PicoGRAM: Practical Garbled RAM from Decisional Diffie-Hellman

Tianyao Gu[1,2], Afonso Tinoco[1,2,3], Sri Harish G Rajan[1], and Elaine Shi[1,2]

[1]Carnegie Mellon University
[2]Oblivious Labs, Inc.
[3]Instituto Superior Técnico

August 14, 2025

Abstract

Making 2-party computation scale up to big datasets is a long-cherished dream of our community. More than a decade ago, a line of work has implemented and optimized interactive RAM-model 2-party computation (2PC), achieving somewhat reasonable concrete performance on large datasets, but unfortunately suffering from $\widetilde{O}(T)$ roundtrips for a T-time computation. Garbled RAM promises to compress the number of roundtrips to 2, and encouragingly, a line of recent work has designed concretely efficient Garbled RAM schemes whose asymptotic communication and computation costs almost match the best known interactive RAM-model 2PC, but still leaves $(\mathsf{poly})\log\log$ gaps.

We present PicoGRAM, a practical garbled RAM (GRAM) scheme that not only asymptotically matches the prior best RAM-model 2PC, but also achieves an *order of magnitude* concrete improvement in *online* time relative to interactive RAM-model 2PC, on a dataset of size 8GB. Moreover, our work also gives the first Garbled RAM whose total cost (including bandwidth and computation) achieves an optimal dependency on the database size (up to an arbitrarily small super-constant factor).

Our work shows that for high-value real-life applications such as Signal, blockchains, and Meta that require oblivious accesses to large datasets, Garbled RAM is a promising direction towards eventually removing the trusted hardware assumption that exists in production implementations today. Our open source code is available at https://github.com/picogramimpl/picogram.

Contents

1 Introduction

Garbled circuits, first introduced by Yao [Yao86], are a fundamental cryptographic tool for constant-round secure two-party computation (2PC). Despite decades of optimizations that have made garbled circuits practical [BMR90, NPS99, KS08, ZRE15, RR21, BHKR13, HEKM11, SHS+15], most works focus on Boolean circuit computation and fail to efficiently capture the Random Access Machine (RAM) model, which is ubiquitous in real-world computation. A RAM with space N features a constant number of registers and a memory of N cells, where each register or memory cell can store a W-bit word. In every time step, the RAM performs a CPU instruction over the registers' values, as well as a memory read and write operation.

While CPU instructions can be garbled efficiently as Boolean circuits, garbling memory accesses presents greater challenges. The naïve approach is to convert each memory read and write into a circuit that linearly scans through memory, resulting in prohibitive overhead as the data size grows.

There are two main approaches to avoid this per-instruction linear scan overhead. First, a line of work has shown how to achieve (interactive) RAM-model 2PC [GKK+12, GGH+13b, WHC+14, ZWR+16, DS17, WCS15, LHS+14, LWN+15]. State-of-the-art constructions [WCS15, LWN+15, Kel20] in this literature typically use Circuit ORAM [WCS15] to convert the RAM to a sequence of interactive circuits, and then use a 2PC protocol (e.g., garbled circuits) to evaluate them. When the word size W is at least $\Omega(\log^2 N)$, the resulting 2PC protocol achieves a per-instruction communication cost of $O(\lambda \cdot W \cdot \log N \cdot \omega(1))$, where $\omega(1)$ is an arbitrarily small super-constant function in the RAM's space N, but suffers from $\widetilde{O}(T)$ rounds, where T is the running time of the original RAM [WCS15]. The large round complexity makes this approach unsuitable for deployments with high end-to-end latency (e.g., over a wide-area network), especially when T is large.

The second approach is to use an efficient garbled RAM (GRAM) [LO13] scheme. Like garbled circuits, a garbled RAM requires only two rounds when used in a semi-honest 2PC protocol. However, unlike the naïve approach of converting each memory access into a linear scan, garbled RAM significantly reduces the per-instruction cost to polylogarithmic in N. A key challenge is designing a concretely efficient garbled RAM scheme. Unfortunately, earlier works in this literature [LO13, GHL+14, GLO15, GLOS15, LO17] focused only on theoretical feasibility, without explicitly quantifying the poly factor in the poly log overhead. While recent works [HKO22, PLS23, HKO23] have made encouraging progress in minimizing the poly log factor, their communication and computation costs are still a (poly) log log factor away from the prior best interactive RAM-model 2PC [WCS15, LWN+15].

1.1 Our Theoretical Contribution

We propose a new garbled RAM construction called PicoGRAM. We reduce the extra poly log log N factor in previous constructions [PLS23, HKO23] to an arbitrarily small super-constant factor, leveraging the Decisional Diffie-Hellman (DDH) assumption. For word size $W = \Omega(\log^2 N)$, PicoGRAM achieves a per-instruction communication cost of $O(\lambda \cdot W \cdot \log N \cdot \omega(1))$, where we abuse the $\omega(1)$ notation to mean an arbitrarily small super-constant function in N. In this sense, PicoGRAM matches the communication cost of the best known RAM-model 2PC scheme [WCS15][1], but we reduce the round complexity from $\widetilde{O}(T)$ to only two rounds.

[1]Since we care about concrete efficiency, we compare only with concretely efficient constructions. The line of work on succinct garbled RAM [LP14, BGL+15, CHJV15, KLW15, CCHR16, CH16, CCC+16, ACC+16, AL18] is asymptotically better but completely impractical due to their reliance on indistinguishability obfuscation [JLS20, GGH+13a]. See Section 1.4 for more discussion.

Table 1: Comparison of communication cost with prior works for semi-honest 2PC, where N is the RAM's space, T is the RAM's runtime, and W is the word width. OWF stands for one-way functions and CCRH stands for circular correlation robust hash. Interactive represents the RAM-model 2PC construction in the work of Circuit ORAM [WCS15].

	Comm. in bits per instruction	**Rounds**	**Assumption**
Interactive [WCS15]	$O\left(\lambda \cdot (W \cdot \log N \cdot \omega(1) + \log^3 N)\right)$	$O(T \cdot \log N)$	OWF
EpiGRAM [HKO22]	$O\left(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot \log N\right)$	2	CCRH
NanoGRAM [PLS23]	$O\left(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot (\log \log N)^2\right)$	2	CCRH
Tri-State [HKO23]*	$O\left(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot \log \log N\right)$	2	OWF
PicoGRAM	$O\left(\lambda \cdot (W \cdot \log N \cdot \omega(1) + \log^3 N)\right)$	2	DDH

* While tri-state GRAM [HKO23, Hea24] presents their results assuming $N = T$, we show in Section A.1 that this assumption can be removed.

A more generalized version of our asymptotic result is stated in the following theorem, where communication is measured in bits, and computation is measured in terms of group multiplications.

Theorem 1.1. *Assume the Decisional Diffie-Hellman (DDH) assumption. There exists a garbled RAM with an amortized communication and computation cost of $O(\lambda \cdot (W \cdot \log N \cdot \omega(1) + \log^3 N))$ per instruction, where N is the RAM's space (counted in the number of words), W is the word width, λ is the security parameter, and $\omega(1)$ is an arbitrarily small super-constant factor in N.*

For both the RAM-model 2PC literature [WCS15, LWN+15, Kel20] and PicoGRAM, the extra $\omega(1)$ factor stems from the underlying ORAM construction [WCS15], and it is an open question whether we can eliminate it while still preserving concrete efficiency. Even the more restricted problem of devising a concretely efficient Oblivious RAM scheme (suitable for garbled RAM or 2PC) without the extra $\omega(1)$ factor remains open.

We give a more detailed comparison with prior work in Table 1. Note that in PicoGRAM, the security parameter λ denotes the number of bits used to represent a group element, whereas in other schemes λ denotes the number of bits of a one-way function or a circular correlation robust hash (CCRH) [KS08, CKKZ12].

1.2 Open-Source Implementation and Concrete Performance

Open-source implementation. Although Theorem 1.1 can be based solely on the DDH assumption, for our practical implementation, we additionally adopt FreeXOR-style optimizations [KS08, RR21] for better concrete performance. Therefore, our practical variant of PicoGRAM additionally relies on the existence of a random oracle (RO) for these concrete optimizations to work. We implemented this practical variant in C++ and open-sourced the code at `https://github.com/picogramimpl/picogram`.

For our current implementation, we store all the garbled circuitry in memory. For this reason, in our evaluation results later, all results for up to $N = 2^{16}$ are from actual measurements, whereas results for larger N are extrapolated. Note that reported communication costs are exact regardless of N, since communication costs can be calculated in "count mode" (i.e., a simulator that does not actually execute the cryptography). With some extra engineering work, it is not too hard to extend our current implementation with a better memory management scheme. Using appropriate pipelining, we expect that the memory management should not incur noticeable slowdown, so our extrapolated numbers for $N > 2^{16}$ should be quite accurate.

Table 2: Comparison of concrete costs with prior works for semi-honest 2PC, assuming a word width of 64 bits, network bandwidth of 300 Mbps, and a round trip time of 100 ms. N is the RAM's space. Every column represents a different variant of each scheme optimized for the corresponding metric. The results for $N = 2^{30}$ are extrapolated.

Scheme	N	**Communication**		**End-to-end Time**		**Online Time**	
		MB	Relative to Interactive	ms	Relative to Interactive	ms	Relative to Interactive
Interactive [WCS15]	2^{16}	**2.94**	**1**×	378	1×	102	1×
	2^{30}	**13.1**	**1**×	1365	1×	455	1×
Prior-best GRAM [PLS23]	2^{16}	16.4	5.58×	490	1.30×	30.9	0.30×
	2^{30}	120	9.16×	3565	2.61×	220	0.48×
PicoGRAM (tuned for each metric)	2^{16}	3.94	1.34×	**156**	**0.41**×	**9.73**	**0.095**×
	2^{30}	18.5	1.41×	**600**	**0.44**×	**41.0**	**0.090**×

Concrete performance. We evaluated the concrete performance of PicoGRAM. In Table 2, we compare PicoGRAM's performance with two baselines: 1) state-of-the-art RAM-model 2-party computation [WCS15, LWN+15] (also called the "interactive" baseline), and 2) state-of-the-art Garbled RAM [PLS23]. The metrics we focus on include the communication cost, the end-to-end time, and the online time. Note that when reporting each metric, we are using a different variant of the scheme specifically optimized for the metric of concern.

Making RAM-model 2-party computation (2PC) practical for big data has been a long-cherished dream of our community. Table 2 shows that excitingly, standing on the shoulders of recent efforts [HKO22, PLS23, HKO23], practical Garbled RAM can finally beat the interactive baseline [WCS15, LWN+15] in terms of both overall time and online time! Specifically, relative to the prior best interactive baseline, although PicoGRAM incurs 34% to 41% more communication, it achieves roughly **2.3**× and **11**× improvement in overall time and online time, respectively. Compellingly, since PicoGRAM compresses the round complexity to two, all the garbling and transmission of the garbled circuitry can be now performed offline. Therefore, in applications where the online response time is critical, PicoGRAM has a significant advantage over the interactive baseline.

Relative to the prior best Garbled RAM, PicoGRAM achieves **4.2 - 6.5**× improvement in communication, **3.1 - 5.9**× improvement in overall time, and **3.2 - 5.4**× improvement in online time. Note that we use NanoGRAM [PLS23] as the baseline here since its concrete performance is better than tri-state [HKO23] and VISAs [YPHK23] (despite being asymptotically slightly worse than tri-state).

Comparison with "ORAM + trusted hardware" and PIR. Today, real-life applications that require privately accessing large datasets such as Signal's private contact discovery [sig], oblivious accesses to blockchain data [olab], and Meta [met] settle for an "trusted hardware + ORAM" solution, not because people believe trusted hardware to be bullet-proof, but more as a near-term compromise in exchange for its fast performance and the ability to support general computation. However, many industry leaders want to remove the trusted hardware assumption, and are thus eyeing a cryptography-based solution for the medium to longer term.

