorithm is divided into ordered sub-computations, and the leakage function operates separately (if adaptively) on each sub-computation. Those works also assumed the existence of perfect (simple) secure hardware components, and showed general leakage-resilience compilers under different cryptographic assumptions. More recently, Goldwasser and Rothblum [GR12] showed how to remove both the use of secure hardware and the computational assumption, obtaining an unconditional compiler for protecting computations from the "only-computation leaks information" (OC) family of leakage functions.

Comparing this to the other models described above, we note that (for a fixed leakage bound) OC leakage is qualitatively richer than wire probe leakage, but incomparable to $\mathcal{AC}^0$ leakage.

1.1 This Work

Our main result is a leakage-resilience compiler for length-bounded $\mathcal{AC}^0$ leakage *that does not use secure hardware*. The compiler's security is proved based on an (unproven) assumption about $\mathcal{AC}^0$-hardness of computing inner products with pre-processing.

$\lambda(\cdot)$-IPPP Assumption (informal). The Inner-Product with Pre-Processing (IPPP) Problem considers predicting the inner product of two uniformly random vectors $x, y \in \{0,1\}^\kappa$ using an $\mathcal{AC}^0$ circuit and an arbitrary polynomial-time pre-processing step that is *run separately* on x and on y (with polynomial output length). Without pre-processing, it is known that predicting the inner product is hard for $\mathcal{AC}^0$ [Raz87, Smo87]. With *joint* pre-processing on x and y, one can simply compute the inner product, and so the problem becomes easy. With (polynomial time) pre-processing that is run separately on x and on y, there is no known $\mathcal{AC}^0$ predictor with non-negligible advantage (we emphasize that the pre-processing step's output length can be polynomial). This question has been explicitly considered in the literature for some time (e.g. [BFS86]). We introduce the 1-IPPP Assumption, which says that even given polynomial-time pre-processing (separately on x and on y), no $\mathcal{AC}^0$ circuit ensemble can predict the inner product with non-negligible advantage.

More generally, we consider also the problem of *compressing* the instance size of inner product from κ to $\lambda(\kappa) < \kappa$ bits using an $\mathcal{AC}^0$ circuit and arbitrary polynomial-time pre-processing on x and on y separately. By "compressing", we mean that the instance size is reduced while still maintaining noticeable statistical difference between the distribution of YES and NO instances (inner product 1 and 0 respectively), see [HN10, DI06]. Without pre-processing, Dubrov and Ishai [DI06] showed

[1] Their full result is actually more general, and can handle $ACC^0[p]$ or noisy leakage

that it is hard for $\mathcal{AC}^0$ to compress parity instances to sub-linear length $\kappa^{1-\delta}$ (in fact this was later used in the result of [FRR+10]). We introduce the $\lambda(\kappa)$-IPPP Assumption, which says that even given polynomial-time pre-processing (on x and on y, each separately), no $\mathcal{AC}^0$ circuit ensemble can compress the instance size of inner product to length $\lambda(\kappa)$.

See Section 2.2 and Appendix A for formal definitions and a further study of IPPP.

Main result. Our main result is a leakage resilience compiler for $\mathcal{AC}^0$ leakage. Security relies on the $\lambda(\cdot)$-IPPP assumption. For security parameter κ, the transformed circuit is resilient to $\lambda(\kappa)$ bits of $\mathcal{AC}^0$ leakage from each execution.

Theorem 1.1. *For any function $\lambda(\cdot) : \mathbb{N} \to \mathbb{N}$, under the $\lambda(\cdot)$-IPPP assumption there exists a leakage resilience compiler for $\mathcal{AC}^0$ leakage. For security parameter $\kappa \in \mathbb{N}$, and for a $\mathrm{poly}(\kappa)$-size input circuit C to be transformed, the adversary's runtime can be polynomial in κ, and the leakage from each execution can be any (adaptively chosen) $\mathcal{AC}^0$ function of length $\lambda(\kappa)$. The size of the transformed circuit is $O(|C| \cdot \kappa^3)$.*

See Definition 2.3 for the formal definition of a secure compiler for $\mathcal{AC}^0$ leakage.

We emphasize that the leakage bound in this result is equal to the "compression" factor in the IPPP assumption. In particular, assuming the $\lambda(\kappa)$-IPPP Assumption for all sub-linear $\lambda(\kappa) = \kappa^{1-\delta}$ (we find this to be a plausible assumption), we get resilience to any sub-linear leakage amount of leakage per execution (similar to the leakage bound in [FRR+10]). We remark that even a compiler that handles only a single bit of leakage from each execution is already non-trivial—in particular, since the number of executions is an unbounded polynomial, the total combined leakage from the repeated executions can be much larger than the size of the transformed circuit. We also recall that for polynomial-time leakage (i.e. leakage not restricted to $\mathcal{AC}^0$), it is impossible to build a leakage-resilience compiler that handles even a single bit of leakage (see above).

Comparison to prior work. We now compare the result of Theorem 1.1 to prior works on leakage-resilience compilers (see also the discussion above on these prior works).

Wire Probe Leakage. Comparing to the work of [ISW03] on wire probe leakage and to the more recent work of Ajtai [Ajt11], the main novelty of our result is in handling the richer class of $\mathcal{AC}^0$ leakage. On the other hand, those results did not rely on unproven assumptions and the quantitative leakage bounds (as a fraction of the transformed computation's size) were better.

$\mathcal{AC}^0$ Leakage. The most closely related work is that of Faust *et al.* [FRR+10], and their construction is a starting point for ours (see below). The main added benefit of our work is in removing the secure hardware assumption (again, this can be viewed as handling a larger class of leakage functions). The main qualitative disadvantage is in the introduction of the unproved IPPP assumption. Quantitatively, the amount of leakage we can handle depends on the function $\lambda(\cdot)$ for which IPPP is hard. If we assume hardness for any sub-linear $\lambda(\cdot)$ function, we get similar leakage bounds to [FRR+10]. Finally, the circuit blowup of the [FRR+10] compiler is κ^2, whereas ours is κ^3.

OC Leakage. Our end result is qualitatively incomparable to Goldwasser and Rothblum [GR12], because only-computation leakage and $\mathcal{AC}^0$ leakage are incomparable. We do note, however, that their result is unconditional and the circuit blowup is smaller (κ^ω, where ω is the exponent for matrix multiplication, versus κ^3 in our work). Both our work and theirs tackle the challenge of leakage-resilience compilation for a rich class of leakage functions without using secure hardware. In fact, we use the "ciphertext bank" machinery introduced in [GR12] for handling this challenge.

Our high-level approach is to build on the construction of [FRR+10] and remove the secure hardware using the techniques of [GR10]. Unfortunately, this is far from straightforward. In

a nutshell, the main technical challenge is constructing an $\mathcal{AC}^0$ and length-preserving security reduction from the problem of compressing inner product instances (or rather from the IPPP problem), to distinguishing real and simulated leakage on repeated executions of the transformed circuit. The main issue is that the [GR10] machinery relies (extensively) on computations that cannot be performed in $\mathcal{AC}^0$ (e.g. matrix multiplication). We elaborate in Section 1.2.

1.2 Overview of the Construction and Security Reduction

At a high level, one central difficulty in leakage-resilience compilation without secure hardware is that it requires simulating a *complete view of multiple executions in their entirety*. The simulated view needs to be indistinguishable from the real execution under a wide family of leakage functions. Intuitively, secure hardware makes the simulation task considerably easier because some regions of the computation are opaque to the leakage, and their internals need not be simulated. Work on specific leakage-resilient cryptographic primitives (e.g. [AGV09, DP08, BKKV10]) avoids this difficulty because the security definitions do not require simulation. We follow [GR12] in taking on this simulation challenge.

The simulator has no knowledge of the computation's internals (beyond its input and output). Moreover, the *entire computation* in the simulated view should be consistent with the initial state and the choice of random coins; otherwise, an $\mathcal{AC}^0$ leakage function that checks consistency of the internal computations will distinguish the real and simulated views. This means that *the simulator cannot make any "illegal" computational steps*, but presumably is still acting very differently from the real execution. Note that this is a crucial difference from the setting where secure hardware is used: the simulated outputs of the secure hardware can essentially be an "illegal" output that would never be generated in a real execution (but the leakage functions cannot tell the difference).[2] Indeed, this is what the [FRR$^+$10] simulator does (see more below).

In our setting, without secure hardware, the simulator cannot make any "illegal" steps. Its only freedom to diverge from a "real" execution is in generating the initial state (where there is no leakage), and in choosing the random coins. Our simulator generates (only once) a "trapdoor" initial state, and this gives it extensive freedom in shaping the simulated view, even under repeated leakage from multiple executions and state updates.

Against this backdrop, we recall the approach of [FRR$^+$10]. They proved security by showing a reduction from compressing an instance of the inner product problem (or rather the problem of computing a vector's parity) to distinguishing the real and simulated views (or rather various hybrids of these views). They showed that the security reduction can be computed using length-bounded $\mathcal{AC}^0$ access to the inner product instance, and so the real and simulated views are statistically close. Our high-level approach is to build on the construction of [FRR$^+$10], and remove the secure hardware using the techniques of [GR10]. As hinted above, this is far from straightforward, mainly because running the [GR10] machinery requires computations, such as matrix multiplication, that cannot be implemented in $\mathcal{AC}^0$. This creates (multiple) difficulties in implementing a reduction from compressing inner product instances to distinguishing the real and simulated views (or hybrids thereof) that only uses length-bounded $\mathcal{AC}^0$ access to the inner product instance. We relax the requirement from the security reduction: we reduce from the IPPP problem, rather than the full-blown inner product problem without pre-processing. We use the additional power of pre-processing to implement a reduction from the IPPP problem to distinguishing (hybrids of) the real and simulated views. This is our main technical contribution.

[2] This difficulty is avoided in [ISW03] as their leakage functions are too weak to check consistency of the computation.

We proceed with an overview of our construction and security proof. In what follows we restrict our attention to transforming *stateless* computations to be leakage resilient, but the results all hold also for stateful circuits (as considered say in [FRR+10]). We note that the transformed computations will be stateful circuits (even if the original computation is stateless), and their state is updated (under leakage) between executions.

Overview of [FRR+10] In the [FRR+10] construction, every wire i in the original circuit, say carrying value a_i (on a certain input), was replaced by a *bundle* of κ wires carrying a uniformly random vector of bits whose parity/XOR is a_i. Intuitively, an adversary with bounded-length $\mathcal{AC}^0$ leakage access to all of an execution's wire-bundles cannot distinguish the true value on any wire (i.e. the parity on any wire-bundle), and so the adversary learns nothing about the internals of the original computation. The main challenge is for the transformed circuit to emulate the computation of each gate in the original circuit, while maintaining the invariant that the bundle corresponding to each wire is a uniformly random vector with the correct parity, and without leaking anything about the parity. For example, to emulate an AND gate, the transformed circuit needs to take two wire bundles and output a wire bundle carrying a uniformly random vector whose parity is the AND of the parities of the input wire bundles.

The gate computations of the original circuits are emulated using "gate gadgets", one for every gate in the original circuit. In their elegant security proof, [FRR+10] separate the wires of the transformed circuit into the "external wires" described above, where each *wire* in the original circuit corresponds to the κ *"external wires"* in the transformed circuit. In each execution (on a certain input), these κ wires carry a bundle whose parity equals the wire's value in the original circuit (on that input). Each *gate* in the original circuit is replaced by a *"gate gadget"* in the transformed circuit. We call the wires in these gate gadgets "internal wires". In their security proof, [FRR+10] show that: (i) the *external wire distributions* (the distributions of values on the external wires) in the real and simulated executions are indistinguishable using bounded length $\mathcal{AC}^0$ leakage. As sketched above, this follows from the hardness of compressing parity instances in $\mathcal{AC}^0$. Then (ii) they show that the values on the internal wires of each gate gadget can be simulated *in* $\mathcal{AC}^0$ given only the gate gadget's input and output external wires. In fact, the simulation for gate gadgets is not perfect, but rather indistinguishable under bounded length $\mathcal{AC}^0$ leakage. The $\mathcal{AC}^0$ simulators for gate gadgets are called *reconstruction procedures*, and their existence guarantees that indistinguishability of the external wire distributions in the real and simulated executions implies indistinguishability of the complete view (including the internal wires of the gate gadgets).