As a reality check, the state-of-the-art "trusted hardware + ORAM" solution [olaa] can support a single key-value look up in $20 - 50\mu s$ for a database of $N = 2^{16}$ to $N = 2^{30}$ with 64-byte records. A state-of-the-art Private Information Retrieval (PIR) scheme can achieve roughly 12ms online computation and 100ms overall time for a database of size roughly 100GB [ZPSZ24] (assuming about 60ms ping latency). Besides Garbled RAM, PIR is another promising approach for removing

the trusted hardware. However, although state-of-the-art PIR schemes [ZPSZ24] enjoy similar online time as PicoGRAM, they suffer from the following drawbacks relative to PicoGRAM [Shi25]:

1. So far, practical PIR cannot support generic computation, which makes it difficult for software maintenance and updates. The more general form that supports generic computation called RAM-model FHE [LMW23] remains in theory land.
2. So far, practical PIR schemes are in the so-called client-preprocessing model [CHK22, ZPSZ24], which makes it challenging to deploy in scenarios with fast-evolving databases.

Therefore, in summary, our work suggests that practical Garbled RAM is a promising direction towards eventually removing the trusted hardware in high-value applications such as Signal, blockchains, and Meta, while retaining the generality and relative ease of software updates.

1.3 Technical Highlight

Inefficiencies in existing garbled RAM. First, there remains an asymptotic gap to optimality. Recent works [PLS23, HKO23], including NanoGRAM [PLS23] and tri-state GRAM [HKO23], achieve a per-instruction communication cost of $O\left(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot \mathsf{poly}\log\log N\right)$ bits, where N is the RAM's space, W is the word width, and λ is the security parameter. For word size $W = \Omega(\log^2 N)$, the cost simplifies to $O(\lambda \cdot W \cdot \log N \cdot \mathsf{poly}\log\log N)$, which remains a $\mathsf{poly}\log\log N$ factor away from optimality in N due to known logarithmic Oblivious RAM lower bounds [GO96, BN16, LN18, PLS23].

Second, although known garbled RAM schemes reduce the round complexity to two, they fail to match the best known RAM-model 2PC schemes [WCS15, Kel20] in terms of communication cost (i.e., the number of bits transmitted). As mentioned, state-of-the-art RAM-model 2PC achieves a per-instruction communication cost of $O(\lambda \cdot W \cdot \log N \cdot \omega(1))$ for sufficiently large words, and the best known garbled RAMs [PLS23, HKO23] are a $\mathsf{poly}\log\log$ factor worse. For example, for $N = 2^{20}$, tri-state GRAM and NanoGRAM consume 14.9× and 6.8× more bandwidth, respectively, compared to state-of-the-art interactive RAM-model 2PC [WCS15]. This bandwidth overhead demonstrates the concrete performance penalty resulting from the extra $\mathsf{poly}\log\log$ factors.

Last but not least, both NanoGRAM [HKO22, PLS23] and tri-state GRAM [HKO23] assess concrete performance only via cost simulators, which are inadequate for evaluating the actual running time of the garbler and evaluator. A full-fledged implementation remains essential for a comprehensive understanding of practical performance.

In existing garbled RAM constructions [HKO22, PLS23, HKO23], a core building block is a garbled stack, which over time receives an array of t data elements, each of width W, marked with a bit indicating whether the element is wanted. The stack then routes all desired elements to the front of the output array, discarding the undesired ones. Prior works showed how to construct such a garbled stack of size $O(\lambda \cdot W \cdot \log t)$. Since the garbled stack is repeatedly used in the garbled RAM construction, it becomes a performance bottleneck, both asymptotically and concretely.

Our novel techniques. We devise a new garbling technique that reduces the garbled stack size to $O(\lambda(W + \log t))$. This improvement not only eliminates the extra $\mathsf{poly}\log\log$ factors asymptotically, but also significantly improves the concrete cost of garbled RAM.

To achieve this, our key idea is to augment the tri-state circuit model proposed by Heath et al. [HKO23] with SIMD gates, which can apply the same (routing) operations to the W bits of a data element. We then devise a new technique, based on the DDH assumption, to garble SIMD gates efficiently. We believe that our SIMD garbling technique can be of independent interest, for

example, in constructing customized garbled data structures and algorithms without relying on the generic garbled RAM transformation.

1.4 Additional Related Work

We now review some additional related work.

Succinct Garbled RAM. A theoretical line of work on succinct Garbled RAM [LP14, BGL+15, CHJV15, KLW15, CCHR16, CH16, CCC+16, ACC+16, AL18] showed that assuming the existence of indistinguishability obfuscation (iO) [GGH+13a, JLS20], we can garble a RAM program such that the size, space requirements, and runtime of the garbled program are the same as those of the input program, up to polylogarithmic factors and a polynomial in the security parameter. While this line of work is theoretically fascinating, it is completely impractical due to the use of iO. Also, the total cost of these schemes (bandwidth + computation) are asymptotically worse than PicoGRAM— in fact, this line of work never even bothered to spell out the "poly" term in the polylogarithmic overhead.

Oblivious RAM. A Garbled RAM scheme can be viewed as a non-interactive version of Oblivious RAM [GO96]. Known Garbled RAM [LO13, GHL+14, GLO15, LO15, HKO22, PLS23, HKO23] and RAM-model 2PC [GKK+12, WCS15, LWN+15, Kel20] schemes are also built from an underlying ORAM scheme. It is well known that an ORAM scheme must suffer from a logarithmic slowdown relative to the original program [GO96, LN18]. In the *computationally secure* setting, recent work has shown how to get an ORAM scheme that matches the logarithmic lower bound [PPRY18, AKL+23, AKLS23], relying on the existence of pseudorandom functions (PRFs). In comparison, the best known *statistically secure* ORAMs [SDS+18, WCS15] are a logarithmic factor worse in performance asymptotically.

Despite this, all known RAM-model 2PC as well as garbled RAM schemes adopt a *statistically secure* ORAM for two reasons: 1) the best known statistically secure constructions [SDS+18, WCS15] are simple and enjoy small constants in the big-O; and 2) using a computationally secure scheme would require garbling a PRF, which is concretely inefficient due to the non-blackbox use of cryptography. For these reasons, we use 2PC atop Circuit ORAM [WCS15] as the interactive baseline in our paper.

2 Roadmap

2.1 Background: Tri-State Circuits and Inefficient Strawman

From standard Boolean circuits to tri-state circuits. As mentioned, the main challenge is due to the difficulty of efficiently expressing a memory access in a standard Boolean circuit. An alternative way to view this problem is that the garbler cannot predict in advance which address will be accessed at each time step. Thus, it cannot determine a priori which *label* will encode the memory states or the outcome of memory fetches.

The elegant tri-state GRAM work [HKO23] proposed a solution to this problem. Instead of using standard Boolean circuits as the underlying computation model, it introduces a new computation model called tri-state circuits (TSC). The key differences between a standard Boolean circuit and a TSC are as follows:

1. *Order of evaluation.* In a standard Boolean circuit, we can always evaluate the circuit by populating the wires in some fixed topological order, regardless of the input values. In contrast,

a TSC allows gates to have control signals. Such a gate can be invoked only if its control signal receives a certain input value (e.g., 0). Since the control signals eventually depend on the input values, whether each gate can be invoked and the order in which the gates are invoked depend on the input values.

2. *Subset of active gates under partial input.* As a direct consequence of the above, in a TSC, upon receiving a *partial* input, the subset of gates that can be invoked depends on the input values. In contrast, in a standard Boolean circuit, the subset is determined solely by which input wires are populated, irrespective of their values.

Note that the second difference is relevant even if the RAM program always receives its input all at once. This is because, from the perspective of the garbled gadgets underlying the garbled RAM, inputs may still arrive gradually over time.

Using a TSC to express Circuit ORAM. With TSCs, we can encode dynamic memory accesses efficiently with the help of an Oblivious RAM (ORAM) algorithm. It is well-known that every RAM program can be converted to an ORAM with only polylogarithmic blowup in its running time. Specifically, we rely on Circuit ORAM [WCS15]. In Circuit ORAM — ignoring the recursion for now — every memory access translates to accessing $O(1)$ random paths in a binary tree of size $O(N)$, where each path goes from the root to some leaf node[2]. Further, which paths are invoked is computed dynamically at evaluation time.

To gain intuition, it helps to focus on the following technical core of the problem, that is, *routing data along a tree path*. Imagine that in each time step, the root receives some data (denoted data) and an address (denoted addr) as input, and it needs to pass the data to a leaf node selected by the address. This task can be efficiently expressed as a TSC — imagine that each node in the tree is associated with some routing circuitry, and which nodes are invoked in each time step is determined dynamically. As mentioned, this can be naturally expressed in a TSC by using control wires to signal whether a gate can be invoked.

Heath et al. [HKO23] proposed an elegant approach for garbling such a TSC, as shown in Figure 1. Specifically, one can imagine that each non-leaf node u is associated with two garbled stacks, denoted StackL and StackR, used for routing to u's left and right children, respectively. Below we take StackL as an example since StackR is symmetric. The stack has $n_{\max}(u)$ input wires for receiving up to $n_{\max}(u)$ inputs of the form $(\mathsf{addr}, \mathsf{data})$ — see Table 3 for the choices of $n_{\max}(u)$. Whenever it receives a new input, the stack examines a corresponding bit in the addr field: if the bit is 1, no new output wire will be populated; else if the bit is 0, then the next group of $W = |\mathsf{addr}| + |\mathsf{data}|$ output wires will be populated with the value $(\mathsf{addr}, \mathsf{data})$, to be passed to the left child. Further, when the output wires are populated, the tuple $(\mathsf{addr}, \mathsf{data})$ will be encoded under the label that the left child expects. Specifically, for any node v, the τ-th time it is invoked, it expects the inputs to be encoded using the label L_τ^v — earlier works also referred to τ as node v's *local clock* [PLS23].

For simplicity, we consider the most important special case when $T = N$, meaning that the RAM's running time equals its space. Given this case, the other cases $T < N$ and $T > N$ can be easily handled using the analysis in Section A.1. In this setting, we know that a stack at the root level (i.e., level 0) will be invoked exactly N times; a stack at level 1 will be invoked $N/2$ times in expectation; a stack at level 2 will be invoked $N/4$ times in expectation; and so on. Finally, a stack at level $\log(N/B)$ (i.e., the last level of the ORAM tree) will be invoked $B \in \log N \cdot \omega(1)$ times in expectation for some super-constant function $\omega(1)$. Due to the Chernoff bound, it suffices to

[2]For best concrete efficiency, the read path is random while the eviction paths should be chosen based on a deterministic reverse lexicographical ordering [WCS15]. However, this detail is not important at this point.

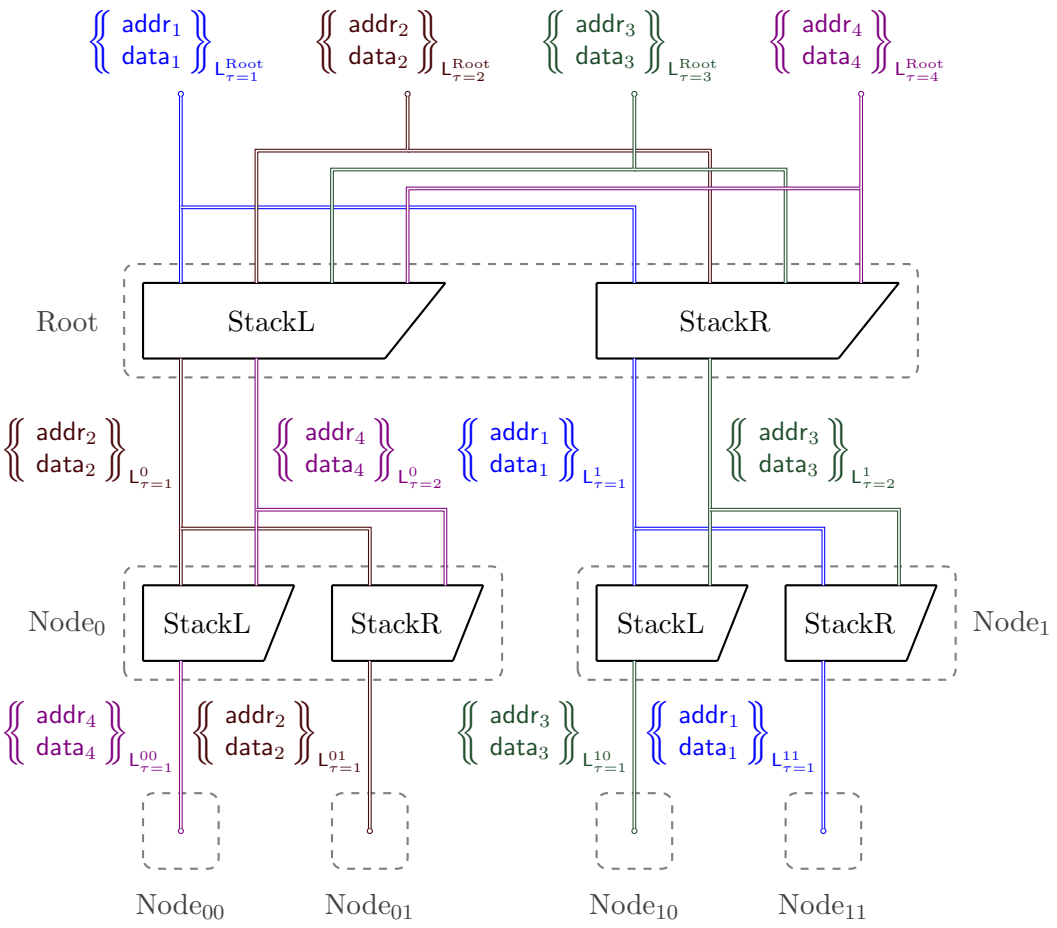
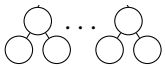


Figure 1: Routing data in a tree using a SIMD tri-state circuit. The notation $\{\!\{\mathsf{addr}, \mathsf{data}\}\!\}_{\mathsf{L}_\tau^u}$ means that the tuple $(\mathsf{addr}, \mathsf{data})$ is encoded under the τ-th label associated with node u. Each non-leaf node u maintains two garbled stacks, StackL and StackR, for routing data to its children. Each stack looks at a bit in addr to decide whether to route data to its left or right child, populating the next unconsumed cable of either StackL or StackR. For example, $(\mathsf{addr}_1, \mathsf{data}_1)$ is eventually routed to the leaf Node_{10}, $(\mathsf{addr}_2, \mathsf{data}_2)$ is eventually routed to the leaf Node_{00}, and so on.

Table 3: Amortized communication costs from the stacks for routing along a path in a single ORAM tree where $B = \log N \cdot \omega(1)$. In the full ORAM, each recursion level will consume $O(1)$ copies of this routing circuitry, for the read and eviction operations, respectively.

ORAM Tree	**Exp.# access**	**Max# access**	**Stack Comm.** *TSC Stack*	**Stack Comm.** *PicoGRAM Stack*	**Bkt Comm.** Both
	N	$2N$	$O(\lambda W \log 2N)$	$O(\lambda(W + \log 2N))$	$O(\lambda W \log N)\omega(1)$
	$N/2$	N	$O(\lambda W \log N)$	$O(\lambda(W + \log N))$	$O(\lambda W)$
⋮ ⋮ ⋮	⋮	⋮	⋮	⋮	⋮
	$2B$	$4B$	$O(\lambda W \log 4B)$	$O(\lambda(W + \log 4B))$	$O(\lambda W)$
	B	$2B$	$O(\lambda W \log 2B)$	$O(\lambda(W + \log 2B))$	$O(\lambda W B)$
Total			$O(\lambda W \log^2 N)$	$O(\lambda(W \log N + \log^2 N))$	$O(\lambda W \log N)\omega(1)$

provision each level with a maximum number of invocations that is twice its expectation. Since we stop the tree at level $\log(N/B)$ with a super-logarithmic number of invocations in expectation, it is not hard to see that the actual number of invocations does not exceed the provisioned maximum except with negligible probability.

Inefficiency of tri-state GRAM. The cost of tri-state GRAM [HKO23] is dominated by garbling the routing stacks in the ORAM tree. Specifically, tri-state GRAM showed how to garble a stack achieving the following efficiency:

Fact 2.1 (Inefficient garbled stack of tri-state GRAM [HKO23])**.** Garbling a stack provisioned for t accesses results in a garbled circuit of size $O(\lambda \cdot W \cdot \log t)$ amortized.

As shown in Table 3, plugging in Fact 2.1 and summing over all garbled stacks in the entire ORAM tree, the amortized communication cost comes to $O(\lambda \cdot W \cdot \log^2 N)$.

To complete the analysis, we must also account for the fact that a full ORAM scheme has $O(\log N)$ recursively constructed ORAM trees: one data tree that adopts a word size of $W = |\mathsf{data}| + |\mathsf{addr}|$, and multiple metadata trees. The ℓ-th metadata tree has $2^{O(\ell)}$ leaves and a word size of $O(\ell)$. Therefore, the amortized cost for garbling the stacks across all recursion levels is

$$O(\lambda \cdot W \cdot \log^2 N) + \sum_{\ell=1}^{O(\log N)} O(\lambda \cdot \ell \cdot \ell^2) = O(\lambda \cdot (W + \log^2 N) \cdot \log^2 N)$$

Remark 2.2. Table 3 shows an asymptotically slower variant of tri-state GRAM [HKO23]. To achieve their asymptotics as claimed in Table 1, they need an additional "reset" trick similar to Nanogram [PLS23]. Specifically, instead of provisioning for $2N, N, N/2, \ldots$ stack accesses at each level of the ORAM tree, they create stacks provisioned for polylogarithmically many accesses at all levels, but each level must initialize a new garbled stack every polylogarithmically many accesses. Using the reset trick, each single ORAM tree in tri-state GRAM incurs $O(\lambda \cdot W \cdot \log N \cdot \log\log N)$ cost instead of $O(\lambda \cdot W \cdot \log^2 N)$ per access. Therefore, the total garbled stack cost over all recursion levels is $O(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot \log\log N)$. However, this reset trick does not further benefit PicoGRAM, as we already get significant savings by reducing the multiplicative overhead to additive — see Section 2.2 for more details.

2.2 Our Blueprint: Extending Tri-State Circuits with SIMD Gates

From multiplicative to additive. Tri-State GRAM's garbled stack incurs $O(\lambda \cdot W \cdot \log t)$ cost for supporting t accesses, where the W and the $\log t$ factors accumulate *multiplicatively*. We devise new techniques leading to a new garbled stack whose cost is only $O(\lambda \cdot (W + \log t))$, where the W and the $\log t$ factors are now *additive*.

Assuming we can achieve this, then, by Table 3 and the analysis shown in Section 2.1, we can reduce the total garbled stack cost to $O(\lambda \cdot (W \cdot \log N + \log^3 N))$, summing over all recursion levels of the ORAM. At this point, the garbled stack cost is no longer dominating. Instead, PicoGRAM's cost is dominated by the data-read circuitry associated with the nodes in the ORAM tree. Specifically, besides the routing circuitry (i.e., the stacks), each node in the ORAM tree also has additional garbled circuitry that implements a Garbled Bucket, for reading and writing the memory words stored in this node. In PicoGRAM, every node has a constant-sized Garbled Bucket, except the root and the leaf levels, which have super-logarithmically sized Garbled Buckets. Moreover, the circuit size for each access in Circuit ORAM is linear in the bucket size and the word width, assuming a word-RAM model where the word width $W \in \Omega(\log N)$. In Table 3, we also account for the cost of the Garbled Buckets. For a single ORAM tree, this cost comes to $O(\lambda W \log N)\omega(1)$. Since there are logarithmically many trees, one with word size W, and the rest with word size $W' = O(\log N)$, the total cost across all recursion levels is

$$O(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot \omega(1))$$

With some additional algorithmic tricks, we can further reduce the above expression to $O(\lambda \cdot (W \cdot \log N \cdot \omega(1) + \log^3 N))$. We describe the details in Section A.2.

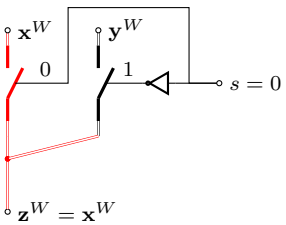
Introducing SIMD gates. We observe that the dominant cost of the garbled stack comes from the routing circuitry that routes all W bits of the payload in a synchronized fashion. We refer to such operations as SIMD (Single Instruction Multiple Data). Our key idea is to extend the original tri-state circuit—designed to operate on individual bits—with new SIMD gates that process groups of W bits in parallel. Furthermore, leveraging the DDH assumption, we propose an efficient method for garbling these SIMD gates.

Henceforth, we use the term *cable* to denote a group of W wires, where each wire carries a single bit. A wire that is not part of a cable is sometimes called an *independent wire*. A wire that is part of a cable is called a *subwire*. We use boldface notation such as $\mathbf{x}^W$ to denote a cable, whose i-th wire is denoted as x_i. Our enriched circuit model consists of the following types of gates:

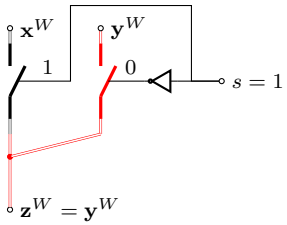
- **Boolean gate ($z \leftarrow f(x, y)$):** A Boolean gate takes two independent input wires x and y and outputs an independent wire z. It defines an evaluation rule $z \leftarrow f(x, y)$, where f is a Boolean function. The Boolean gate can be evaluated only when both x and y are set.
- **Group gate $y_i \xleftarrow{\text{grp}} x$:** The Group gate converts an independent wire x into a subwire y_i of a cable. It defines the evaluation rule $y_i \leftarrow x$.
- **Ungroup gate $y \xleftarrow{\text{ungrp}} x_i$:** The Ungroup gate converts a subwire x_i back to an independent wire y. It defines the evaluation rule $y \leftarrow x_i$.
- **Switch gate ($\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$):** A Switch gate connects two cables $\mathbf{x}^W$ and $\mathbf{y}^W$ of equal width and an independent control wire c that determines whether the gate is active. If $c = 0$, the gate enforces equality between every pair of subwires x_i and y_i. In other words, the Switch gate defines $2W$ evaluation rules: $c = 0 \Rightarrow y_i \leftarrow x_i$ and $c = 0 \Rightarrow x_i \leftarrow y_i$ for $i \in [W]$. The gate may be evaluated partially even if some subwires are not set. Conceptually, our Switch gate has

the same functionality as a collection of W Buffer gates sharing the same control wire in prior works [HKO23, Hea24]. Moreover, we remove the Join gate from prior works [HKO23, Hea24] for simplicity, and instead allow a subwire to be set by multiple gates.

Operational semantics. Unlike the standard Boolean circuit model, the SIMD tri-state circuit allows a subwire to be set by multiple gates. In Figure 2, we show an example that employs this behavior to achieve indirect addressing with a 1-bit address space. To avoid non-deterministic behavior during evaluation, we say a SIMD tri-state circuit is **well-formed** only if every wire in the circuit can be set to a unique value, regardless of the order of evaluation. For well-formed SIMD tri-state circuits, we further define a stronger property called **strictly well-formed**, which enforces a partial ordering on the wires. We formally describe our computational model in Section 3.