Given this framework, the main challenge is implementing the gate gadgets and their $\mathcal{AC}^0$ reconstruction procedures. This is where the secure hardware devices come into play. The secure hardware outputs a uniformly random bundle of κ values with parity 0. The simulator can simulate the secure hardware's output to be a vector with any desired parity, and the leakage cannot tell the difference. As an example of how such a device is used, consider the (relatively simple) case of addition: the gadget gets two input wire bundles, $\vec{d_i}$ and $\vec{d_j}$ and computes the pairwise XORs of their bits (call this bundle $\vec{q}$). Rather than simply output this $\vec{q}$ (which already has the correct XOR), the gadget calls the secure hardware to compute a bundle $\vec{o}$ whose XOR is 0, and finally outputs the pairwise XOR of $\vec{q}$ and this "masking" bundle $\vec{o}$. This is not the only way in which the secure hardware is used (e.g. it is called more extensively for multiplication gates), but it is instructive. Intuitively, masking the gadget's output with the output $\vec{o}$ of the secure hardware helps secure simulation: the gate gadget's reconstruction procedure has some freedom in choosing a bundle (with arbitrary XOR) as the output of the secure hardware. Another important point here is that using the secure hardware "erases" any accumulated leakage on the input bundles: the

gadget's output bundle is statistically independent (given its XOR) from the input bundles. In particular, for any given values for the gate gadget's input and output bundles, we can (in $\mathcal{AC}^0$) compute an output of the secure hardware (i.e. a "secure hardware" bundle) for which the gate gadget's, on the given input bundles, outputs the given output bundle.

Construction. The secure hardware in [FRR+10] generates a random bundle whose parity is 0, but can be simulated as having generated bundles whose parity is 0 or 1 (as needed by the simulator). An initial idea is simply to pre-compute all the needed 0-bundles as part of the initial state (i.e. without leakage). The simulator can then choose whatever bundles it wants (0 or 1) to put in the initial state. The main drawback to this approach, of course, is that we can only pre-compute a finite and bounded number of these bundles, and so this construction cannot support repeated executions (alternatively, the initial state grows linearly with the number of executions).

This recalls the situation in [GR12]. They suggested using a "ciphertext bank". Translating that idea to our setting, our construction can use a small number of pre-computed 0-bundles, a *"bundle bank"*, to generate an essentially unbounded (polynomial) number of new 0-bundles. The simulator can generate an (illegally formed) initial bundle bank, and then control each subsequent generation arbitrarily to create a 0-bundle or a 1-bundle. All of this is done in a way that is indistinguishable, even under repeated $\mathcal{AC}^0$ leakage, from the real execution.

We implement the bundle banks as follows. Recall that each bundle encodes a bit $b \in \{0,1\}$ as a vector in $\{0,1\}^\kappa$ with parity b. The initial bundle bank includes 2κ such bundles, whose parities (in the real execution) are all 0. Within each execution of the circuit we generate 0-bundles as needed by taking random linear combinations of the bundle bank (a random linear combination of vectors whose parities are all 0 also has parity 0). Between executions we "regenerate" or update the entire bundle bank by taking 2κ new random linear combinations, and then we erase the old bundle bank. In the simulation, the initial bundle bank is generated so that each bundle is a uniformly random vector encoding a uniformly random bit. Now when we generate a new bundle, some linear combinations of the bundles in the bank give a new bundle encoding 0, and some give a new bundle encoding 1. The simulator chooses a uniformly random linear combination yielding an encoding of whatever value it wants. Between executions, as in the real view, the bundle bank is refreshed by taking 2κ new random linear combination, giving a new bank of uniformly random bit encodings (independent of the old bank). The full construction is in Section 3.

We note that our implementation of the bundle bank is considerably simpler than the ciphertext banks of [GR12]. In particular, there is no need for their "piecemeal matrix multiplication" procedure, and we compute matrix multiplication using the straightforward naive procedure (in time κ^3). We also remark that proving the security properties of the bundle bank in our setting requires completely different arguments (due to the completely different nature of the leakage attack).

Security. The intuition for indistinguishability of the real and simulated views is that an adversary cannot distinguish (under bounded length $\mathcal{AC}^0$ leakage) whether the bundles in the bank encode 0's (real execution) or are uniformly random (simulated execution). Nor can the adversary distinguish whether the linear combinations are uniformly random (real execution) or chosen from a $(\kappa-1)$-dimensional subspace so as to fix the value of the resulting bundle to 0 or 1 (simulation).

Transforming this intuition into a reduction from compressing parity/inner-product instances to distinguishing the real and simulated views, however, is far from straightforward. The major source of difficulty is that neither the real construction nor the simulator are actually $\mathcal{AC}^0$ algorithms, and neither are the computations in the "gate gadgets" used to emulate each gate in the original circuit.[3] In particular, if we want to use the bundle bank machinery, the gate gadgets need to

[3]We note that a similar difficulty also arose in [FRR+10], where in the real view (but not the simulated one) the

compute linear combinations of bundles in the bank. Moreover, this difficulty is compounded by the fact that, unlike [FRR+10], we cannot use leakage-free hardware to make the bundles used between gates and across executions statistically independent of each other. Even within a single circuit execution, this is already a serious concern. Each bundle is dependant on the bank, and through it on all other bundles. If (as seems natural) we want to use hybrid arguments to focus on differences in the distribution of a single bundle or gate emulation, we need to generate the rest of the view (on which both hybrids agree). This generation, however, is both not in $\mathcal{AC}^0$ and is not independent of the hybrid bundle. We now give intuition for how these two difficulties are overcome, further details are in Section 3.

Step *I*: $\mathcal{AC}^0$ reconstruction via "beefed up" external wire distributions. Our first step towards resolving these difficulties is to add more information to the *external wire distribution* so that we can reconstruct the internal view of each gate-gadget in $\mathcal{AC}^0$. For example, adapting the addition gate gadget of [FRR+10] to our setting, we take the bundle bank to be a matrix $G \in \{0,1\}^{\kappa \times 2\kappa}$. Our addition gate gadget takes as input two bundles $\vec{d_i}$ and $\vec{d_j}$, generates a "masking" 0-bundle $\vec{o} = G \times \vec{r}$ using the bundle bank and a random linear combination $\vec{r} \in_R \{0,1\}^{2\kappa}$. It then computes an output bundle $\vec{d_k} = ((\vec{d_i} + \vec{d_j}) + \vec{o})$. The reconstruction procedure gets the input and output bundles $\vec{d_i}, \vec{d_j}, \vec{d_k}$, as well as the bundle bank G (a $\kappa \times 2\kappa$ matrix), and needs to generate the internal view of the computation. This requires finding a vector $\vec{r}$ s.t. $G \times \vec{r} = (\vec{d_k} - (\vec{d_i} + \vec{d_k}))$ and generating the wire values of the matrix-vector multiplication $G \times \vec{r}$. For arbitrary $\vec{d_i}, \vec{d_j}, \vec{d_k}$ and G this cannot be done in $\mathcal{AC}^0$. To resolve this, we give the reconstruction some additional "advice"; We "beef up" the external wire distribution with vectors $\vec{r_i}, \vec{r_j}, \vec{r_k}$ s.t. $\vec{d_i} = G \times \vec{r_i}$, $\vec{d_j} = G \times \vec{r_j}$, and $\vec{d_k} = G \times \vec{r_k}$, and with all the wire values for computing these matrix-vector multiplications. The reconstruction procedure (for addition gates) can now compute in $\mathcal{AC}^0$ the vector $\vec{r_o} = G \times (\vec{r_k} - (\vec{r_i} - \vec{r_j}))$ and (by linearity of matrix multiplication) the wire values for the matrix-vector multiplication $\vec{o} = G \times \vec{r_o}$ to output a consistent view of the addition gate gadget's internal wires. We show that when the external wire distribution is distributed as in the real or simulated execution, our reconstruction procedures generate an indistinguishable view for the internal wires of each gate gadget. We note that reconstructing the multiplication gadget is significantly more complicated, and requires further "beefing up" of the external wire distribution.

Step *II*: indistinguishability of "beefed up" external wire distributions. Given the $\mathcal{AC}^0$ reconstruction procedures, it remains to argue that the "beefed up" external wire distributions of the real and simulated executions are indistinguishable under bounded-length $\mathcal{AC}^0$ leakage. The external wire distribution now includes a lot of information about the bundles, and in particular the bundles are no longer independent of each other because they are tied together via the bundle bank G and, for each bundle $\vec{d}$, the vector $\vec{r}$ for which $\vec{d} = G \times \vec{r}$. We will argue indistinguishability using a hybrid argument, replacing the bundle bank from real (i.e. 0 parities) to simulated (i.e. uniform), and replacing the parities of bundles on the external wires one-by-one from real to simulated values. As noted above, using a hybrid argument over the bundles one-by-one creates a challenge because the bundles (which all depend on the same bundle bank) are not independent of each other.

To use a hybrid argument over bundles, we *decompose* the execution's view into two parts: the first part depends on the bundle bank, *but not on the hybrid bundle*, and contains essentially all the information needed to generate the parts of the computation that do not involve the hybrid bundle.

emulation of multiplication gates required computing parities. In their work this difficulty was solved using the secure hardware (essentially replacing each real emulation of a multiplication gate with an indistinguishable emulation that could be computed in $\mathcal{AC}^0$). We, on the other hand, want to avoid the use of secure hardware.

The second part depends only on the randomness used to form the hybrid bundle. I.e., if $\vec{d}$ is the hybrid bundle then we take $\vec{r}$ to be the linear combination of the bundle bank that yields $\vec{d}$. We use $\vec{r}$ to pre-compute non-$\mathcal{AC}^0$ information that helps in generating the part of the view involving the hybrid bundle. The key point is that we give *an* $\mathcal{AC}^0$ *procedure* for combining these two separate parts and generating the entire view of one of the hybrid distributions. Of the two hybrids, which one is generated is determined by the inner product of $\vec{r}$ with a vector $\vec{x}$ that depends only on the bundle bank. Now, by the IPPP assumption (assuming we can use $\vec{x}$ to generate the bundle bank), we know that even given these two "pre-processed" parts of the view (computed from $\vec{x}$ and $\vec{r}$ separately), we can combine them in $\mathcal{AC}^0$ to get one of the hybrid external wire distributions, and thus no $\mathcal{AC}^0$ leakage on the hybrid can be statistically correlated with the inner product of $\vec{x}$ and $\vec{r}$. Thus, under the IPPP Assumption, $\mathcal{AC}^0$ leakage should not be able to distinguish the hybrids.

In summary, we construct (hybrids of) the real and simulated external wire distributions using "pre-processing", where two pre-computed pieces are combined in $\mathcal{AC}^0$ to give the appropriate external wire distribution. We get a reduction from the IPPP Problem to distinguishing the real and simulated external wire distributions.

2 Model and Definitions

Preliminaries. For a vector or string x we denote by $|x|$ the length of the vector, and by x_i or $x[i]$ the i'th item in the vector. We denote by $x|_i$ the restriction of a vector x to its first i items. For a vector x in $\{0,1\}^n$ we say that x is an *encoding* of $b \in \{0,1\}$ is the XOR (or sum over $\mathbb{GF}[2]$) of x's entries equals b. For vectors $\vec{x}, \vec{y} \in \{0,1\}^\kappa$, we use $\vec{x} + \vec{y}$ to denote bitwise addition over $\mathbb{GF}[[]2]$ (i.e. XOR) of the vectors' entries. We use $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n$ to denote the unit vectors over $\{0,1\}^n$ (n will be clear from the context), and we use U_n to denote the uniform distribution over $\{0,1\}^n$. For a finite set S we denote by $y \in_R S$ that y is a uniformly distributed sample from S. For a distribution D we use $y \sim D$ to denote the experiment of sampling y by D. We use $\Delta(D, F)$ to denote the statistical (L_1) distance between distributions D and F. For a circuit (or function) C and an oracle PPTM M we use $M^C(x)$ to denote M run on input x with oracle access to C.