(a) When $s = 0$, the Switch on the left is active and can set each i-th subwire of $\mathbf{z}^W$ to the value of x_i.



(b) When $s = 1$, the Switch on the right is active and can set each i-th subwire of $\mathbf{z}^W$ to the value of y_i.

Figure 2: A 1×2 multiplexer implemented with SIMD tri-state gates. Since the two Switches can never be active at the same time, the value of each z_i is uniquely determined when the inputs $\mathbf{x}^W$, $\mathbf{y}^W$, and s are fixed. Compared to Boolean circuit, the SIMD tri-state circuit allows z_i to be set in an eager manner. For instance, when $s = 0$, the value of z_i can be set to x_i without waiting for the value of y_i to be known.

2.3 Garbling SIMD Tri-State Circuits

In this work, we propose two garbling schemes for the SIMD tri-state circuit, both assuming DDH. The first scheme achieves the asymptotic efficiency of GRAM and is proven secure in the plain model, while the second scheme offers better concrete efficiency at the cost of relying on the Random Oracle (RO) model. Additionally, the first scheme requires the SIMD tri-state circuit to be strictly well-formed, whereas the second scheme only requires well-formedness, which further helps to reduce the concrete circuit size of GRAM.

We focus on introducing the first scheme in the main body and defer the second scheme to Section C.

Background: Yao's Garbling for Boolean Gates [Yao86]. We use the standard Yao's garbling scheme [Yao86] to garble the Boolean gates in our SIMD tri-state circuit. For every independent wire x, the garbler samples two random labels $\mathsf{L}_{x=0}$ and $\mathsf{L}_{x=1}$ from $\{0,1\}^\lambda$, where λ is the security parameter. Assume that we have an encryption scheme that reveals whether the decryption key is correct with high probability. To garble a Boolean gate $z \leftarrow f(x, y)$, the garbler encrypts the output label $\mathsf{L}_{z=f(a,b)}$ with the labels of the input wires $\mathsf{L}_{x=a}$ and $\mathsf{L}_{y=b}$ for every $a, b \in \{0,1\}$, and randomly shuffles the ciphertexts before sending them to the evaluator. The evaluator, in turn, only learns the label corresponding to the runtime value of each wire, called the active label. Initially, the evaluator only holds the active labels of the circuit's input wires. If the evaluator

learns the active labels of both x and y in the Boolean gate, they can decrypt the corresponding ciphertext and obtain the active label of z.

Encoding of subwires. Next, we propose a novel technique to encode the subwires in the SIMD tri-state circuit, enabling efficient garbling of Switch gates.

Conceptually, our garbling scheme encodes all cables in the circuit with two "orthogonal" sets of keys. The first set of keys is global for all cables but unique for each subwire offset. The second set of keys is local to each cable but shared among all subwires of that cable.

Let $\mathbb{G}_q$ be a prime-order group where the DDH assumption holds and g be a generator of $\mathbb{G}_q$. The garbler samples $2 \cdot W_{\max}$ global keys $\Gamma_{i,b}$ from $\mathbb{Z}_q^*$, where $i \in [W_{\max}]$, $b \in \{0,1\}$, and $W_{\max}$ is the maximum width of any cable in the circuit. For each W-bit cable $\mathbf{x}^W$, the garbler samples a local key $\mathsf{K}_{\mathbf{x}}$ from $\mathbb{Z}_q^*$.

We define the labels of the i-th subwire as

$$\mathsf{L}_{x_i=0} = g^{\mathsf{K}_{\mathbf{x}} \cdot \Gamma_{i,0}}, \quad \mathsf{L}_{x_i=1} = g^{\mathsf{K}_{\mathbf{x}} \cdot \Gamma_{i,1}}.$$

As a sanity check, the evaluator should not be able to infer whether a subwire carries 0 or 1 from the labels. Consider t cables each of width W. Let k_τ be the local key of the τ-th cable. Then, the subwire labels are a subset of the following $t \times 2W$ matrix over $\mathbb{G}_q$:

$$M = \begin{pmatrix} g^{k_1 \cdot \Gamma_{1,0}} & \dots & g^{k_1 \cdot \Gamma_{W,0}} & g^{k_1 \cdot \Gamma_{1,1}} & \dots & g^{k_1 \cdot \Gamma_{W,1}} \\ \vdots & \ddots & \vdots & & & \\ g^{k_t \cdot \Gamma_{1,0}} & \dots & g^{k_t \cdot \Gamma_{W,0}} & g^{k_t \cdot \Gamma_{1,1}} & \dots & g^{k_t \cdot \Gamma_{W,1}} \end{pmatrix}$$

It has been shown in the work of the BHHO cryptosystem [BHHO08] that M is computationally indistinguishable from a random matrix. Nonetheless, in Section 4, we prove more directly that all the subwire labels can be replaced one by one with random group elements.

Garbling Switch gates. To garble a Switch gate $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$, the garbler encrypts the ratio of the two cables' local keys with the zero-label of the control wire c. Specifically, the garbler sends $\mathsf{Enc}_{\mathsf{L}_{c=0}}\left(\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}\right)$.

If the evaluator learns $\mathsf{L}_{c=0}$, the zero-label of the control wire, they can decrypt the ciphertext and obtain $\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}$. If the evaluator also learns the active label of a subwire x_i, they can compute the active label of the corresponding subwire y_i as follows:

$$\mathsf{L}_{y_i=b} = g^{\mathsf{K}_{\mathbf{y}} \cdot \Gamma_{i,b}} = \left(\mathsf{L}_{x_i=b}\right)^{\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}}.$$

Similarly, the evaluator can compute the modular inverse of $\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}$ and obtain the active label of x_i from that of y_i.

These properties match our evaluation rule for the Switch gate: the evaluator can propagate values between each pair of subwires if and only if the control wire is 0.

The garbled Switch gate reveals the value of the control wire c, as the evaluator learns whether the active label of the control wire is the correct decryption key. Previous works [HKO23, Hea24] have shown that such leakage does not affect the security of the final Garbled RAM construction, due to the randomness of the ORAM algorithms.

We set the prime order q to be an $O(\lambda)$-bit prime. The communication cost of our garbled Switch gate is $O(\lambda)$ bits, independent of the width W of the cables. This is a significant improvement over previous works [HKO23, Hea24], which require $O(W \cdot \lambda)$ bits of communication to garble an equivalent Switch gate.

Garbling Group and Ungroup gates. Finally, we garble Group and Ungroup gates using Yao's garbling [Yao86], similar to Boolean gates. Since each gate has only one input wire, the garbler sends two ciphertexts $\mathsf{Enc}_{\mathsf{L}_{x_{\text{in}}=0}}(\mathsf{L}_{x_{\text{out}}=0})$ and $\mathsf{Enc}_{\mathsf{L}_{x_{\text{in}}=1}}(\mathsf{L}_{x_{\text{out}}=1})$ in random order, where x_{in} is the input wire and x_{out} is the output wire. The communication cost of garbling a Group or Ungroup gate is $O(\lambda)$ bits.

2.4 Efficient Garbled Stack

Applying SIMD gates to the stack. Recall that our goal is to eliminate the multiplicative factor W from the communication cost of garbled stack constructions in prior work [HKO23]. To achieve this, we store data words as cables within the stack and route them using our new Switch gates. Although we need to convert independent wires into cables when pushing data into the stack, and convert them back when popping data, the number of Group and Ungroup gates required depends only on the number of input and output wires, not on the size of the stack circuit. As a result, the total cost of the garbled stack is $O(\lambda \cdot (W + \log t))$ bits, where t is the maximum number of accesses to the stack and W is the word width.

Optimizing stack circuitry. While our SIMD gates reduce the asymptotic cost of GRAM, we further optimize the stack's circuitry to improve PicoGRAM's concrete performance. We observe that it suffices to construct the garbled stack based on an oblivious stable compaction algorithm [Goo11], which is much simpler and more efficient than the oblivious stack data structure [ZE13] used in previous works [HKO22, PLS23, WCS15]. To adapt the original compaction algorithm [Goo11] into a SIMD tri-state circuit, we devised a new method to compute the routing plan, ensuring that the GRAM is strictly well-formed. This optimization alone reduces the communication cost of the garbled stack by a factor of 4.1× for $N = 2^{16}$ and word width $W = 64$, even without using SIMD gates. Combined with the SIMD garbling technique, we achieve a total of 24× savings for the stack at $N = 2^{16}$, and 29× at $N = 2^{20}$.

3 Computational Model: SIMD Tri-State Circuits

In this section, we formally define SIMD tri-state circuits, which extend the original tri-state circuit model [HKO23] with gates that support SIMD (Single Instruction, Multiple Data) operations.

3.1 Definitions

Wires and cables. As in the original tri-state circuit model [HKO23], we treat a circuit as a finite state machine, where each wire holds a state of 0, 1, or $\mathcal{Z}$, meaning the wire has not yet received any signal.

In addition to wires that each carry a single bit, we introduce the concept of a **cable**, representing a group of W wires that act in a SIMD fashion. A wire is **set** if its state is not $\mathcal{Z}$. A wire can belong to at most one cable. Wires that do not belong to any cable are called *independent wires*.

We denote an independent wire with a lowercase letter, e.g., x, and a cable of W wires with a boldfaced letter and a superscript, e.g., $\mathbf{x}^W$, where W is the **width** of the cable. The i-th wire of $\mathbf{x}^W$ is written as x_i, where $i \in [W]$ is called the **offset** of the subwire.

Each wire or cable has a type: *input*, *output*, or *internal*. We may assume all cables are of type *internal*, i.e., all input and output wires are independent wires. This is because we can always introduce additional Group and Ungroup gates (defined below) to convert an independent wire to a subwire of a cable and vice versa, without increasing the circuit size asymptotically.

Evaluation rules. An *evaluation rule* specifies how to set a wire. We represent an evaluation rule as $\mathsf{P} \Rightarrow x \leftarrow \mathsf{S}$, where P is a precondition and S is an expression that computes the state of wire x. To apply the evaluation rule, S must evaluate to either 0 or 1, and P must be true. If P is trivial, we write the rule simply as $x \leftarrow \mathsf{S}$. The evaluation rule is allowed to change the state of a wire.

Gates. Each gate defines a set of evaluation rules applied to wires or cables. Our SIMD tri-state circuit comprises the following types of gates:

- **Boolean** ($z \leftarrow f(x, y)$): A Boolean gate sets an independent output wire z based on a public Boolean function $f : \{0,1\} \times \{0,1\} \mapsto \{0,1\}$ and the values of two independent input wires x and y.
- **Group** ($y_i \xleftarrow{\text{grp}} x$): A Group gate sets a subwire of a cable $\mathbf{y}^W$ to the value of an independent wire x.
- **Ungroup** ($y \xleftarrow{\text{ungrp}} x_i$): An Ungroup gate sets an independent output wire y to the value of a subwire of a cable $\mathbf{x}^W$.
- **Switch** ($\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$): A Switch gate connects two cables $\mathbf{x}^W$ and $\mathbf{y}^W$ of equal width and an independent wire c, called the **control wire**. For each $i \in [W]$, when $c = 0$, the gate is **active** and can either set the i-th subwire x_i to the value of y_i or set y_i to the value of x_i, depending on which wire is set first. When $c = 1$, the gate is **inactive**, meaning it cannot set any wire.

Number of wire sources. As in the classic circuit model, we require each independent wire to be either an input wire of the circuit or the output wire of a single gate. However, we do not apply this restriction to cables, since the subwires of cables are meant to be set by one of multiple gates, depending on the runtime state of the circuit. Also, we do not bound the circuit's fan-out: each wire can be input to any number of gates.

3.2 Operational Semantics

Evaluating a SIMD tri-state circuit. Unlike traditional Boolean circuits, in a (SIMD) tri-state circuit, both the order in which gates are evaluated and the directions of evaluation may vary based on the input values. Thanks to this new semantics, we can express RAM computations efficiently in a (SIMD) tri-state circuit [HKO23].

The *state* of a SIMD tri-state circuit $\mathcal{C}$ refers to the combined states of all wires in $\mathcal{C}$. Evaluating the circuit $\mathcal{C}$ can thus be viewed as a sequence of state transitions.

- *Initial state.* Initially, every input wire receives a bit, and all other wires have the state $\mathcal{Z}$.
- *Evaluation.* At each time step, look for a gate with an evaluation rule that can be invoked, and execute the evaluation rule to populate a wire.
- *Output.* Eventually, output the values on the output wires.

The above evaluation algorithm exhibits non-determinism: a wire may be set by the evaluation rules of multiple gates. If, under some evaluation state, multiple rules (associated with gates) are invocable, the evaluator may break ties arbitrarily and select one of them. Despite this non-deterministic behavior, we want a *well-formed* SIMD tri-state circuit to satisfy the following natural property: given any valid input inp, every wire can be set, and moreover, that value must be unique.

Table 4: **Gate set of our SIMD tri-state circuit**.

Type	Notation	Evaluation Rules
Boolean	$z \leftarrow f(x, y)$	$z \leftarrow f(x, y)$
Group	$y_i \xleftarrow{\text{grp}} x$	$y_i \leftarrow x$
Ungroup	$y \xleftarrow{\text{ungrp}} x_i$	$y \leftarrow x_i$
Switch	$\mathbf{x}^W$ —$\overset{c}{/}$— $\mathbf{y}^W$	$c = 0 \Rightarrow y_i \leftarrow x_i$ $c = 0 \Rightarrow x_i \leftarrow y_i$

More specifically, under any fixed input inp: 1) a wire should not take on two different values across two evaluations; and 2) a wire should not acquire a value $b \in \{0, 1\}$ and then later switch to $1 - b$ in the same evaluation. We introduce a sufficient condition, called *dynamic well-formedness*, to formalize this notion.

Henceforth, we say that a state $\mathcal{S}$ of $\mathcal{C}$ is **total** if and only if no wire holds $\mathcal{Z}$. A state $\mathcal{S}$ of $\mathcal{C}$ is **reachable** from another state $\mathcal{S}'$ of $\mathcal{C}$ if and only if $\mathcal{S}$ can be obtained from $\mathcal{S}'$ by applying a sequence of evaluation rules defined by the gates in $\mathcal{C}$.

We often use inp to denote the input or the input wires to $\mathcal{C}$ and $|\mathsf{inp}|$ to represent the input length.

Definition 3.1 (Well-formed)**.** A SIMD tri-state circuit $\mathcal{C}$ is **well-formed** if and only if for any input $\mathsf{inp} \in \{0, 1\}^{|\mathsf{inp}|}$, there is a unique total state $\mathcal{S}$ of $\mathcal{C}$ reachable from the initial state determined by inp.

Efficient evaluation. As a direct implication of Definition 3.1, a well-formed SIMD tri-state circuit $\mathcal{C}$ can be evaluated *efficiently*. Specifically, in the evaluation algorithm above, at each time step, we always look for a rule that populates a previously unset wire. This way, the evaluation algorithm completes in $|\mathcal{C}|$ steps, where $|\mathcal{C}|$ denotes the size of the circuit. We prove this formally in Lemma 3.2. Note that the evaluation rules in Table 4 also specify a **topological ordering for evaluation** that governs the order in which gates are invoked for a fixed input.

Lemma 3.2. *Let $\mathcal{C}$ be a well-formed SIMD tri-state circuit. Then, the above evaluation algorithm terminates within $O(|\mathcal{C}|)$ state transitions on any $\mathsf{inp} \in \{0, 1\}^{|\mathsf{inp}|}$, and reaches a total state at the end, as long as in every time step, we always invoke a rule that populates an unset wire.*

Proof. First, we show that during the evaluation, every wire must be set to the same value as in the unique total state $\mathcal{S}$ reachable from the initial state given by inp. Suppose that a wire x is the first wire set to a different value than in $\mathcal{S}$, and let $\mathcal{R}$ be the evaluation rule applied. Then all the rest of the wires used in $\mathcal{R}$ must have the same value as in $\mathcal{S}$ when the rule is applied, so $\mathcal{R}$ can also be applied upon $\mathcal{S}$ and transition it to another total state where x is set differently. This contradicts the uniqueness of total state.

Now, suppose that the evaluation algorithm reaches a state $\mathcal{S}'$ which is not total but no wire can be further set. Consider a state transition to the total state $\mathcal{S}$, and let y be the first wire set during the transition such that y is not set in $\mathcal{S}'$. Then the evaluation rule applied to set y must be applicable to $\mathcal{S}'$, since all the other wires used in the rule are set in $\mathcal{S}'$ and have the same value as in $\mathcal{S}$. This contradicts the assumption that no wire can be further set in $\mathcal{S}'$. Therefore, the evaluation must terminate within $O(|\mathcal{C}|)$ state transitions. □

Incremental evaluation of a tri-state circuit. If a SIMD tri-state circuit is well-formed, then it is insensitive to the order in which the evaluation rules are invoked. A direct implication is

that the circuit can be evaluated incrementally, accommodating inputs that arrive gradually rather than all at once. This is important for defining the garbled gadgets underlying our Garbled RAM scheme. For example, for the core building block stack, inputs do not arrive all at once. The evaluator must eagerly and incrementally evaluate the stack over time as more input bits become available. In fact, when the stack is integrated into a larger tri-state circuit (e.g., our Garbled RAM), subsequent input bits to the stack can only be set once part of the output bits have been evaluated.

In Section C, we propose a garbling scheme for well-formed SIMD tri-state circuits, which is secure under the random-oracle model.

Strict well-formedness. To prove the security of the garbling scheme in the plain model, we require the circuit to satisfy a stronger property called strict well-formedness, which imposes additional restrictions on the order in which wires can be set. First, we define the notion of dependency between wires.

Definition 3.3 (Dependency)**.** Let $\mathcal{S}$ be the total state of a well-formed SIMD tri-state circuit $\mathcal{C}$. We say a wire y in $\mathcal{C}$ **depends on** a wire x (or x is a **dependency** of y) with respect to $\mathcal{S}$ if any of the following conditions holds:

- y is the output wire of a Boolean, Group, or Ungroup gate with input wire x.
- y is a subwire connected to a Switch gate whose control wire is x.
- y is a subwire connected to an **active** Switch gate under the state $\mathcal{S}$, and z depends on x, where z is the subwire with the same offset in the other cable connected to the Switch gate.
- y depends on another wire z, and z depends on x.

The dependency relationship defines the order in which we replace garbled gates with their idealized counterparts when proving security through hybrid arguments. Now, we define the notion of strict well-formedness.

Definition 3.4 (Strictly Well-formed)**.** A well-formed SIMD tri-state circuit $\mathcal{C}$ is **strictly well-formed** if for any input $\mathsf{inp} \in \{0,1\}^{|\mathsf{inp}|}$, there is an ordering to set all the wires in $\mathcal{C}$ such that every wire is set after all its dependencies with respect to the unique total state determined by inp.

For a general (SIMD) tri-state circuit, there may not be an efficient algorithm to check well-formedness or strict well-formedness. However, for all circuits we construct in this paper, we can prove that they are strictly well-formed.

4 Garbling SIMD Tri-State Circuits

4.1 Definitions

Definition 4.1 (Garbling scheme)**.** A garbling scheme, defined for some computation models (e.g., SIMD tri-state circuit, RAM), consists of a tuple of possibly randomized algorithms:

- $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{params})$: upon receiving the security parameter 1^λ and parameters params specific to the computation model of concern, output a secret key sk.
- $\widetilde{\mathsf{inp}} \leftarrow \mathsf{Encode}(\mathsf{sk}, \mathsf{inp})$: upon receiving the secret key sk and input inp, output a garbled version of the input string denoted $\widetilde{\mathsf{inp}}$.

- $\widetilde{\mathcal{C}} \leftarrow \mathsf{Garble}(\mathsf{sk}, \mathcal{C})$: upon receiving sk and a deterministic program $\mathcal{C}$ under the computation model of concern, output a garbled version of the program denoted $\widetilde{\mathcal{C}}$.
- $\mathsf{out} \leftarrow \mathsf{Eval}(\widetilde{\mathcal{C}}, \widetilde{\mathsf{inp}})$: upon receiving the garbled program $\widetilde{\mathcal{C}}$ and garbled input $\widetilde{\mathsf{inp}}$, outputs the evaluation outcome out.

Later, for a SIMD tri-state circuit, we only need to include the input length $|\mathsf{inp}|$ in params; for a RAM, we include its maximum space requirement N and maximum runtime T in params.

Correctness. A garbling scheme $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Garble}, \mathsf{Eval})$ is **correct** if for polynomial-time program $\mathcal{C}$ with parameters params, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all λ, for any input inp, except with $\mathsf{negl}(\cdot)$ probability, the following holds: let $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{params})$, $\widetilde{\mathsf{inp}} \leftarrow \mathsf{Encode}(\mathsf{sk}, \mathsf{inp})$, $\widetilde{\mathcal{C}} \leftarrow \mathsf{Garble}(\mathsf{sk}, \mathcal{C})$, then, it must be that $\mathsf{Eval}(\widetilde{\mathcal{C}}, \widetilde{\mathsf{inp}}) = \mathcal{C}(\mathsf{inp})$ where $\mathcal{C}(\mathsf{inp})$ denotes the outcome of executing the program $\mathcal{C}$ on the input inp in cleartext.

Security. Since we assume a semi-honest setup and a deterministic program $\mathcal{C}$, the security definition only requires simulating the evaluator's view given the circuit's output values and the output of a leakage function leak.

Definition 4.2 (Security of garbling scheme)**.** A garbling scheme $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Garble}, \mathsf{Eval})$ is **secure** with respect to some (deterministic) leakage function leak, if and only if there exists a probabilistic polynomial-time algorithm Sim such that for any program $\mathcal{C}$ with parameters params, for any inp, the following experiments are computationally indistinguishable:

- **Real:** Run $sk \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{params})$, $\widetilde{\mathsf{inp}} \leftarrow \mathsf{Encode}(\mathsf{sk}, \mathsf{inp})$, $\widetilde{\mathcal{C}} \leftarrow \mathsf{Garble}(\mathsf{sk}, \mathcal{C})$, and output $(\widetilde{\mathcal{C}}, \widetilde{\mathsf{inp}})$.
- **Ideal:** Output $\mathsf{Sim}(1^\lambda, \mathcal{C}, \mathsf{leak}(\mathcal{C}, \mathsf{inp}), \mathcal{C}(\mathsf{inp}))$.

4.2 Our Construction from DDH

Labels. Similar to Yao's garbled circuit [Yao86], we assign two labels to each wire in a SIMD tri-state circuit, corresponding to the bits 0 and 1, respectively. Henceforth, we use the notation $\mathsf{L}_{x=b}$ to represent the label for the value $b \in \{0, 1\}$ on the wire x.

We sometimes overload the notation x to also mean the value the wire x carries; in such cases, we use the shorthand L_x for the label of wire x when it carries the value x.

Labels of an independent wire. The garbler samples two labels from $\{0, 1\}^\lambda$ for each independent wire, as in Yao's garbling scheme [Yao86].