2.1 Leakage Attacks and Security

Definition 2.1 ($\mathcal{AC}^0$-Leakage Attack $\mathcal{A}^\lambda[C, \mathit{Update}](1^\kappa)$)**.** A *continual* λ-bit $\mathcal{AC}^0$-leakage attack of adversary $\mathcal{A}$ on C with update procedure *Update* proceeds in rounds. In each round $t = 1, 2, \ldots$, there is a circuit C_t computed in the previous round (where $C_1 = C$). The adversary $\mathcal{A}$ chooses an input x for C_t and an $\mathcal{AC}^0$ leakage function ℓ, which takes as input: (i) the entire computation of C on input x (including all circuit wire values and all random coins), (ii) the entire computation of the update procedure *Update* run on C_t and outputting C_{t+1} (including all circuit wire values and all random coins of *Update*). The adversary's view in round t is $\mathit{view}_t = (x, C_t(x), \ell(\text{entire computation of } C(x) \text{ and } \mathit{Update}(C_t)))$. The adversary's choices of the input x and the leakage ℓ are adaptive can depend on the views in all previous rounds.

The attack proceeds for T rounds (where T is chosen by the adversary). The attack's output (or adversary view in the attack) is $\mathit{view} = (\mathit{view}_1, \mathit{view}_2, \ldots, \mathit{view}_T)$. The running time of $\mathcal{A}$ is its total running time in the attack, i.e. a polynomial-time adversary can only run for $\text{poly}(\kappa)$ rounds.

Remark 2.2. *Throughout this work when any of our algorithms compute matrix multiplication (or matrix-vector/vector-vector multiplication), this is done in the straightforward (if inefficient) way. The* partial sums *are the sub-computations in the multiplication, e.g. in computing the inner*

product $\langle x, y\rangle$, the partial sums are $(\langle x|_1, y|_1\rangle, \langle x|_2, y|_2\rangle, \ldots, \langle x, y\rangle)$. The leakage on the computation takes all of these partial sums as a part of its input.

Definition 2.3 ($\lambda(\cdot)$-Secure Compiler for $\mathcal{AC}^0$ leakage)**.** Let *Init* and *Update* be two PPTMs that take as input a circuit C and security parameter κ. We say that (*Init*, *Update*) is a $\lambda(\cdot)$-secure compiler for $\mathcal{AC}^0$ leakage, for any circuit C the following two requirements hold:

- Functionality: the circuits $C_1 = \mathit{Init}(C, \kappa), C_2 = \mathit{Update}(C_1, \kappa), C_3 = \mathit{Update}(C_2, \kappa), \ldots$ are each with all but negligible probability functionally equivalent to the original circuit C. For all $i \geq 1$ the circuits C_i are of the same size.
- Security: for any PPTM adversary $\mathcal{A}$ there exists a PPTM simulator $\mathcal{S}$ such that the view $\mathcal{A}^{\lambda(\kappa)}[\mathit{Init}(C, \kappa), \mathit{Update}(\cdot, \kappa)](1^\kappa)$ of the adversary in an $\mathcal{AC}^0$ leakage attack is indistinguishable from the view $\mathcal{S}^C(1^\kappa)$ generated by the simulator from black-box access to C. Note that there is no leakage on the *Init* procedure.

Remark 2.4. *Note we may consider many relaxations and strengthenings of this definition. For example, we could relax functionality to require only that each C_i is functionally equivalent to C w.h.p. on each input over the coins of the Init and Update procedures and/or the coins of C_i. We could also strengthen security to restrict the simulator $\mathcal{S}$ to only have oracle access to the inputs queried by the adversary (and in fact our construction meets this more stringent requirement). The choices in the definition above were made mainly for the sake of simplicity.*

2.2 The IPPP Problem and Assumption

We give formal definitions of the IPPP Problem and Assumption, see also the discussion in the introduction. For lack of space, further discussion has been moved to Appendix A. That appendix includes connections to problems in complexity theory, as well as further properties such as a worst-case to average-case hardness reduction.

Problem 2.5 ($(\lambda(\cdot), s(\cdot), \varepsilon(\cdot))$-Inner Product with Pre-Processing (IPPP))**.** *A triplet $(\mathcal{C}, \mathcal{C}^1, \mathcal{C}^2)$ of circuits ensembles solves the $(\lambda(\cdot), s(\cdot), \varepsilon(\cdot))$-Inner Product with Pre-Processing Problem (IPPP) if:*

1. *there exists a polynomial $p(\cdot) : \mathbb{N} \to \mathbb{N}$, such that $\forall \kappa \in \mathbb{N}$:*
   (a) *$\mathcal{C}^1_\kappa$ and $\mathcal{C}^2_\kappa$ are of size at most $p(\kappa)$, with input length κ (and output length at most $p(\kappa)$). These are both randomized circuits with a common random input. We call $\mathcal{C}^1$ and $\mathcal{C}^2$ the* pre-processing circuits ensembles *(or circuits for short)*
   (b) *$\mathcal{C}_\kappa$ is of size at most $s(\kappa)$, with input length at most $p(\kappa)$ and output length $\lambda(\kappa)$. We call $\mathcal{C}$ the* output circuit ensemble *(or circuit for short).*
   (c) *the combined output lengths of $\mathcal{C}^1_\kappa$ and $\mathcal{C}^2_\kappa$ equal the input length of $\mathcal{C}_\kappa$*
2. *for infinitely many $\kappa \in \mathbb{N}$:*

$$\Delta\Big(\{C_\kappa(C^1_\kappa(x), C^2_\kappa(y)) : x, y \in_R \{0,1\}^\kappa \text{ s.t. } \langle x, y\rangle = 0\}, \{C_\kappa(C^1_\kappa(x), C^2_\kappa(y)) : x, y \in_R \{0,1\}^\kappa \text{ s.t. } \langle x, y\rangle = 1\}\Big) \geq \varepsilon(\kappa)$$

randomness in both distributions is over the choice of x, y and the (shared) coins of $\mathcal{C}^1, \mathcal{C}^2$.

Assumption 2.6 ($\lambda(\cdot)$-IPPP Assumption)**.** *For any polynomial $s(\cdot)$ and inverse polynomial $\varepsilon(\cdot)$, no triplet of circuit ensembles $(\mathcal{C}, \mathcal{C}^1, \mathcal{C}^2)$ with $\mathcal{C}$ in $\mathcal{AC}^0$ can solve the $(\lambda(\cdot), s(\cdot), \varepsilon(\cdot))$-IPPP problem.*

3 Transformation Against $\mathcal{AC}^0$ Leakage

In this section we detail a secure compiler for $\mathcal{AC}^0$ leakage and give more details on the proof of Theorem 1.1. For ease of exposition we consider throughout this section a circuit $C(\cdot,\cdot)$ that is known to the adversary and takes two inputs x and y. The input x is the one chosen adaptively by the adversary in each round, whereas the input y is *secret* and protected by the compiler. In particular this gives a compiler a la Definition 2.3 by viewing C as a universal circuit and y as the description of a particular circuit to be compiler. This is similar to what is done in the secure multi-party computation literature.

The compiler transforms y into a secret state for an emulation of the universal computation. This secret state includes a *wire bundle* for each y-input wire of the circuit. The XOR of this bundle is the value of its y-input wire. The secret state also includes a "bundle bank" of bundles encoding 0 (see the overview in the introduction). Given bundles for the input wires, the transformed circuit's computation then proceeds gate by gate, using the bundles computed for that gate's input wires and the bundles in the bank to compute an output bundle. In the construction below, we present *Init* and *Update* as procedures for initializing and updating the secret state — a collection Y (viewed as a matrix whose columns are the bundles) of bundles for the y input wires, and the bundle bank G (also a matrix). We call the "bundle bank" in round t the *"generating matrix"* for that round (as it is used to generate fresh 0-bundles).

In Figure 3 we present the procedures for emulating the computation of each gate in the original circuit. Given this view of the compiler's operation (slightly modified from the one in Definition2.3), we view the *Init* and *Update* procedure's outputs as secret states that are later plugged into the gate-by-gate emulator of (public) circuit's computation. There are in Figures 1 and 2. The secret states can be transformed into a full-blown circuit a la Definition 2.3 by augmenting them with the (publicly and known to all parties) emulation of the circuit C.

Initialization $Init(1^\kappa, y)$

1. for every input wire i, corresponding to bit j of the input y, generate a new encoding: $\vec{d_i} \in \{0,1\}^\kappa$, a uniformly random vector whose XOR is y_j. Let Y be the matrix whose columns are these encodings.
2. generate a new uniformly random $\kappa \times 2\kappa$ matrix G whose columns are all encodings of 0.
3. output $state_1 \leftarrow (Y, G)$.

Figure 1: *Init* procedure, to be run in an offline stage on circuit C and secret y.

State Update $Update(1^\kappa, state_t = (Y, G))$

1. for each column Y_i of Y: $Y_i' \leftarrow Y_i + G \times \vec{r}$, where $\vec{r} \sim U_{2\kappa}$
2. pick a uniformly random $2\kappa \times 2\kappa$ matrix R. $G' \leftarrow G \times R$.
3. output $state_{t+1} \leftarrow (Y', G')$.

Figure 2: *Update* procedure, to be run under leakage between evaluations.

Security Proof: Further Details and Organization. The simulator is specified in Figure 4. We want to prove that the view it generates is indistinguishable from the real view, under bounded length $\mathcal{AC}^0$ leakage in each round. See the introduction for the high-level intuition behind the security proof. The formal argument used in the reduction is more involved. We consider the real

Gate Computations (with bundle-bank/generating matrix G)

Addition $(\vec{d}_i, \vec{d}_j)$**:**

1. $\vec{q} \leftarrow \vec{d}_i + \vec{d}_j$
2. $\vec{o} \leftarrow G \times \vec{r}$, where $\vec{r} \sim U_{2\kappa}$
3. output $\vec{d}_k \leftarrow \vec{q} + \vec{o}$

Multiplication $(\vec{d}_i, \vec{d}_j)$**:**

1. $B \leftarrow \vec{d}_i \times \vec{d}_j^T$,
   i.e. the $\kappa \times \kappa$ matrix where entry $[\ell, m]$ equals $\vec{d}_i[\ell] \cdot \vec{d}_j[m]$
2. $S \leftarrow G \times R$,
   where R is a $\{0,1\}$ uniformly random $2\kappa \times \kappa$ matrix
3. $U \leftarrow B + S$
4. for each *row* ℓ of U, compute $\vec{q}[\ell] = \oplus_{m \in [\kappa]} U[\ell, m]$
5. output $\vec{d}_k \leftarrow \vec{q}$

Constant Gate (b_i)**:**

1. $\vec{q} \leftarrow [b_i, 0, 0, \ldots, 0]$
2. $\vec{o} \leftarrow G \times \vec{r}$, where $\vec{r} \sim U_{2\kappa}$
3. output $\vec{d}_k \leftarrow \vec{q} + \vec{o}$

Duplication $(\vec{d}_i)$**:**

1. $\vec{o} \leftarrow G \times \vec{r}$, where $\vec{r} \sim U_{2\kappa}$
2. $\vec{o}' \leftarrow G \times \vec{r}'$, where $\vec{r}' \sim U_{2\kappa}$
3. output $\vec{d}_j \leftarrow \vec{d}_i + \vec{o}$ and $\vec{d}_k \leftarrow \vec{d}_i + \vec{o}'$

Output Gate $(\vec{d}_{output})$**:**

1. $\vec{o} \leftarrow G \times \vec{r}$, where $\vec{r} \sim U_{2\kappa}$
2. $\vec{d} \leftarrow \vec{d}_{output} + \vec{o}$
3. output $\oplus_{m \in [\kappa]} \vec{d}[m]$

Figure 3: Gate computations during evaluation, all run under leakage.

and simulated views, *Real* and *Simulated* (respectively). We want to use a distinguisher between the real and simulated view to build an $\mathcal{AC}^0$ circuit for solving the IPPP problem (or rather one of the variants we derive from it in Appendix B), leading to a contradiction. This requires care, and in particular we will use several hybrid views and seek methods for generating them in $\mathcal{AC}^0$ (with some pre-processing, see below).