Labels of a cable. We devise a new technique for computing labels of a cable. Let $\mathbb{G}_q$ be a public prime-order group of order q where DDH is assumed to hold and g be a generator. Let $W_{\max}$ be the maximum width of any cable in the system. The garbler draws $2W_{\max}$ global keys $\Gamma_{1,0}, \Gamma_{1,1}, \Gamma_{2,0}, \Gamma_{2,1}, \ldots, \Gamma_{W_{\max},0}, \Gamma_{W_{\max},1}$ from $\mathbb{Z}_q^*$ and hides them from the evaluator.

For each cable $\mathbf{x}^W$, the garbler also samples a secret cable key $\mathsf{K}_\mathbf{x} \in \mathbb{Z}_q^*$. Then the labels of the i-th wire x_i of the cable are defined as

$$\mathsf{L}_{x_i=b} = g^{\mathsf{K}_\mathbf{x} \cdot \Gamma_{i,b}}$$

Symmetric-key Encryption Scheme from DDH. Our garbling scheme assumes the existence of a symmetric-key encryption scheme $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ that is IND-CPA secure under the DDH assumption and the ciphertext's length is $O(|m| + \lambda)$ where $|m|$ is the length of the plaintext and

$\mathsf{Gen}(1^\lambda, |\mathsf{inp}|)$:

- Sample labels $\mathsf{L}_{x=b} \xleftarrow{\$} \{0,1\}^\lambda$ for each input wire x and $b \in \{0,1\}$.
- Select a prime-order DDH group $\mathbb{G}_q$ with security parameter λ, and a generator g of the group.
- Return $\mathsf{sk} = (\mathsf{L}, \mathbb{G}_q, g)$.

$\mathsf{Encode}(\mathsf{sk} = (\mathsf{L}, \_, \_), \mathsf{inp})$:

- Return $\widetilde{\mathsf{inp}}$ where $\widetilde{\mathsf{inp}}[i] \leftarrow \mathsf{L}_{x=\mathsf{inp}[i]}$ for each i-th input wire x.

$\mathsf{Garble}(\mathsf{sk} = (\mathsf{L}, \mathbb{G}_q, g), \mathcal{C})$:

- Sample $\mathsf{L}_{x=b} \xleftarrow{\$} \{0,1\}^\lambda$ for every non-input independent wire x and $b \in \{0,1\}$. Sample $\mathsf{K}_{\mathbf{x}} \xleftarrow{\$} \mathbb{Z}_q^*$ for each cable $\mathbf{x}^W$. Sample $\Gamma_{i,b} \xleftarrow{\$} \mathbb{Z}_q^*$ for every $i \in [W_{\max}]$ and $b \in \{0,1\}$, where $W_{\max}$ is the maximum width of any cable. Compute the labels $\mathsf{L}_{x_i=b} = g^{\mathsf{K}_{\mathbf{x}} \cdot \Gamma_{i,b}}$ for every subwire x_i of every cable $\mathbf{x}^W$ and $b \in \{0,1\}$.
- For each gate gid, compute the GC material $\widetilde{\mathsf{Gates}}[\mathsf{gid}]$ based on its gate type:
  - **Boolean** $z \leftarrow f(x,y)$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left\{\mathsf{Enc}_{\mathsf{L}_{x=a}, \mathsf{L}_{y=b}}\left(\mathsf{L}_{z=f(a,b)}\right) \mid a, b \in \{0,1\}\right\}$$
  - **Group** $y_i \xleftarrow{\mathbf{grp}} x$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \{\mathsf{Enc}_{\mathsf{L}_{x=a}}\left(\mathsf{L}_{y_i=a}\right) \mid a \in \{0,1\}\}$$
  - **Ungroup** $y \xleftarrow{\mathbf{ungrp}} x_i$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left\{\mathsf{Enc}_{\mathsf{L}_{x_i=a}}\left(\mathsf{L}_{y=a}\right) \mid a \in \{0,1\}\right\}$$
  - **Switch** $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \mathsf{Enc}_{\mathsf{L}_{c=0}}\left(\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}\right)$$
- Let $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{Enc}_{\mathsf{L}_{y=0}}(0)$ for each i-th output wire y.
- Return $\widetilde{\mathcal{C}} = \left(\mathcal{C}, \mathbb{G}_q, \widetilde{\mathsf{Gates}}, \widetilde{\mathsf{Out}}\right)$

Figure 3: Our garbling scheme based on DDH (continued in Figure 4).

$\mathsf{Eval}(\widetilde{\mathcal{C}} = \left(\mathcal{C}, \mathbb{G}_q, \widetilde{\mathsf{Gates}}, \widetilde{\mathsf{Out}}\right), \widetilde{\mathsf{inp}})$:

- Let $\mathsf{L}_x \leftarrow \widetilde{\mathsf{inp}}[i]$ for each i-th input wire x. Evaluate every gate gid in the topological order specified by the evaluation rules in Table 4:
  - **Boolean** $z \leftarrow f(x, y)$**:**
    For each entry $R \in \widetilde{\mathsf{Gates}}[\mathsf{gid}]$, let $m \leftarrow \mathsf{Dec}_{\mathsf{L}_x, \mathsf{L}_y}(R)$ and set $\mathsf{L}_z \leftarrow m$ if $m \neq \perp$.
  - **Group** $y_i \xleftarrow{\mathbf{grp}} x$**:**
    For each entry $R \in \widetilde{\mathsf{Gates}}[\mathsf{gid}]$, let $m \leftarrow \mathsf{Dec}_{\mathsf{L}_x}(R)$ and set $\mathsf{L}_{y_i} \leftarrow m$ if $m \neq \perp$.
  - **Ungroup** $y \xleftarrow{\mathbf{ungrp}} x_i$**:**
    For each entry $R \in \widetilde{\mathsf{Gates}}[\mathsf{gid}]$, let $m \leftarrow \mathsf{Dec}_{\mathsf{L}_{x_i}}(R)$ and set $\mathsf{L}_y \leftarrow m$ if $m \neq \perp$.
  - **Switch** $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$:
    Let $m \leftarrow \mathsf{Dec}_{\mathsf{L}_c}\left(\widetilde{\mathsf{Gates}}[\mathsf{gid}]\right)$. Skip the gate if $m = \perp$. Otherwise, for each $i \in [W]$, set $\mathsf{L}_{y_i} \leftarrow (\mathsf{L}_{x_i})^m$ if L_{x_i} is known, and set $\mathsf{L}_{x_i} \leftarrow (\mathsf{L}_{y_i})^{m^{-1}}$ if L_{y_i} is known.†
- For each i-th output wire y, let $m \leftarrow \mathsf{Dec}_{\mathsf{L}_y}\left(\widetilde{\mathsf{Out}}[i]\right)$. If $m \neq \perp$, set $\mathsf{out}[i] \leftarrow 0$; else set $\mathsf{out}[i] \leftarrow 1$.
- Return out

† *Instead of computing the exponentiation repeatedly, the evaluator can keep track of an exponent pending to be applied, and perform modular multiplications on the exponent until reaching an Ungroup gate.*

Figure 4: Our garbling scheme based on DDH (continued from Figure 3).

λ is the security parameter. Moreover, we assume that Dec outputs $\perp$ with overwhelmingly high probability if the key is incorrect. Such an encryption scheme can be constructed directly from ElGamal [ElG85] by

- using the private key in ElGamal as the symmetric key for Π, and
- adding a λ-bit nonce to the plaintext m, and let Dec output $\perp$ if the decrypted nonce doesn't match.

In addition, we use the notation $\mathsf{Enc}_{k_1,k_2}(m)$ as a shorthand for the double encryption $\mathsf{Enc}_{k_1}(\mathsf{Enc}_{k_2}(m))$, and $\mathsf{Dec}_{k_1,k_2}(c)$ as a shorthand for the following operations:

- If $\mathsf{Dec}_{k_2}(c) = \perp$, then return $\perp$.
- Else return $\mathsf{Dec}_{k_1}(\mathsf{Dec}_{k_2}(c))$.

Full construction. We present our full garbling scheme in Figure 3 and Figure 4. Without loss of generality, we assume that all the input wires are independent wires.

On a high level, the Boolean, Group, Ungroup gates are garbled similar to Yao's garbled circuit [Yao86], except that we encode cables with our custom labels. For the Switch gates, we encrypt the quotient of the two cable keys under the zero label of the control wire. If the control wire is

zero, the evaluator can decrypt the quotient and hence learn the labels of one cable from the other, which matches our evaluation rules for the Switch gates in Table 4. Otherwise, the decryption fails, and the evaluator learns that the Switch gate is inactive. Since the circuit is well-formed, the evaluator can eventually compute the label corresponding to the unique value of each wire.

4.3 Analysis

Efficiency. Each gate or output wire of $\mathcal{C}$ adds an $O(1)$-row garbled truth table to the garbled circuit, with each row being an $O(\lambda)$-bit ciphertext string. Therefore, each gate or output wire incurs $O(\lambda)$ bits of communication, and the total communication cost of $\mathcal{C}$ is $O(\lambda \cdot |\mathcal{C}|)$ bits. In comparison, prior works [HKO23, Hea24] require $O(\lambda \cdot W)$ bits to garble a circuitry equivalent to our Switch gate, where W is the width of the cables the Switch connects to.

Correctness. Given the input labels to each gate, the evaluator can identify the correct row to decrypt in garbled truth table and learn the output wire's label with overwhelmingly high probability. It is straightforward to check that the truth tables agree to the evaluation rules in Table 4 for each gate. By dynamic well-formedness (Definition 3.1), the evaluator learns a unique label L_x that matches the value of each wire x, and can decode the value of each i-th output wire y by checking whether $\widetilde{\mathsf{Out}}[i]$ can be decrypted successfully with the key L_y.

Security. We prove the security of the garbling scheme assuming DDH. First, we introduce a simple lemma.

Lemma 4.3. *Let g be a generator of a DDH group $\mathbb{G}_q$ with prime order q and security parameter λ, and $k, \gamma_0, \gamma_1 \xleftarrow{\$} \mathbb{Z}_q^*$. Then, the ensemble of tuple $\overline{T} = \left(g,\, g^{\gamma_0},\, g^{\gamma_1},\, g^{k\cdot\gamma_0},\, g^{k\cdot\gamma_1}\right)$ is computationally indistinguishable from the ensemble $\overline{T}' = (g,\, g^{\gamma_0},\, g^{\gamma_1},\, R_0,\, R_1)$, where $R_0, R_1 \xleftarrow{\$} \mathbb{G}_q$.*

Proof. By the DDH assumption, $\left(g,\, g^k,\, g^\gamma,\, g^{k\cdot\gamma}\right)$ is computationally indistinguishable from $\left(g,\, g^k,\, g^\gamma,\, R\right)$, for uniformly random γ and R. Therefore,

$$
\begin{aligned}
\left\{\overline{T}\right\}_{\lambda\in\mathbb{N}} &= \left\{\left(g,\, g^k,\, g^{\gamma_0},\, (g^{\gamma_0})^{\gamma_r},\, g^{k\cdot\gamma_0},\, \left(g^{k\cdot\gamma_0}\right)^{\gamma_r}\right) \mid \gamma_r \xleftarrow{\$} \mathbb{Z}_q^*\right\}_{\lambda\in\mathbb{N}} \\
&\overset{c}{\approx} \left\{\left(g,\, g^k,\, g^{\gamma_0},\, (g^{\gamma_0})^{\gamma_r},\, R_0,\, \left(g^{k\cdot\gamma_0}\right)^{\gamma_r} \mid \gamma_r \xleftarrow{\$} \mathbb{Z}_q^*\right)\right\}_{\lambda\in\mathbb{N}} \\
&= \left\{\left(g,\, g^{\gamma_0},\, g^{\gamma_1},\, R_0,\, g^{k\cdot\gamma_1}\right)\right\}_{\lambda\in\mathbb{N}} \\
&\overset{c}{\approx} \left\{(g,\, g^{\gamma_0},\, g^{\gamma_1},\, R_0,\, R_1)\right\}_{\lambda\in\mathbb{N}} = \left\{\overline{T}'\right\}_{\lambda\in\mathbb{N}}
\end{aligned}
$$

□

Now, we present the main theorem. At a high level, we follow the proof of Yao's garbled circuit [Yao86, LP09], replacing the labels of wires with random labels in the order of evaluation, and apply additional techniques to handle Switch gates and subwire labels.

Theorem 4.4 (Security of the garbling scheme in the plain model)**.** *Assuming DDH, The construction in Figure 3 and 4 is a secure garbling scheme for the family of strictly well-formed SIMD tri-state circuit w.r.t. the leakage function* $\mathsf{controls}(\cdot,\cdot)$ *as defined below: given a well-formed SIMD tri-state circuit $\mathcal{C}$ and input* inp, $\mathsf{controls}(\mathcal{C}, \mathsf{inp})$ *outputs the values on all the control wires of the Switch gates when evaluating $\mathcal{C}$ over* inp.

$\mathsf{Sim}(1^\lambda, \mathsf{ctrl}, \mathsf{out}, \mathcal{C})$:

- Select a prime-order DDH group $\mathbb{G}_q$ with security parameter λ, and a generator g of the group, same as Gen.
- Sample $\mathsf{L}_x, \mathsf{L}'_x \xleftarrow{\$} \{0,1\}^\lambda$ for every independent wire x. Let $\widetilde{\mathsf{inp}}[i] \leftarrow \mathsf{L}_{x_{\text{in}}}$ for each i-th input wire x_{in}.
- For every union of cables connected by active Switch gates, sample union-wise labels $\mathsf{L}_{U_i}, \mathsf{L}'_{U_i} \xleftarrow{\$} \mathbb{G}_q$ for every offset $i \in [W]$, where W is the width of each cable in the union. For every cable, sample $\mathsf{K}_{\mathbf{x}} \xleftarrow{\$} \mathbb{Z}_q^*$, and compute its subwires' labels as $\mathsf{L}_{x_i} = (\mathsf{L}_{U_i})^{\mathsf{K}_{\mathbf{x}}}$ and $\mathsf{L}'_{x_i} = \left(\mathsf{L}'_{U_i}\right)^{\mathsf{K}_{\mathbf{x}}}$.
- For each gate gid, compute the GC material $\widetilde{\mathsf{Gates}}[\mathsf{gid}]$ based on its gate type:
  - **Boolean** $z \leftarrow f(x,y)$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left\{\mathsf{Enc}_{l_x,l_y}(\mathsf{L}_z) \mid l_x \in \{\mathsf{L}_x, \mathsf{L}'_x\}, l_y \in \{\mathsf{L}_y, \mathsf{L}'_y\}\right\}$$
  - **Group** $y_i \xleftarrow{\mathbf{grp}} x$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left\{\mathsf{Enc}_l(\mathsf{L}_{y_i}) \mid l \in \{\mathsf{L}_x, \mathsf{L}'_x\}\right\}$$
  - **Ungroup** $y \xleftarrow{\mathbf{ungrp}} x_i$**:**
  $$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left\{\mathsf{Enc}_l(\mathsf{L}_y) \mid l \in \{\mathsf{L}_{x_i}, \mathsf{L}'_{x_i}\}\right\}$$
  - **Switch** $\mathbf{x}^W \overset{c}{\text{---}/\text{---}} \mathbf{y}^W$**:**
  $$\text{If the value of } c \text{ is 0 in } \mathsf{ctrl}, \text{ then } \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \mathsf{Enc}_{\mathsf{L}_c}\left(\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}\right)$$
  $$\text{Else, } \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \mathsf{Enc}_{\mathsf{L}'_c}(0)$$
- For each i-th output wire y, if its value is 0 in out, $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{Enc}_{\mathsf{L}_y}(0)$, else, $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{Enc}_{\mathsf{L}'_y}(0)$.
- Return $\left(\mathcal{C}, \mathbb{G}_q, \widetilde{\mathsf{Gates}}, \widetilde{\mathsf{Out}}\right), \widetilde{\mathsf{inp}}$.

Figure 5: The simulator Sim for the ideal experiment.

Proof. Since $\mathcal{C}$ is well-formed, every wire x must be set to a unique value val_x given the circuit's input in the Real experiment. From now on, we call $\mathsf{L}_{x=\mathsf{val}_x}$ the *active label* of wire x, and simply denote it as L_x, and the other label the *inactive label* of x, denoted as L'_x. Moreover, we define a *union* of cables to be a set of cables connected through active Switch gates. We construct a simulator Sim that outputs the ideal view, as shown in Figure 5, and we construct the following hybrids to show that the ideal view is computationally indistinguishable from the real view with respect to the security parameter λ:

$\mathbf{Hyb}_0$: We first perform a refactoring of the real view and obtain Hybrid $\mathbf{Hyb}_0$: For each union of cables connected by active Switch gates, we sample a union-wise key $\mathsf{K}_U \xleftarrow{\$} \mathbb{Z}_q^*$, and compute union-wise labels $\mathsf{L}_{U_i=b} = g^{\mathsf{K}_U \cdot \Gamma_{i,b}}$ for every $i \in [W]$ and $b \in \{0,1\}$, where W is the width of each cable in the union. For every cable $\mathbf{x}^W$ in the union, sample its key $\mathsf{K}_{\mathbf{x}} \xleftarrow{\$} \mathbb{Z}_q^*$, and compute its subwires' labels as $\mathsf{L}_{x_i=b} = (\mathsf{L}_{U_i=b})^{\mathsf{K}_{\mathbf{x}}}$. Furthermore, when garbling every inactive Switch gate, we replace $\mathsf{K}_{\mathbf{x}}^{-1} \cdot \mathsf{K}_{\mathbf{y}}$ with $(\mathsf{K}_{\mathbf{x}} \cdot \mathsf{K}_U)^{-1} \cdot (\mathsf{K}_{\mathbf{y}} \cdot \mathsf{K}_V)$, where U and V are the unions $\mathbf{x}^W$ and $\mathbf{y}^W$ belong to respectively. Intuitively, the refactoring allows us to make the cable's key public, while only keeping the union-wise keys secret.

$\mathbf{Hyb}_1, \ldots, \mathbf{Hyb}_{2n}$: We construct a series of hybrids $\mathbf{Hyb}_1, \cdots, \mathbf{Hyb}_{2n}$, where n is the number of wires in $\mathcal{C}$. By Definition 3.4, there is an ordering of all the wires such that each wire is set after all its dependencies. Let x denote the τ-th wire in the ordering. Then each hybrid differs from the previous hybrid as follows:

- $\mathbf{Hyb}_{2\tau-2} \to \mathbf{Hyb}_{2\tau-1}$: Suppose that x is a subwire at offset i of the cable $\mathbf{x}^W$, and U is the union containing $\mathbf{x}^W$. Then, we randomly sample the union-wise label $\mathsf{L}_{U_i=0}$ and $\mathsf{L}_{U_i=1}$, rather than computing them from $g^{\mathsf{K}_U \cdot \Gamma_{i,0}}$ and $g^{\mathsf{K}_U \cdot \Gamma_{i,1}}$. Note that the labels of the i-th subwire of every cable in the union will change correspondingly.

- $\mathbf{Hyb}_{2\tau-1} \to \mathbf{Hyb}_{2\tau}$: For each Boolean, Group, Ungroup gate with input wire x, and each row in the garbled gate encrypted under the inactive label L'_x, we replace the row's plaintext (i.e., the inactive label of the output wire y) with the active label L_y of the output wire. Moreover, for each inactive Switch gate with control wire x, we replace the garbled material with $\mathsf{Enc}_{\mathsf{L}'_x}\left(0^\lambda\right)$.

Next, we show that each view is computationally indistinguishable from the previous one.

$\mathbf{Hyb}_0 = \mathbf{Real}$. Compared to **Real**, in $\mathbf{Hyb}_0$ we divide the key $\mathsf{K}_{\mathbf{x}}$ of every cable $\mathbf{x}^W$ by a union-wise key. Notice that for each active Switch gate, the quotient of its two cables' keys do not change, as the two cables belong to the same union. Thus, the distribution of all the labels and garbled materials remain the same.

$\mathbf{Hyb}_{2\tau-2} \overset{c}{\approx} \mathbf{Hyb}_{2\tau-1}$. We only consider the case when x is a subwire and the union-wise labels $\mathsf{L}_{U_i=0}$ and $\mathsf{L}_{U_i=1}$ have not yet been replaced with a random group element, since otherwise the views are the same.

By Lemma 4.3, the tuple $\left(g, g^k, g^{\gamma_0}, g^{\gamma_1}, g^{k\cdot\gamma_0}, g^{k\cdot\gamma_1}\right)$ is computationally indistinguishable from $\left(g, g^k, g^{\gamma_0}, g^{\gamma_1}, R_0, R_1\right)$, where $k, \gamma_0, \gamma_1 \xleftarrow{\$} \mathbb{Z}_q^*$ and $R_0, R_1 \xleftarrow{\$} \mathbb{G}_q$. We construct a P.P.T algorithm $\mathcal{B}_{\mathcal{C},\tau,i}$ that takes one of these tuples as input, and we show that the algorithm's output distribution is identical to $\mathbf{Hyb}_{2\tau-2}$ if the input is $\left(g, g^k, g^{\gamma_0}, g^{\gamma_1}, g^{k\cdot\gamma_0}, g^{k\cdot\gamma_1}\right)$ and identical to $\mathbf{Hyb}_{2\tau-1}$ if the input is $\left(g, g^k, g^{\gamma_0}, g^{\gamma_1}, R_0, R_1\right)$.

$\mathcal{B}_{\mathcal{C},\tau,i}(g, g^k, g^{\gamma_0}, g^{\gamma_1}, R'_0, R'_1)$:

1. Sample the global keys $\Gamma_{j,b}$ for every $j \neq i$ and $b \in \{0,1\}$.
2. Sample the union-wise key K_V for every union $V \neq U$.
3. For each union V and pair $(j,b) \in [W_{\max}] \times \{0,1\}$, if the union-wise label $\mathsf{L}_{V_j=b}$ is not already replaced with a random group element in $\mathbf{Hyb}_{2\tau-2}$,

$$\text{then set } \mathsf{L}_{V_j=b} \leftarrow \begin{cases} R'_b & \text{if } V = U \text{and } j = i \\ \left(g^k\right)^{\Gamma_{j,b}} & \text{if } V = U \text{and } j \neq i \\ \left(g^{\gamma_b}\right)^{\mathsf{K}_V} & \text{if } V \neq U \text{and } j = i \\ g^{\mathsf{K}_V \cdot \Gamma_{j,b}} & \text{if } V \neq U \text{and } j \neq i \end{cases}$$

4. Run the simulator of $\mathbf{Hyb}_{2\tau-2}$ using the union-wise keys and labels in steps 2 and 3.

A crucial observation is that $\mathcal{B}_{\mathcal{C},\tau,i}$ does not sample the union-wise key K_U, and we show that indeed K_U is not needed for simulating $\mathbf{Hyb}_{2\tau-2}$ or $\mathbf{Hyb}_{2\tau-1}$. First, we prove that x depends on the control wire c of an inactive Switch gate if the gate is connected to a cable in U, where U is the union that contains $\mathbf{x}^W$.