One important difference between the *Real* and *Simulated* views is in the generating matrices they use (whose columns all encode 0 in *Real*, but are uniformly random in *Simulated*). In both views we want to use each generating matrix G to generate encodings of 0's (or 1's in *Simulated*) upon request. The immediate way of doing this is choosing randomness $\vec{r}$ from the proper distribution and computing $G \times \vec{r}$ together with all of the partial sums (the partial sums are needed for generating the entire view of each wire in the matrix-vector multiplication). Unfortunately, this is not an $\mathcal{AC}^0$ computation. See the discussion in Section 1.2 for intuition regarding this obstacle and the high-level ideas for overcoming it.

We will use several hybrid views, and set up the views (real, simulated or hybrid) as follows.

Simulator

Initialize G_0 as a uniformly random $\kappa \times 2\kappa$ matrix, and Y_0 as a uniformly random $\kappa \times n$ matrix (where n is the bit length of y).
For rounds $t \leftarrow 1, 2, \ldots$, on input x_t, generate the view for that round gate by gate:

1. For addition, multiplication, constant and duplication gates, operate exactly as the real gate computations in Figure 3 (albeit here G is uniformly random).
2. For the output gate, on encoding $\vec{d}_{output}$, retrieve the output bit $C(x_t, y)$. Generate $\vec{o}$ as a uniform linear combination of the columns of G s.t. $\oplus_{m\in[\kappa]}(\vec{d}_{output} + \vec{o})[m] = C(x_t, y)$.
3. For the state update, operate as in the real state update in Figure 2. I.e. generate Y_{t+1} by adding random linear combinations of G_t to the columns of Y_t. To generate G_{t+1}, multiply G_t by a random invertible $2\kappa \times 2\kappa$ matrix.

Figure 4: Simulator $\mathcal{S}$.

For every round t we will have a generating matrix G_t (either uniformly random, or random s.t. all columns encode 0). We generate an *external wire distribution* which specifies the encoding/bundle on each of the circuit wires together with additional information about it and global information about this round. This recalls the high-level structure of the security proof in [FRR+10], though here even the *simulated* view cannot be generated in $\mathcal{AC}^0$ and so we need a "'beefed up" external wire distribution (as discussed in Section 1.2). In particular, for each wire encoding we also include its decomposition into a linear combination of G's columns (the bundle bank) XORed with a 0 or a 1 bit. The full external wire distribution is specified in Figure 5 of Appendix C.

We argue that distinguishing the external wire distributions for any pair of adjacent views (simulated, hybrid, real) is hard given only bounded length $\mathcal{AC}^0$ leakage from each round (this usually involves another hybrid argument over the rounds and/or wires).

To finish the argument and show the complete views in their entirety are also indistinguishable we need to generate the actual views, including also the internal gate wires. As discussed in Section 1.2, we give *reconstruction procedures* for completing the view and setting values for all of the internal gate wires in both the *Real* and *Simulated* views. The reconstruction procedures are in $\mathcal{AC}^0$, and so indistinguishability of the views follows immediately from the indistinguishability of their external wire distributions.

Organization of the security proof. For lack of space, most of the security proof is in the appendices. In Appendix C we fully specify the external wire distributions and give both an intuitive overview and a full proof for the indistinguishability of the real and simulated external wire distributions (under bounded length $\mathcal{AC}^0$ leakage attacks). Indistinguishability uses the hardness of several intermediate problems (based on the IPPP Assumption), see Appendix B. The $\mathcal{AC}^0$ reconstruction procedures are in Appendix D, where we also argue that, for the *Real* and *Simulated* external wire distributions, the reconstruction procedures generate the appropriate view (respectively).

References

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[Ajt11] Miklós Ajtai. Secure computation with information leaking to an adversary. In *STOC*, pages 715–724, 2011.

[BFS86] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *FOCS*, pages 337–347, 1986.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BKKV10] 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 *FOCS*, pages 501–510, 2010.

[DI06] Bella Dubrov and Yuval Ishai. On the randomness complexity of efficient sampling. In *STOC*, pages 711–720, 2006.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[GR12] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. *Electronic Colloquium on Computational Complexity (ECCC)*, page 010, 2012.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20, 1986.

[HN10] Danny Harnik and Moni Naor. On the compressibility of *np* instances and cryptographic applications. *SIAM J. Comput.*, 39(5):1667–1713, 2010.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Wiener, editor, *CRYPTO99*, pages 388–397, 1999.

[Lok] Satyanarayana V. Lokam. Complexity lower bounds using linear algebra.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[Raz87] Alexander Razborov. Lower bounds for the size of circuits of bounded depth with basis and, xor. Math. Notes of the Academy of Science of the USSR 41, 1987.

[RCL] Boston University Reliable Computing Laboratory. Side channel attacks database. http://www.sidechannelattacks.com.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

A More on IPPP

See the introduction for an informal discussion of the IPPP problem and assumptions, and Section 2.2 for formal definitions. We provide further discussion of the problem and the assumption.

The IPPP Problem considers solving or compressing instances of the inner product problem, i.e. vectors x and y, with arbitrary (polynomial-time) pre-processing of x and y (each separately). "Solving" is in the usual sense: i.e. predicting the correct inner-product with non-negligible advantage. "Compressing" means shrinking the instance size from κ to $\lambda(\kappa)$ bits while maintaining noticeable statistical difference between the distribution of YES and NO instances.

Without pre-processing, computing the inner product of a pair of vectors is equivalent (via $\mathcal{NC}^0$ reductions) to computing parity. Dubrov and Ishai [DI06] showed $\mathcal{AC}^0$-hardness of compressing parity instances (and thus also of compressing inner product instances) to sub-linear length $\kappa^{1-\delta}$ for any $\delta > 0$. Their proof was based on Hastad's Switching Lemma [Hås86]. Their result was then used in [FRR+10] to obtain security against $\mathcal{AC}^0$ leakage of any sub-linear length.

Prior work on IPPP. Using polynomial time (and polynomial length) pre-processing that operates separately on x and on y, the complexity of *computing* (rather than compressing) inner products is a long-standing open problem in complexity theory. This fascinating problem is essentially equivalent to the 1-IPPP problem. The main difference is that prior works allow arbitrary pre-processing of polynomial output length (the pre-processing is not necessarily efficient). I.e., the 1-IPPP problem is even harder than the variant considered in that literature. Another difference (which we view as more minor) is that for IPPP we only require successful computation for infinitely many input lengths. The (debatable) "folklore belief" seems to be that the problem is hard for $\mathcal{AC}^0$ circuits even with (separate) pre-processing.

Razborov observed that rigidity of the Hadamard matrix implies lower bounds for IPPP. See e.g. Lokam [Lok] for a recent exposition of matrix rigidity and its applications to communication complexity. We note also that Babai, Frankl and Simon [BFS86] showed that a lower bound would imply separations between communication complexity classes.

Given that this variant of 1-IPPP has been a long-standing and well known problem in complexity theory, we are fairly comfortable making the 1-IPPP assumption. At the very least, breaking the 1-IPPP assumption requires resolving a long-standing open question in an unexpected direction (i.e. showing that a problem thought to be hard is actually easy). This assumption implies that our scheme is resilient to a single bit of leakage from each execution (which is already non-trivial, see the introduction). Actually, it implies resilience to logarithmic leakage in each execution (more on that below).

While we do not know of prior work on the question of using separate pre-processing to *compress* (rather than solve) parity instances, to the best of our knowledge no known techniques imply that separate pre-processing is helpful in compressing instances of the inner product function. As far as we know, the best $\mathcal{AC}^0$ algorithm is the straightforward one that uses downward self-reducability to recursively compress instances of size κ to size $\kappa/\text{polylog}\kappa$ (the exponent in the polylog expression depends on the circuit's depth). Assuming that this is indeed the best possible $\mathcal{AC}^0$ algorithm with separate pre-processing gives the $\lambda(\cdot)$-IPPP Assumption for any sub-linear λ, i.e. any $\lambda(\kappa) = \kappa^{1-\delta}$ for $\delta > 0$. This gives the same leakage bound as that achieved by [FRR+10].

Hardness vs. Leakage Tradeoff. We remark that hardness of the 1-IPPP problem for $\mathcal{AC}^0$ circuits of size $s(\cdot)$ (here we measure the size of the output circuit, the pre-processing circuits are always polynomial), implies the $\log s(\cdot)$-IPPP Assumption. See Proposition A.1 below.

Proposition A.1. *If a triplet of ensembles $(\mathcal{C}, \mathcal{C}_1, C_2)$ solves the $(\lambda, s, \varepsilon)$-IPPP problem and $\mathcal{C}$ is of constant depth, then there is a constant-depth ensemble $\mathcal{C}'$ s.t. for size bound $s'(\kappa) = O(s(\kappa)+2^{\lambda(\kappa)})$, the triplet $(\mathcal{C}', C_1, \mathcal{C}_2)$ solves the $(1, s', \varepsilon)$-IPPP problem.*

Proof Sketch. For distributions with support size $\lambda(\kappa)$, the optimal distinguisher is of size $O(2^{\lambda(\kappa)})$ and outputs a single bit. For each $\kappa \in \mathbb{N}$, the circuit $\mathcal{C}'_\kappa$ simply runs $\mathcal{C}$ and then runs the optimal distinguisher on $\mathcal{C}$'s output to distinguish YES instances from NO instances. ■

We conclude that the 1-IPPP assumption (for $\text{poly}(\kappa)$ circuits) actually implies $\log(\kappa)$-IPPP, and that our construction can resist logarithmic leakage in each execution. Moreover, assuming that 1-IPPP is hard for $\mathcal{AC}^0$ circuits of sub-exponential size (i.e. size smaller than $\exp(\text{poly}(\kappa))$), or that 1-IPPP is as hard as computing inner product without pre-processing, we get the $\lambda(\cdot)$-IPPP Assumption for any sub-polynomial λ. Under this assumption our construction resists any sub-polynomial leakage bound.

Random Self-Reducability of IPPP. The IPPP Assumption is made on random instances of the parity problem. We conclude this section by noting that this is actually equivalent (under randomized $\mathcal{NC}^0$ reductions) to worst-case hardness of IPPP using separate pre-processing. Thus, our construction is actually secure based only on the worst-case IPPP Assumption.

Proposition A.2. *If a triplet of ensembles $(\mathcal{C}, \mathcal{C}_1, C_2)$ solves the $(\lambda, s, \varepsilon)$-IPPP problem, then there exists a triplet $(\mathcal{C}', \mathcal{C}'_1, \mathcal{C}'_2)$ that are of the same depth as $(\mathcal{C}, \mathcal{C}_1, C_2)$ up to a constant additive factor, s.t. for any input length $\kappa \in \mathbb{N}$ for which the original triplet succeeds in solving the "average case" $(\lambda(\kappa), s(\kappa), \varepsilon(\kappa))$-IPPP problem (on random instances), the new triplet solves the $(\lambda(\kappa), s(\kappa) + O(\kappa), \varepsilon(\kappa)/2)$-IPPP problem* in the worst case.

By "solving IPPP in the worst case", we mean that on every YES instance, the output is a sample from some distribution D_{YES}, and on every NO instance, the output is a sample from some distribution D_{NO}, and these two distributions are at statistical distance at least $\varepsilon(\kappa)/2$.

Proof Sketch. Let $\vec{x}, \vec{y} \in \{0,1\}^\kappa$ be the instance. The new pre-processing circuits choose randomly $\vec{r}_x, \vec{r}_y \in_R \{0,1\}^\kappa$ (these are joint random choices). They then run as follows: $\mathcal{C}'_1$ runs the original $\mathcal{C}_1$ on $\vec{x}' = (\vec{x} + \vec{r}_x)$ and outputs that pre-processing information, together with the bit $b_x = \langle \vec{x}, \vec{r}_y \rangle$. $\mathcal{C}'_2$ runs the original $\mathcal{C}_2$ on $\vec{y}' = (\vec{y} + \vec{r}_y)$ and outputs that pre-processing information, together with the bit $b_y = \langle \vec{r}_x, (\vec{y} + \vec{r}_y) \rangle$. The output circuit $\mathcal{C}'$ computes $b_x \oplus b_y$ (note this is just the XOR of two bits, and can be computed in $\mathcal{NC}^0$). If this is 0, it runs the original $\mathcal{C}$ on the given pre-processing information and outputs the same answer as $\mathcal{C}$. Otherwise, it outputs 0.

We will show that, conditioned on $b_x \oplus b_y = 0$, the vectors $\vec{x}', \vec{y}'$ are uniformly random vectors with the same inner product as $\vec{x}$ and $\vec{y}$. Proposition A.2 follows since $b_x \oplus b_y = 0$ with probability $1/2$. The distribution of $\vec{x}', \vec{y}'$ is as claimed because $\vec{x}', \vec{y}'$ on their own are independently and uniformly random vectors. The additional bit $b_x \oplus b_y$ is a uniformly and independently random bit that indicates only whether the inner product of $\vec{x}', \vec{y}'$ is identical to that of $\vec{x}, \vec{y}$:

$$\begin{aligned}
\langle \vec{x}, \vec{y} \rangle \oplus \langle \vec{x}', \vec{y}' \rangle &= \langle \vec{x}, \vec{y} \rangle \oplus \langle (\vec{x} + \vec{r}_x), (\vec{y} + \vec{r}_y) \rangle \\
&= \langle \vec{x}, \vec{y} \rangle + (\langle \vec{x}, \vec{y} \rangle + \langle \vec{x}, \vec{r}_y \rangle + \langle \vec{r}_x, \vec{y} \rangle + \langle \vec{r}_x, \vec{r}_y \rangle) \\
&= \vec{0} + \langle \vec{x}, \vec{r}_y \rangle + \langle \vec{r}_x, \vec{y} \rangle + \langle \vec{r}_x, \vec{r}_y \rangle \\
&= \langle \vec{x}, \vec{r}_y \rangle + \langle \vec{r}_x, (\vec{y} + \vec{r}_y) \rangle \\
&= b_x \oplus b_y
\end{aligned}$$

■

B Hard Problems

The following intermediate problems will be useful in arguing indistinguishability of the external wire distributions of the *Real*, *Simulated*, and hybrid views. We will show that, under the IPPP Assumption, they are hard for $\mathcal{AC}^0$ output ensembles.

Problem B.1. *[$\lambda(\cdot)$-bit $U \times \vec{r}$] For security parameter κ, given a uniformly random $\kappa \times 2\kappa$ matrix U, a random vector $\vec{r} \in \{0,1\}^{2\kappa}$, and:*

1. *Polynomial time pre-processing of U and $\vec{r}$ each separately.*
2. *A vector $\vec{v} = U \times \vec{r}$ and all of the partial sums.*

Generate a $\lambda(2\kappa)$-bit vector $\vec{o}$. The statistical distance between the distributions of the output $\vec{o}$ in the cases that the parity of $\vec{v}$ is 0 or 1 should have statistical distance at least $1/\text{poly}(\kappa)$.

Problem B.2 ($\lambda(\cdot)$-bit $U \times \vec{r}, V \times \vec{s}$)**.** *For security parameter κ, given a uniformly random $\kappa \times 2\kappa$ matrix U, a uniformly random $2\kappa \times 2\kappa$ matrix R, a random vector $\vec{s} \in \{0,1\}^{2\kappa}$, a vector $\vec{r} \in \{0,1\}^{2\kappa}$ (not necessarily random), and:*

1. *Polynomial time pre-processing of (U, R) and $(\vec{r}, \vec{s})$ each separately.*
2. *The matrix $V = U \times R$ and all of the partial sums.*
3. *A vector $\vec{v} = U \times \vec{r} = V \times \vec{s}$ and all of the partial sums for both of these products.*

Generate a $\lambda(2\kappa)$-bit vector $\vec{o}$. The statistical distance between the distributions of the output $\vec{o}$ in the cases that the parity of $\vec{v}$ is 0 or 1 should have statistical distance at least $1/\text{poly}(\kappa)$.

Problem B.3 ($\lambda(\cdot)$-bit $\vec{u}, Z \times \vec{r}$)**.** *For security parameter κ, given a uniformly random $\kappa \times 2\kappa$ matrix Z, whose columns all encode 0 and uniformly random vectors $\vec{r}, \vec{u} \in \{0,1\}^{2\kappa}$, and:*

1. *Polynomial time pre-processing of $(Z, \vec{u})$ and $\vec{r}$ each separately.*
2. *A vector $\vec{v} = Z \times \vec{r}$ and all of the partial sums.*

Generate a $\lambda(2\kappa)$-bit vector $\vec{o}$. The statistical distance between the distributions of the output $\vec{o}$ in the cases that the inner product $\langle \vec{u}, \vec{r} \rangle$ is 0 or 1 should have statistical distance at least $1/\text{poly}(\kappa)$.

Problem B.4 ($\lambda(\cdot)$-bit $\vec{u}, Z \times \vec{r}, Y \times \vec{s}$)**.** *For security parameter κ, given a uniformly random $\kappa \times 2\kappa$ matrix Z whose columns all encode 0, a uniformly random $2\kappa \times 2\kappa$ matrix R, random vectors $\vec{s}, \vec{u} \in \{0,1\}^{2\kappa}$, a vector $\vec{r} \in \{0,1\}^{2\kappa}$ (not necessarily random), and:*

1. *Polynomial time pre-processing of $(Z, R, \vec{u})$ and $(\vec{r}, \vec{s})$ each separately.*
2. *The matrix $Y = Z \times R$ and all of the partial sums.*
3. *A vector $\vec{v} = Z \times \vec{r} = Y \times \vec{s}$ and all of the partial sums for both of these products.*

Generate a $\lambda(2\kappa)$-bit vector $\vec{o}$. The statistical distance between the distributions of the output $\vec{o}$ in the cases that the inner product $\langle \vec{u}, \vec{r} \rangle$ is 0 or 1 should have statistical distance at least $1/\text{poly}(\kappa)$.

These problems are all hard under Assumption 2.6.

Lemma B.5. *Under the $\lambda(\cdot)$-Assumption 2.6, Problems B.1, B.2, B.3, B.4 are all hard for $\text{poly}(\kappa)$-size $\mathcal{AC}^0$ circuits with the same $\lambda(\cdot)$ function.*

Proof Sketch. We prove here the hardness for Problem B.1, the proofs for the remaining problems are similar. Consider the IPPP problem: we are given two vectors $\vec{x}, \vec{y} \in \{0,1\}^{2\kappa}$, we are allowed to pre-process $\vec{s}$ and $\vec{y}$ separately (in polynomial time), and may then output a $\lambda(2\kappa)$-bit vector $\vec{o}$ whose distribution should be correlated with $\langle \vec{x}, \vec{y} \rangle$.

We first choose a uniformly random $\kappa \times 2\kappa$ matrix Z, whose columns are all encodings of 0 (this can be done in $\mathcal{AC}^0$). Consider the matrix U formed by adding the items of $\vec{x}$ to the main diagonal of Z, and take $\vec{r} = \vec{y}$. The parity of $\vec{v} = U \times \vec{r}$ is equal to the inner product of $\vec{x}$ and $\vec{y}$. To generate the pre-processing information for Problem B.1, we observe that since Z is independent of $\vec{x}$ and $\vec{y}$, U is a function only of $\vec{x}$ and $\vec{r}$ is a function only of $\vec{y}$. Moreover, we can use pre-processing on only $\vec{y}$ to compute all of the partial sums of $Z \times \vec{r}$. From these, we can compute in $\mathcal{AC}^0$ the partial sums of $U \times \vec{r}$ in (the addition of $\vec{x}$ to Z's main diagonal only adds a single bit to each row of Z). We have generated, using separate pre-processing on $\vec{x}$ and $\vec{y}$ and $\mathcal{AC}^0$ computations on the results, all of the pre-processing and input information for problem B.1. Thus, if we had a procedure for solving B.1 and outputting a vector $\vec{o}$ of length $\lambda(2\kappa)$ correlated with the parity of $U \times \vec{r}$, it could also be used to solve IPPP on instances of length 2κ. ■

C External Wire Distributions and Indistinguishability

The external wire distribution is in Figure below 5 below. We will reduce distinguishing the *Real* and *Simulated* external wire distributions (under bounded length $\mathcal{AC}^0$ leakage) to the IPPP Assumption.

Remark C.1. *The external wire distribution, as specified in Figure 5, includes information that might not be available to the simulator, e.g. the correct wire value a_i for each wire i. This, however, is not a concern. All we need to show is that: (i) the external wire distributions for the Real, Simulated and hybrid views, in their entirety, are indistinguishable (under bounded length $\mathcal{AC}^0$ leakage), and (ii) the Real and Simulated views can be generated in $\mathcal{AC}^0$ from their external wire distributions. The indistinguishability of the views (under $\mathcal{AC}^0$ leakage) follows, even though the simulator might not be able to generate the external wire distribution of the simulated view.*

External wire distribution

For round t with generating matrix G, the external wire distribution includes:

1. for each wire i (input, internal, output, or y-state update):
   (a) the real value $a_i \in \{0,1\}$ on wire i (on input (x_t, y)).
   (b) vectors $\vec{c}_i \in \{0,1\}^\kappa$ and $\vec{s}_i \in \{0,1\}^{2\kappa}$ such that $\vec{c}_i = G \times \vec{s}_i$, and also all the partial sums of the matrix-vector product.
   the case that i is an output wire is as above, except that the XOR of $\vec{c}_i$ is always 0.

   The encoding on wire i is decomposed as $\vec{d}_i = \vec{c}_i + (a_i \cdot \vec{e}_0)$ ($\vec{e}_0$ is the unit vector with 1 in the first coordinate and 0 elsewhere).
2. For the G-matrix update, a matrix R and the matrix-vector product $G \times R$ and all its partial sums. $G \times R$ is the generating matrix for round $t+1$.
3. For each multiplication gate, $\kappa - 1$ vectors $\vec{h}_1, \ldots \vec{h}_{\kappa-1} \in \{0,1\}^\kappa$ that are each uniformly random in the column span of G. We also include $\vec{f}_1, \ldots \vec{f}_{\kappa-1} \in \{0,1\}^{2\kappa}$, s.t. for each $j \in \{1, \ldots, \kappa-1\}$ we have $\vec{h}_j = G \times \vec{f}_j$ and all of the partial sums for this matrix-vector product.
4. For each unit vector $\vec{e}_j$ ($j \in [\kappa]$), a vector $\vec{f}_j$ such that $(\vec{e}_0 + \vec{e}_j) = G \times \vec{f}_j$ and the partial sums for this matrix-vector product.

Figure 5: External wire distribution.

Outline of the different views. To argue indistinguishability of the external wire distributions for real and simulated executions, we use two hybrid views (and additional hybrid arguments on the execution rounds and circuit gates). We consider the following external wire distributions:

- The simulated view *Simulated*, where all the generating matrices G are uniformly random, the y input wires carry dummy encodings of uniformly random values, and for every (internal) wire i we have $\vec{c}_i = G \times \vec{s}_i$ for uniformly random $\vec{s}_i$. Thus $\vec{c}_i + a_i \cdot \vec{e}_0$ encodes a uniformly random bit. The encoding $\vec{c}_i$ of the output wire *output* in each round is an encoding of 0 (thus $\vec{c}_{output} + a_{output} \cdot \vec{e}_0$ encodes the correct output bit). Note that we do not require that the $\vec{s}_i$ values for the output wires of the y-state update be uniformly random, their distribution can be arbitrary.

- The hybrid view *RealFree*, where again the generating matrices G are uniformly random, and the update of G is always done by multiplying it by a uniformly random matrix. In this view, however, the y input wires carry encodings of the real values, and for every internal wire i we have that the XOR of $\vec{c}_i$ is 0. The update of each G matrix is still done via multiplication with a uniformly random R in each round. Thus for internal and y-input wires, the $\vec{s}_i$ values are not uniformly random, but rather taken from a $(2\kappa - 1)$-dimensional subspace, and the encoding $\vec{c}_i + a_i \cdot \vec{e}_0$ on each wire i encodes the correct bit a_i.

  Intuitively, the *Simulated* and *RealFree* views are indistinguishable because given a uniformly random generating matrix G, it is hard to distinguish whether the XOR of $G \times \vec{s}$ is 0 or 1 given only bounded length $\mathcal{AC}^0$ leakage from $\vec{s}$.

- The hybrid view *RealSubspace*, where the y input wires carry encodings of their real values, but the generating matrix G is *not* uniformly random. Rather, its columns are all encodings of 0, and thus it is always the case that the XOR of c_i is 0, and $\vec{c}_i + a_i \cdot \vec{e}_0$ the correct bit

a_i. Here we still have that the $\vec{s}_i$ values are not uniformly random, but rather taken from a $(2\kappa - 1)$-dimensional subspace. The update of each G matrix is still done via multiplication with a uniformly random R in each round.

Intuitively, the *RealSubspace* and *RealFree* views are indistinguishable because given a generating matrix and encodings of 0 generated from it, it is hard to distinguish whether the columns are uniformly random or encodings of 0 using only bounded length $\mathcal{AC}^0$ leakage.

- The real view *Real*, where the y input wires carry encodings of their real values and again the columns of the generating matrix G are all encodings of 0. Here the $\vec{s}_i$ values *are* uniformly random in $\{0,1\}^{2\kappa}$ (as are the matrices R used to update G in each round). here too we do not require that the $\vec{s}_i$ values for the output wires of the y-state update be uniformly random, their distribution can be arbitrary.

  Intuitively, the *RealSubspace* and *Real* views are indistinguishable because given the random $\vec{s}_i$ values it is hard to tell whether they are taken from a full or low-dimensional subspace using only bounded-length $\mathcal{AC}^0$ leakage.

To argue indistinguishability, we use the hardness of intermediate problems that have to do with distinguishing the parity of a matrix-vector product (under preprocessing). These intermediate problems, and proofs that they are hard under the IPPP Assumption, are in Appendix B. The full proofs of Indistinguishability for the *Simulated*, *Real* and hybrid distributions follow below.

C.1 *Simulated* and *RealFree* views

In both of these views, in every round the generating matrix is uniformly random and its update is performed using a uniformly random matrix R. The views differ in the values of the wire encodings, which are uniformly random in *Simulated* but encode the correct values in *RealFree*. We reduce distinguishing the external wire distributions of these two views to Problems B.1 and B.2. We conclude that the *Simulated* and *RealFree* views are indistinguishable.

Simulated. Each of the generating matrices G is a uniformly random $\kappa \times 2\kappa$ matrix, and the matrices R used for updating the G matrix are also uniformly random $2\kappa \times 2\kappa$ matrices. The wire encodings for the output wire in each round is of the correct value (i.e. $\vec{c}_{output}$ always encodes 0). The encodings on all other wires (except the output wires for the y-state update) are uniformly random. In particular, $\vec{c}_i$ encodes a uniformly random bit. The $\vec{f}_i$ vectors for Item 3 of the external wire distribution are also uniformly random.

RealFree. Each of the generating matrices G is a uniformly random $\kappa \times 2\kappa$ matrix, and the matrices R used for updating the G matrix are also uniformly random $2\kappa \times 2\kappa$ matrices. The wire encodings are *all* of the correct values. I.e. the $\vec{c}_i$ encoding for each wire i in each round t is an encoding of 0. The $\vec{f}_i$ vectors for Item 3 of the external wire distribution are also uniformly random.

Lemma C.2. *Under the $\lambda(\cdot)$-IPPP Assumption, the external wire distributions of the Simulated and RealFree distributions are indistinguishable under $\mathcal{AC}^0$ leakage of length $\lambda(\kappa)$ in each round.*

Proof. First, note that in *Simulated* we are indeed generating the external wire distribution for the simulated distribution—the generating matrices and all encodings (except those of the output wires) are uniformly random. Recall, as mentioned in Remark C.1, that this is the case even though in the complete *Simulated* view the true a_i values are not known to the simulator.

If these two external wire distributions can be distinguished, then by a hybrid argument there exists a round t and an internal wire i (we ignore for now the case where the hybrids differ on a state update wire), such that we can examine two distinguishable hybrids H_0 and H_1. For both of these, up to round t and wire i all the c_i's encode uniformly random values, and from wire $i+1$ in round t and onwards all the c_i's encode 0. The hybrids differ in that in round t and wire i, the c_i value in H_0 is uniformly random, whereas in H_1 it is an encoding of 0. We reduce distinguishing H_0 and H_1 to Problem B.1 or B.2.

First we consider the case that wire i in round t is not a state update wire. Given a uniformly random $\kappa \times 2\kappa$ matrix U_{target} we pre-process it to set up auxiliary information for generating the external wire distribution. We use $G_t \leftarrow U_{target}$ as the generating matrix for round t. For every other round we use a uniformly random generating matrix. For every round up to t and every wire j up to i, we generate $c_j = G \times \vec{s}_j$ as a random linear combination of the rows of G (and pre-compute all the partial sums). For every wire j from (and including) wire i in round t we generate $c_j = G \times \vec{s}_j$ as a random linear combination of an encoding of 0 (and the partial sums). Given the uniformly random generating matrices we can also pre-compute the remaining information in the external wire distribution.

Now we are given a challenge $\vec{s}_{target}$ for Problem B.1, and want to distinguish whether the XOR of $\vec{s} = U_{target} \times \vec{s}_{target}$ is 0 or uniform. We will modify $\vec{s}_i$ in round t (where t is the hybrid round and i is the hybrid wire on which H_0 and H_1 differ) by adding to it $\vec{s}_{target}$. We need also to update the external wire distribution, but first observe that if the XOR of $U_{target} \times \vec{s}_{target}$ is 0, then the new $\vec{c}_i$ value will be distributed by H_1, whereas if the XOR is 1, then the new $\vec{c}_i$ value will be distributed by H_0. Finally, given $U \times \vec{s}_{tar}$ and its partial sums we can compute in $\mathcal{AC}^0$ the new $\vec{c}_i$ and $\vec{s}_i$ values, as well as the partial sums in the generation. Thus, depending on the XOR of $U_{target} \times \vec{s}_{target}$ (0 or uniform), we get the external wire distribution of H_0 or H_1 (respectively).

Finally, for the case that wire i is a state update wire for the Y-state, there is a slight difference because we need to use the $\vec{c}_i$ encoding both in rounds t and $t+1$, and so we need $\vec{s}_i$ and $\vec{s}'_i$ s.t. $\vec{c}_i = G_t \times \vec{s} = G_{t+1} \times \vec{s}'$ (and all the partial sums). In this case, we reduce distinguishing the hybrids to Problem B.2, where we use $G_t \leftarrow U$, and $G_{t+1} \leftarrow V$. Other than this difference, the reduction proceeds as above. ■

C.2 *RealFree* and *RealSubspace* Views

In both of these views, for every wire i, the vector $\vec{c}_i$ is a uniformly random encoding of 0 (and so for each wire i, its encoding is of the correct value a_i).

RealFree. In this distribution, the generating matrices are uniformly random (i.e. w.h.p. they all have rank κ), the $\vec{s}_i$ vectors are uniformly random under the constraint that the XOR of $\vec{c}_i = G \times \vec{s}_i$ is 0. Effectively, in each round t we can consider the vector $\vec{u}_t$ which is 1 in coordinate m iff the m-th column of G_t encodes 1. We choose all the $\vec{s}_i$ vectors in round t s.t. $\langle \vec{s}_i, \vec{u}_t \rangle = 0$. I.e. the $\vec{s}_i$ vectors come from a $(2\kappa - 1)$-dimensional subspace. Each matrix R used for a G state update is a uniformly random $2\kappa \times 2\kappa$ matrix. Note that the $\vec{f}_i$ vectors for item 3 of the external wire distribution are still uniformly random, and the $\vec{h}_i$ vectors are random linear combinations of G's columns.

RealSubspace. In this distribution, the generating matrices are of rank $\kappa - 1$ with columns that are all encodings of 0. The $\vec{s}_i$ vectors are also uniformly random in a $(2\kappa - 1)$-dimensional subspace (as in *RealFree*). This is guaranteed by associating with every round t a uniformly random vector $\vec{u}_t \in \{0,1\}^{2\kappa}$ (here $\vec{u}_t$ is independent of G_t). The $\vec{s}_i$ and $\vec{h}_i$ vectors all have inner product 0 with $\vec{u}_t$.

The matrix R used for the G state update is a uniformly random $2\kappa \times 2\kappa$ matrix (as in *RealFree*). Here too the $\vec{f_i}$ vectors for item 3 are uniformly random.

RealSubspaceGen. To show that the two views above are indistinguishable, we will use an intermediate view. In this *RealSubspaceGen* distribution, the behavior is similar to that of *RealSubspace*, except in the distribution of the $\vec{u}_t$ vectors and the R matrices used for the G state update (which, in particular, are no longer uniformly random). In every round t, the matrix R_t is uniformly random under the constraint that it has rank exactly $2\kappa - 1$ and each column of R_t has inner product 0 with $\vec{u}_t$ (and so R_t has rank at most $2\kappa - 1$). Taking $\vec{x}_t$ to be the (unique) non-trivial zero linear combination of R_t's columns, we choose $\vec{u}_{t+1}$ uniformly at random s.t. $\langle \vec{x}_t, \vec{u}_{t+1} \rangle = 1$ (this technical claim ensures "nice" properties of the distribution; see Claim C.4 below).

We reduce distinguishing the external wire distributions of the *RealFree* and *RealSubspace* views to Problems B.1 and B.3. This is done by arguing the indistinguishability of each of these views with *RealSubspaceGen*.

Lemma C.3. *Under the $\lambda(\cdot)$-IPPP Assumption, the external wire distributions of RealFree and RealSubspaceGen are indistinguishable under $\mathcal{AC}^0$ leakage of length $\lambda(\kappa)$ from each round.*

Proof. We first take a hybrid argument over the rounds. If *RealFree* and *RealSubspaceGen* are distinguishable, then there exists a round t such that the following hybrids H_0 and H_1 can be distinguished. In both Hybrids, the $\vec{c}_i$ vectors in every round are uniformly random encodings of 0. Up to (but not including) the state update of the matrix G in round $t-1$, both hybrids behave as *RealFree*: the generating matrix is uniformly random, the $\vec{s}_i$ values are uniform under the restriction that the XOR of $\vec{c}_i$ is 0, and the matrix R used to update G is uniformly random. In the state update of round t and in every round $q > t$, both hybrids behave as *RealSubspaceGen*: The generating matrix is uniformly random s.t. its columns encode 0's. We associate a vector $\vec{u}_q$ with each such round (for $q \geq t$), and R_q is a uniformly random $2\kappa \times 2\kappa$ matrix of of s.t. its columns have inner product 0 with $\vec{u}_q$.

The difference between the hybrids is only in the state update in round $t-1$ and the view in round t. In both hybrids, in round $t-1$ we take $\vec{u}_{t-1}$ to be the vector which is 1 in coordinate m iff the m-th column of G_{t-1} is an encoding of 1 (it is used only for the G matrix update).

- The hybrid H_0 behaves as *RealSubspaceGen*. In particular, the state update in round $t-1$ is done using a matrix R_{t-1} distributed as in *RealSubspaceGen*: i.e. a uniformly random matrix s.t. the inner product of each column of R_{t-1} with $\vec{u}_{t-1}$ is 0 (so $G_t = G_{t-1} \times R_{t-1}$ has columns encoding 0). The view in round t is the generated exactly as in *RealSubspaceGen*.
- The hybrid H_1 behaves as *RealFree*, i.e. the state update in round $t-1$ is with a uniformly random matrix R_{t-1}, so the generating matrix G_t in round t is also uniformly random. We take $\vec{u}_t$ to be the vector which is 1 in coordinate m iff the m-th column of G_t encodes 1. The view in round t is then generated accordingly. The state update in round t is performed using a random matrix R_t distributed as in *RealSubspaceGen*: a uniformly random matrix under the constraint that each of its columns has inner product 0 with $\vec{u}_t$ (so $G_t \times R_t$ has columns encoding 0).

Reduction Outline. We reduce distinguishing H_0 and H_1 to Problem B.1. Given a uniformly random $\kappa \times 2\kappa$ matrix U_{target}, a vector $\vec{w}_{target}$ and the partial sums for $\vec{v} = U_{target} \times \vec{w}_{target}$, we want to create a view s.t. if the parity of $\vec{v}$ is 0 then we get H_0, and if the parity is 1 we get

H_1. Distinguishing the views with bounded $\mathcal{AC}^0$ leakage from each round then allows us to solve Problem B.1.

The idea is to pre-process U_{target} to generate what is essentially the view H_0 with $G_{t-1} = U_{target}$. We then get $\vec{v}, \vec{w}_{target}$ and update the view by adding $\vec{w}_{target}$ to some of the columns of R_{t-1}. This has the effect of adding $\vec{v}$ to the same columns in G_t. If $\vec{v}$ is an encoding of 0, then G_t changes, but its columns remain encodings of 0 (as in H_0). If $\vec{v}$ is an encoding of 1, then some of G_t's columns become encodings of 1, and in fact G_t is a uniformly random matrix as in H_1 (see below). This modification to the external wire distribution of R_{t-1}, G_t and the partial sums of $G_{t-1} \times R_{t-1}$ can be done in $\mathcal{AC}^0$. The challenge now, however, is that G_t has changed. We need to recompute *in* $\mathcal{AC}^0$ the external wire distribution for round t (including all the partial sums needed). Moreover, we want the distribution for the later rounds (and in particular G_{t+1}) to remain unchanged, so that the leakage from all rounds but $t-1, t$ is independent of $\vec{w}_{target}$. This requires careful arguments, and in particular this is where we make (extensive) use of the $\vec{u}_t, \vec{u}_{t+1}$ vectors. In what follows we give a full specification of the reduction, including the pre-processing of U_{target}, the $\mathcal{AC}^0$ update using $\vec{w}_{target}$, and the argument that the distribution generated is H_0 or H_1 (depending on the xor of $\vec{v}$).

Auxiliary information. We use $G_{t-1} \leftarrow U_{target}$ as the generating matrix for round $t-1$ (where the hybrid round is t, we ignore here the easier case where the hybrid round is the first round), and generate the view H_0. More specifically:

1. For every round before $t-1$ we set the $\kappa \times 2\kappa$ generating matrix to be uniformly random. For all the rounds up to (and including) $t-1$, we set up and pre-compute all of the wire encodings and additional information needed for the external wire distribution according to *RealFree*.

2. We take $\vec{u}_{t-1}$ to be the vector which is 1 in coordinate m iff the m-th column of G_{t-1} encodes 1. For the state update at round $t-1$ we choose a uniformly random $2\kappa \times 2\kappa$ matrix R_{t-1} such that the inner product of each of its columns with $\vec{u}_{t-1}$ is 0. This means that $Z_t = U_{t-1} \times R_{t-1}$ has columns that are all encodings of 0.

3. For every round q starting with t, $\vec{u}_q \in \{0,1\}^{2\kappa}$ is set to be uniformly random. The matrix R_q for the state update is uniformly random under the restriction that for each column of R_q its inner product with $\vec{u}_q$ is 0. The choice of R_q sets $Z_{q+1} = Z_q \times R_q$, whose columns are (by construction) all encodings of 0.

   For the rounds starting at t we treat (for the moment) these Z_q's as their generating matrices. We then pre-compute the external wire view for *RealSubspaceGen*. This includes the $\vec{c}_i$ and $\vec{s}_i$ values for each round starting at (and including) t, and the partial sums for $G_q \times \vec{s}_i$. We pick the $\vec{s}_i$ values in round q such that $\langle \vec{s}_i, \vec{u}_q \rangle = 0$ and for round t the partial sums in $\langle \vec{s}_i, \vec{u}_t \rangle$ are all known.

4. It remains to generate, for each round, the additional information for items 3 and 4 of the external wire distribution. For rounds 1 through $t-1$ this is done in the natural way. We specify here how this is done for round $q \geq t$ (and why it is possible). For each multiplication gate we generate $\vec{h}$ vectors such that $\vec{h}_i = Z_q \times \vec{f}_i$, where $\vec{f}_i$ is uniformly random and all of the partial sums are known. For each unit vector $\vec{e}_j$ we generate $\vec{f}_j$ such that $(\vec{e}_0 + \vec{e}_j) = Z_q \times \vec{f}_j$, where $\langle \vec{u}_q, \vec{f}_j \rangle = 0$ and the partial sums are known. Claim C.4 below shows that we can do

this, because w.h.p.: (i) Z_q is a rank $\kappa - 1$ matrix whose columns are all encodings of 0 (i.e. they span the subspace of vectors whose XOR is 0), and for every $j \in [\kappa]$ it holds that the XOR of $(\vec{e}_0 + \vec{e}_j)$ is 0, and (ii) for each j we can find $\vec{f}_j$ as above such that $\langle \vec{u}_q, \vec{f}_j \rangle = 0$. In particular, Claim C.4 shows that for any vector $\vec{z}$ encoding 0, there exists $\vec{s}$ s.t. $\vec{z} = Z_q \times \vec{s}$, and the inner product of $\vec{s}$ and $\vec{u}_q$ is 0.

Claim C.4. *With all but $T \cdot 2^{-\Omega(\kappa)}$ probability, in every round $q \geq t$, the matrix Z_q is of rank $\kappa - 1$. For any vector $\vec{s} \in \{0,1\}^{2\kappa}$ there exists $\vec{s}' \in \{0,1\}^{2\kappa}$ s.t. $Z_q \times \vec{s} = Z_q \times \vec{s}'$ and $\langle \vec{u}_q, \vec{s} - \vec{s}' \rangle = 1$.*

Proof. For $q = t$ we know that G_{t-1} is a uniformly random $\kappa \times 2\kappa$ matrix, and so with all but $2^{\Omega(-\kappa)}$ probability it has rank κ. R_{t-1} is a uniformly random $2\kappa \times 2\kappa$ matrix s.t. all its columns have inner product 0 with $\vec{u}_{t-1}$. This implies that with all but $2^{\Omega(-\kappa)}$ probability, Z_t has rank $\kappa - 1$ (its columns span the space of encodings of 0). Moreover, $\vec{u}_t$ is uniformly random and independent of Z_t, and so with all but $2^{\Omega(-\kappa)}$ probability there is a vector $\vec{x}_t$ in the kernel of Z_t (this kernel is of rank $\kappa + 1$), s.t. $\vec{x}_t$ has inner product 1 with $\vec{u}_t$. The proof for later rounds follows along similar lines. Taking a union bound over all T rounds, with all but $T \cdot 2^{-\Omega(\kappa)}$ probability the claim holds. ■

For rounds $t + 1$ and higher, we actually use the Z's as their generating matrix. If we used $G_t \leftarrow Z_t$, then the above pre-computation essentially yields the view H_0. We also have (in addition to the information in the external wire distribution) the partial sums of $\langle \vec{s}_i, \vec{u}_t \rangle$ for each of the $\vec{s}_i$'s in round t and the partial sums for $G_{t-1} \times R_{t-1}$ and for $G_t \times R_t$. All of the above information can be computed in advance, given only $U_{target} = G_{t-1}$, in randomized polynomial time.

Incorporating the challenge. Now, given a challenge $\vec{w}_{target}$ for Problem B.1, we want to modify the view that we generated so that if $\vec{v} = U_{target} \times \vec{w}_{target}$ is a (random) encoding of 0, then we get the distribution H_0, whereas if it is an encoding of 1, then we get the distribution H_1. Furthermore, the modification should be computable in $\mathcal{AC}^0$ given the above auxiliary information (and all of the partial sums for the $U_{target} \times \vec{w}_{target}$ matrix-vector product).

We modify the view generated so far by adding $\vec{w}_{target}$ to some of R_{t-1}'s columns. This gives us a new R'_{t-1} and a new $G_t = G_{t-1} \times R'_{t-1}$. Specifically, we will add $\vec{w}_{target}$ to each column m of R_{t-1} for which $\vec{u}_t[m] = 1$. Examining this new G_t, we will show how (in $\mathcal{AC}^0$) to use the auxiliary information to generate an updated external wire view for rounds $t-1, t$ and the earlier and later rounds. We observe that: $G_t = Z_t + \vec{v} \times \vec{u}_t^T$, i.e. the above modification has the effect of adding $\vec{v} = U_{target} \times \vec{w}_{target}$ to each column m of Z_t where $\vec{u}_t[m] = 1$. If $\vec{v}$ is a (uniformly random) encoding of 1, then G_t has columns encoding 1, and in fact we will show that R_{t-1}, G_t are each uniformly random matrices and the updated view is H_1. If $\vec{v}$ is a (uniformly random) encoding of 0, then the updated view is exactly H_0. We proceed to argue this formally:

1. First we examine the new matrix $R'_{t-1} \leftarrow R_{t-1} + \vec{w}_{target} \times \vec{u}_t^T$. Observe that given the partial sums for $G_{t-1} \times R_{t-1}$ and $G_{t-1} \times \vec{w}_{target}$ we can (in $\mathcal{AC}^0$) compute the partial sums for the matrix product $G_{t-1} \times R'_{t-1}$ (these are needed for the external wire distribution).

   If $\vec{v}$ is an encoding of 1, then R'_{t-1} is a uniformly random matrix, as in H_1. This is because $\vec{u}_t$ is uniformly random, and R_{t-1} is a uniformly random matrix whose m-th column has parity $\vec{u}_t[m]$.

   If $\vec{v}$ is an encoding of 0, then R'_{t-1} is a uniformly random matrix whose columns have inner product 0 with $\vec{u}_{t-1}$, as in H_0. This is because R_{t-1} was such a matrix, and adding a

fixe column that has inner product 0 with $\vec{u}_{t-1}$ to some of its columns does not change its distribution.

2. Now examine the new generating matrix for round t:

$$G_t = G_{t-1} \times R'_{t-1} = Z_t + \vec{v} \times \vec{u}_t^T$$

If the challenge $\vec{v}$ is an encoding of 1, then R_{t-1} is a uniformly random matrix, and so G_t is a distributed exactly as in H_1. If $\vec{v}$ is an encoding of 0, then we argued that R'_{t-1} is distributed as in H_0, and thus so is G_t.

3. In round t, for each wire i and vector $\vec{c}_i$, where we have $\vec{c}_i = Z_t \times \vec{s}_i$ and $\langle \vec{u}_t, \vec{s}_i \rangle = 0$, we leave the $\vec{c}_i$ and $\vec{s}_i$ values unchanged. It still holds that:

$$G_t \times \vec{s}_i = (Z_t + \vec{v} \times \vec{u}_t^T) \times \vec{s}_i = Z_t \times \vec{s}_i = \vec{c}_i$$

i.e. these value remains unchanged and identically distributed. It remains to update the partial sums of the matrix-vector products $G_t \times s_i$ as per the update to G_t. This is done in $\mathcal{AC}^0$ using the partial sums of $Z_t \times \vec{s}_i$ and of $\langle \vec{u}_t, \vec{s}_i \rangle$ (which we pre-computed). We similarly update the partial sums the vectors in item 4 of the external wire distribution. All of these are identically distributed in H_0 and H_1 and this is maintained by the update. Finally, we update the $\vec{f}_i$ and $\vec{h}_i$ values for item 3 of the external wire distribution and the partial sums for $G_t \times \vec{f}_i$. The $\vec{f}_i$ vectors were (and remain) uniformly random (and are distributed identically in H_0 and H_1), the $\vec{h}_i$ vectors are just random linear combinations of the columns of G_t.

The product $G_t \times R_t$ is unchanged (R_t is unchanged, the inner product of its columns with $\vec{u}_t$ are all 0), but we update the partial sums. R_t is distributed as it is in both H_0 and H_1 (uniformly random under the constraints that all its columns have inner product 0 with $\vec{u}_t$).

4. Finally, there is no change to G_{t+1} or any of the auxiliary information beyond round t. Regardless of the value of the challenge $\vec{v}$, the external wire distribution in these later round remains identical to that in H_0. The same is true for all rounds before round $t-1$—they are unchanged and distributed as in H_1.

Conclusion. We have generated the external wire distribution for H_0 or H_1 depending on the xor of the challenge $\vec{v}$ (1 or 0 respectively). Furthermore, the view in every round but $t-1$ and t is independent of $\vec{v}$ and $\vec{w}_{target}$ (depends only on U_{target}). Given the auxiliary information we set up in advance, the leakage in rounds $t-1$ and t are an $\mathcal{AC}^0$ function of U_{target}, $\vec{w}_{target}$, and the partial sums in $U_{target} \times \vec{w}_{target}$. If the leakage in each round is of bounded length $\kappa^{1-\delta}$ for $\delta > 0$, then an adversary who distinguishes H_0 and H_1 can solve Problem B.1, a contradiction. ■

Lemma C.5. *Under Assumption 2.6, for any $\delta > 0$, the RealSubspaceGen and RealSubspace distributions are indistinguishable under $\mathcal{AC}^0$ leakage of length $\kappa^{1-\delta}$ and size* $\text{poly}(\kappa)$ *from each round.*

Proof Sketch. The proof is very similar to that of Lemma C.3. Again we use a hybrid argument to get to two distinguishable views that differ only in rounds $t-1$ and t. In all rounds the generating matrices are uniformly random with columns encoding 0. The only difference is whether the R matrices used to go from one generating matrix to the next are uniformly random (as in *RealSubspace*) or of rank at most $\kappa - 1$ (as in *RealSubspaceGen*).

The main difference from the proof of Lemma C.3 is that here we first reduce to the Problem B.3. The reduction follows the one in the proof of Lemma C.3. ■

C.3 *RealSubspace* and *Real* Views

In both of these views, in every round the generating matrix has columns encoding 0 and its update is performed using a uniformly random matrix R. The views differ in the values of the wire encodings, which are uniformly random encodings of 0 in *Real*, but drawn from a $(2\kappa - 1)$-dimensional subspace in *RealSubspace*. We reduce distinguishing the external wire distributions of these two views to Problems B.3 and B.4. We conclude that the *Real* and *RealSubspace* views are indistinguishable.

Lemma C.6. *Under the $\lambda(\cdot)$-IPPP Assumption, the external wire distributions of the Real and RealSubspace distributions are indistinguishable under $\mathcal{AC}^0$ leakage of length $\lambda(\kappa)$ in each round.*

Proof Sketch. The proof is exactly as in the proof of Lemma C.2, except that here the G matrices all have columns encoding 0, and so we reduce to the hardness of Problems B.3 and B.4 (rather than B.1 and B.2). ■

D Reconstruction Procedures

Recall the external wire distribution, as specified in Figure 5. We specify here the reconstruction procedure for addition and multiplication (the most involved case). See the full version for all other gate types. The two important properties of the reconstruction are (i) it is computable in $\mathcal{AC}^0$, and (ii) if the generating matrix G, the encodings on the external wires are distributed as in the *Real* or *Simulated* distributions, and the $\vec{s}_i$ values for all internal wires are uniformly random, then the view generated by the reconstruction procedure is *identical* to the *Real* or *Simulated* distribution (respectively). (we don't restrict the distribution of $\vec{s}_i$ for the y-state update output wires).[4]

Recall that in the external wire distribution, for both an addition or a multiplication gate with input wires i and j and output wire k and for generating matrix G, we are given: $\vec{c}_i, \vec{c}_j, \vec{c}_k \in \{0,1\}^\kappa$, $\vec{s}_i, \vec{s}_j, \vec{s}_k \in \{0,1\}^{2\kappa}$ and $a_i, a_j, a_k \in \{0,1\}$. For wire $\ell \in \{i, j, k\}$ the encoding on that wire is $\vec{c}_\ell + a_\ell \cdot \vec{e}_0$ and it holds that $\vec{c}_\ell = G \times \vec{s}_\ell$. We are also given all the partial sums for this matrix-vector product, and it holds that for this gate, given inputs a_i, a_j, the correct output is a_k. We generate view on the internal gate wires as follows:

Addition gates. We generate $\vec{q} \leftarrow \vec{c}_i + \vec{c}_j$ and $\vec{o} \leftarrow \vec{c}_k - \vec{q}$. We use the auxiliary information in the external wire distribution to generate $\vec{r} \leftarrow \vec{s}_k - (\vec{s}_i + \vec{s}_j)$ such that $\vec{q} = G \times \vec{r}$ and we know all of the partial sums for the matrix-vector product. Given the auxiliary information, all the computations are in $\mathcal{AC}^0$.

Claim D.1. *If the $\vec{s}_i$ values for all wires (except the output wires for the y-state update) are uniformly random, and if the G matrices and external wire encodings are distributed according to the Real or Simulated external wire distributions, then the view generated by the reconstruction*

[4] Note that in [FRR+10] they only needed to consider a reconstruction procedure for the case of the *real* distribution (because the simulation worked in $\mathcal{AC}^0$), and the view generated by the reconstruction (specifically, for the case of multiplication) was indistinguishable (in $\mathcal{AC}^0$) but not identical to the real view.

procedure on each addition gate is identical *to the Real or Simulated distribution (respectively) and independent of the reconstructed distribution for all other gates.*

Proof Sketch for Claim D.1. When the G matrices are uniformly random, the encodings on every external wire are distributed exactly as in the *Simulated* distribution. When the G matrices are uniformly random with columns which are all encodings of 0, then the encodings on every wire are distributed exactly as in the *Real* distribution.

Fixing $\vec{c}_i, \vec{c}_j, \vec{c}_k$ and a_i, a_j, a_k, the view generated (deterministically) for the internal wires of the addition gate is *exactly* identical to that generated by the *Real* or *Simulated* views given that the encodings on the external wires i, j, k are exactly $(\vec{c}_i + a_i \cdot \vec{e}_0), (\vec{c}_j + a_j \cdot \vec{e}_0), (\vec{c}_k + a_j \cdot \vec{e}_0)$ (respectively). ■

Multiplication gates. Multiplication gates are slightly more involved. We proceed as follows:

1. Compute B as in the "real" gate, i.e. $B \leftarrow \vec{c}_i \times \vec{c}_j^T$. Examining B more closely, for column $m \in [\kappa]$ we take $\alpha_m, \beta_m \in \{0,1\}$ such that the following holds:

$$\begin{aligned} B[m] &= (\vec{c}_i + a_i \cdot \vec{e}_0) \times ((\vec{c}_j + a_j \cdot \vec{e}_0)[m]) \\ &= ((\vec{c}_j + a_j \cdot \vec{e}_0)[m]) \cdot \vec{c}_i + (a_i \cdot ((\vec{c}_j + a_j \cdot \vec{e}_0)[m])) \cdot \vec{e}_0 \\ &= \alpha_m \cdot \vec{c}_i + \beta_m \cdot \vec{e}_0 \end{aligned}$$

   And we can compute $B[m], \alpha_m, \beta_m$ in $\mathcal{AC}^0$. Moreover, recall that the auxiliary information includes $\vec{z}$ s.t. $G \times \vec{z} = \vec{0}$ and all of the partial sums are known. This means that for each column m of B we can derive (in $\mathcal{AC}^0$) $\vec{s}'_m$ such that $\alpha_m \cdot \vec{c}_i = G \times \vec{s}'_m$ and all the partial sums are known (this is done by taking $\vec{s}'_m = \alpha_m \cdot \vec{s}_i + (1 - \alpha_m) \cdot \vec{z}$).

2. We now compute (in $\mathcal{AC}^0$) a random $\kappa \times \kappa$ matrix U such that its m-th column differs from the m-th column of B by a random linear combination of the columns of G. Moreover, (i) the sum of all of U's columns is $\vec{c}_k + a_k \cdot \vec{e}_0$ and all of the partial sums (of this column sum) are known and (ii) for $m \in [\kappa]$, the m-th column of U is $U[m] = G \times \vec{r}'_m + \beta_m \cdot \vec{e}_0$, where $\vec{r}'_m$ and all of the partial sums of $G \times \vec{r}'_m$ are known. We now specify how this computation is performed.

   We use the $\kappa - 1$ pre-computed vectors $\vec{h}_1, \ldots, \vec{h}_{\kappa-1} \in \{0,1\}^\kappa$ from the auxiliary input, where for each of them we have $\vec{h}_m = G \times \vec{f}_m$ and all of the partial sums are known (see Figure 5). Take:

$$U = \begin{pmatrix} \vec{c}_k + a_i \cdot \vec{c}_j + \vec{h}_1, & \vec{h}_1 + \vec{h}_2, & \vec{h}_2 + \vec{h}_3, & \ldots & \vec{h}_{\kappa-1} \end{pmatrix} + \begin{pmatrix} \beta_0 & 0 & \ldots & 0 \\ 0 & \beta_1 & \ldots & 0 \\ & & \ldots & \ldots \\ 0 & \ldots & 0 & \beta_{\kappa-1} \end{pmatrix}$$

   It follows that the sum of all columns of U is simply the vector

$$(\vec{c}_k + a_i \cdot \vec{c}_j) + a_i \cdot (\vec{c}_j + a_j \cdot \vec{e}_0)) = \vec{c}_k + a_i \cdot a_j \cdot \vec{e}_0 == \vec{c}_k + a_k \cdot \vec{e}_0$$

   and moreover all of the partial sums can be computed in $\mathcal{AC}^0$, so requirement (i) above is satisfied. Moreover, note that U is random under the restriction on its columns (differing from

B's by random linear combinations of G's) and their sum. It remains to show requirement (ii) and compute for each column m the vectors $\vec{r}'_m$ s.t.

$$U[m] = G \times \vec{r}'_m + \beta_m \cdot \vec{e}_0$$

(and the partial sums). Examining the decomposition of U into the two matrices above, we can write:

$$U[m] = (G \times \vec{w}_m) + \beta_m \cdot \vec{e}_m$$

(where $\vec{e}_m$ is the m-th unit vector). Here $\vec{w}_m$ and the partial sums for $G \times \vec{w}_m$ are computable in $\mathcal{AC}^0$ from the auxiliary information. It remains to compute $\vec{w}'_m$ such that:

$$G \times \vec{w}'_m = G \times (\vec{r}'_m - \vec{w}_m) = \beta_m \cdot (\vec{e}_0 + \vec{e}_m)$$

(and the partial sums). This is done in $\mathcal{AC}^0$ using the auxiliary information, which includes all of the possible $\vec{w}'_m$ vectors (and their partial sums)—there are only $\kappa + 1$ possibilities (vectors of the form $\beta_m \cdot (\vec{e}_0 + \vec{e}_m)$ for $\beta_m \in \{0, 1\}$ and $m \in \kappa$) . Note that these $\kappa + 1$ possible vectors are completely independent of all the wire encodings. We then take $\vec{r}'_m = \vec{w}_m + \vec{w}'_m$ and the desired properties follow.

3. Given U and B and the sum and partial sums of the columns of U, all we need to complete the view of the internal wires is the generation of the matrix S such that $U = B + S$. For the m-th column of S we take

$$S[m] = U[m] - B[m] = G \times (\vec{r}'_m - \alpha_m \cdot \vec{s}'_m)$$

where the partial sums for the generation of each column can now be computed in $\mathcal{AC}^0$.

Claim D.2. *If the $\vec{s}_i$ values for all wires (except the output wires for the y-state update) are uniformly random, and the $\vec{f}_i$ values in Item 3 of the external wire distribution are all uniformly random, and if the G matrices and external wire encodings are distributed according to the Real or Simulated external wire distributions, then the view generated by the reconstruction procedure on each multiplication gate is* identical *to the Real or Simulated distribution (respectively) and independent of the reconstructed distribution for all other gates.*

Proof Sketch. As in the proof of Claim D.1, when the G matrices are uniformly random, the encodings on every wire are distributed exactly as in the *Simulated* distribution. When the G matrices are uniformly random with columns which are all encodings of 0, then the encodings on every wire are distributed exactly as in the *Real* distribution.

Here, unlike the case of addition, the view for the internal wires is generated probabilistically. Fixing $\vec{c}_i, \vec{c}_j, \vec{c}_k$ and a_i, a_j, a_k, the distribution of the matrix S is *exactly* as in the *Real* or *Simulated* views. The columns of S are uniformly random linear combinations of G under the constraint (in the *Simulated* distribution) that the encodings on the external wires are as given. If the external wire encodings are distributed identically to *Real* or *Simulated*, then the internal wire distribution will also be distributed by the appropriate view, *Real* or *Simulated* (respectively). ■