Let $\mathbf{y}_0^W$ be an alias of $\mathbf{x}^W$, and let $\mathbf{y}_m^W$ be the cable in U connected to the inactive Switch. There must exist a chain of active Switch gates

$$(\mathbf{x}^W =)\mathbf{y}_0^W \overset{c_1}{\text{---}} \mathbf{y}_1^W, \quad \mathbf{y}_1^W \overset{c_2}{\text{---}} \mathbf{y}_2^W, \ \cdots, \ \mathbf{y}_{m-1}^W \overset{c_m}{\text{---}} \mathbf{y}_m^W$$

By Definition 3.3, every subwire of $\mathbf{y}_m^W$ depends on the control wire c of the inactive Switch gate, and moreover, for every $t \in [m]$, if the i-th subwire of $\mathbf{y}_t^W$ depends on c, then the i-th subwire of $\mathbf{y}_{t-1}^W$ also depends on c. By transitivity, we conclude that x depends on c.

As a result, the control wire c is ordered before x, which means we must have replaced the garbled material of the inactive Switch with an encryption of zero in a previous hybrid. Since the only place where union-wise keys may be used is the inactive Switches connected to a cable in the union, we conclude that K_U is not needed.

Now, it is straightforward to check that (1) If $R'_0 = g^{k \cdot \gamma_0}$ and $R'_1 = g^{k \cdot \gamma_1}$, then the output distribution of $\mathcal{B}_{\mathcal{C},\tau,i}$ is identical to $\mathbf{Hyb}_{2\tau-2}$, and (2) If $R'_0 = R_0$ and $R'_1 = R_1$, then the output distribution is identical to $\mathbf{Hyb}_{2\tau-1}$. Notice that the hidden exponent k corresponds to the union-wise key K_U, the hidden exponents γ_0 and γ_1 correspond to the global keys $\Gamma_{i,0}$ and $\Gamma_{i,1}$, and R'_0 and R'_1 corresponds to the labels of x. By the DDH assumption, we conclude that $\mathbf{Hyb}_{2\tau-2} \overset{c}{\approx} \mathbf{Hyb}_{2\tau-1}$.

$\mathbf{Hyb}_{2\tau-1} \overset{c}{\approx} \mathbf{Hyb}_{2\tau}$. To obtain $\mathbf{Hyb}_{2\tau}$, we need to update the garbled truth table rows encrypted under wire x's inactive label L'_x, and we show that the new hybrid remains computationally indistinguishable.

Note that in $\mathbf{Hyb}_{2\tau-1}$, the inactive label L'_x is sampled independently. Moreover, we show that it is used only as encryption keys when generating the view. First, L'_x does not appear in $\widetilde{\mathsf{inp}}$ because it is the inactive label. Second, suppose that L'_x is in the plaintext encrypted in a Boolean, Group, or Ungroup gate. Then x must be the gate's output wire and hence ordered after all the gate's input wire(s) by our dependency definition. Furthermore, since we are using Yao's garbling, the inactive label L'_x must be encrypted under the inactive label of at least one of the input wires. This means we must have replaced L'_x with the active label L_x in a previous hybrid, which leads to a contradiction.

As shown in the proof of the standard Yao's garbled circuit [Yao86,LP09], the double encryption ensures that the updated garbled truth table rows remain computationally indistinguishable from the original. In addition, all the other parts of the view are identical in both hybrids. Therefore, we conclude that $\mathbf{Hyb}_{2\tau-1} \overset{c}{\approx} \mathbf{Hyb}_{2\tau}$.

Ideal $= \mathbf{Hyb}_{2n}$**.** At this stage, we have removed all the usage of union-wise keys and made all the union-wise labels independently sampled. Moreover, the zero and one labels become symmetric for every wire except the control and output wires. Therefore, the simulator Sim only needs to know the values of the output and the control wires, and **Ideal** is equivalent to $\mathbf{Hyb}_{2n}$.

By the chain of hybrids, we have shown that the real view is computationally indistinguishable from the ideal view, and the garbling scheme is secure. □

5 From Garbled SIMD Tri-State Circuit to Garbled RAM

Heath et al. [HKO23] showed how to construct a garbling scheme for RAM from a garbling scheme for tri-state circuits. At a high level, given a RAM program, we first convert it to an *oblivious* SIMD tri-state circuit that computes the same function as the RAM, where obliviousness will be defined shortly. We then garble the resulting oblivious SIMD tri-state circuit.

We will use the same transformation as Heath et al. [HKO23] to get our final result for garbled RAM, except that we replace the SIMD operations in the tri-state circuit with SIMD gates, and garble them using our new SIMD garbling techniques. For completeness, in this section, we review how to eventually get a garbled RAM given a garbling scheme for SIMD tri-state circuits.

5.1 Oblivious Simulation of RAM in SIMD Tri-State Circuit

Controls. Henceforth, let $\mathsf{controls}(\mathcal{C}, \mathsf{inp})$ be the function that outputs the values on all control wires for SIMD Switch gates when evaluating a well-formed SIMD tri-state circuit $\mathcal{C}$ on input inp.

Definition 5.1 (Oblivious simulation of RAM in SIMD tri-state circuit)**.** Given a deterministic RAM program $P : \{0,1\}^{|\mathsf{inp}|} \to \{0,1\}^*$, we say that P is δ-obliviously simulated by a well-formed SIMD tri-state circuit $\mathcal{C}$ if there exists an efficient simulator $\mathsf{Sim}_{\mathrm{ctrl}}$, such that for any inp, the following two distributions have statistical distance at most δ:

- **Real:** Sample $\mathsf{rinp} \xleftarrow{\$} \{0,1\}^*$, and output $\mathcal{C}(\mathsf{inp}||\mathsf{rinp}), \mathsf{controls}(\mathcal{C}, \mathsf{inp}||\mathsf{rinp})$.[3]
- **Ideal:** Output $P(\mathsf{inp}), \mathsf{Sim}_{\mathrm{ctrl}}()$;

Theorem 5.2 (Oblivious tri-state circuit simulation of RAM [HKO23])**.** *Let* $\mathsf{negl}(\cdot)$ *be a suitable negligible function. For any RAM program* P *with maximum space requirement* N*, maximum runtime* $T = \mathsf{poly}(N)$*, and word size* W*, there exists a (SIMD) tri-state circuit that* $\mathsf{negl}(N)$*-obliviously simulates*[4] P*. Furthermore, the circuit consists of:*

1. **Data stacks:** $O(T/N \cdot 2^L)$ stacks each supporting $N/2^L$ accesses on data items of size $O(W + \log N)$ for $L = 0, 1, \ldots, \log N$;

[3] In the original tri-state circuits [HKO23], rinp is sampled from a distribution $\mathcal{D}$ which contains multiplication triples for emulating AND gates. Since we already have Boolean gates in the model, we can simply sample rinp from the uniform distribution.

[4] Without loss of generality, we may assume that $N \geq \lambda$ since we are studying the asymptotic behavior as N goes to infinity. In this case, negligibly small in N implies negligibly small in the security parameter λ.

2. **Metadata stacks:** $O(T/N \cdot 2^L \cdot \log N)$ stacks each supporting $N/2^L$ accesses on data items of size $\log N$ for $L = 0, 1, \ldots, \log N$,

3. **All other gates:** $O(T \cdot (W \cdot \log N + \log^3 N) \cdot \omega(1))$ number of Boolean gates.

5.2 Garbled RAM Construction

Given a garbling scheme denoted $(\mathsf{TSC.Gen}, \mathsf{TSC.Encode}, \mathsf{TSC.Garble}, \mathsf{TSC.Eval})$ for SIMD tri-state circuits, we construct a garbled RAM scheme as follows:

- $\mathsf{RAM.Gen}(1^\lambda, \mathsf{params} = (N, T))$: compute $|\mathsf{rinp}|$ as a function of N and T, let $\mathsf{sk} \leftarrow \mathsf{TSC.Gen}(1^\lambda, |\mathsf{inp}| + |\mathsf{rinp}|)$, and output sk.
- $\mathsf{RAM.Encode}(\mathsf{sk}, \mathsf{inp})$: output $\mathsf{TSC.Encode}(\mathsf{sk}, \mathsf{inp})$.
- $\mathsf{RAM.Garble}(\mathsf{sk}, P)$:
  1. Let $\mathcal{C}$ be a SIMD tri-state circuit that obliviously simulates P with input length $|\mathsf{inp}| + |\mathsf{rinp}|$, and let $\widetilde{\mathcal{C}} \leftarrow \mathsf{TSC.Garble}(\mathsf{sk}, \mathcal{C})$.
  2. Sample rinp uniformly randomly, and let $\widetilde{\mathsf{rinp}} \xleftarrow{\$} \mathsf{TSC.Encode}(\mathsf{sk}, \mathsf{rinp})$.
  3. Output $\widetilde{P} := (\widetilde{C}, \widetilde{\mathsf{rinp}})$.
- $\mathsf{RAM.Eval}(\widetilde{P}, \widetilde{\mathsf{inp}})$: Output $\mathsf{TSC.Eval}(\widetilde{C}, \widetilde{\mathsf{inp}} \| \widetilde{\mathsf{rinp}})$.

Heath et al. [HKO23] proved the following theorem.

Theorem 5.3 (Security of Garbled RAM [HKO23])**.** *Suppose that* $(\mathsf{TSC.Gen}, \mathsf{TSC.Encode}, \mathsf{TSC.Garble}, \mathsf{TSC.Eval})$ *is a secure garbling scheme for tri-state circuits w.r.t. the leakage function* $\mathsf{controls}(\cdot, \cdot)$*. Then, the above garbled RAM construction is secure.*

5.3 GRAM Cost Analysis

We now calculate the per-instruction cost of PicoGRAM assuming $T = N$. In Section A.1, we generalize the result to both $T < N$ and $T > N$.

Data stacks. By Theorem 5.2 and Theorem B.18, the total cost of data stacks is

$$\sum_{\ell=0}^{\log N} 2^L \cdot O(N/2^L \cdot (W + \log N + \log(N/2^L))) = O(W \cdot \log N + \log^2 N)$$

Metadata stacks. Similarly, by Theorem 5.2 and Theorem B.18, the total cost of metadata stacks is

$$\sum_{\ell=0}^{\log N} 2^L \cdot \log N \cdot O(N/2^L \cdot (\log N + \log(N/2^L))) = O(\log^3 N)$$

All other gates. By Theorem 5.2, the total cost of all other gates is

$$O(\lambda \cdot N \cdot (W \cdot \log N + \log^3 N) \cdot \omega(1))$$

Summing up the results, we get the following theorem:

Theorem 5.4. *Assume the hardness of DDH. There exists a garbled RAM scheme with an amortized communication cost of $O(\lambda \cdot (W \cdot \log N + \log^3 N) \cdot \omega(1))$ bits per instruction, where N is the RAM's space, W is the word width, λ is the security parameter, and $\omega(1)$ is an arbitrarily small super-constant factor in N.*

We can slightly improve the cost in Theorem 5.4 to match that of Theorem 1.1 with some extra tricks, as is described in Section A.2.

6 Concretely Efficient Stack

Since the stack is a repeatedly used building block in the SIMD tri-state circuit that obliviously simulates a RAM, it is also the performance bottleneck of the garbled RAM. In this section, we show how to realize a concretely efficient stack in the SIMD tri-state circuit model. For $N = 2^{20}$, our new stack achieves a 29-fold reduction in bandwidth relative to prior stack constructions [PLS23, HKO23, Hea23] with the use of our new SIMD garbling, and a 4-fold reduction even without using our SIMD techniques.

6.1 Syntax: Compaction Stack and Distribution Stack

In both the tri-state GRAM [HKO23] and our PicoGRAM, the stack needs to support two-way communication between parent and child nodes. We refer to the component that passes data from the parent to the child as the "compaction stack," and its counterpart as the "distribution stack" (also called the "co-stack" and "stack" in prior work [HKO23]).

Definition 6.1 (Compaction stack and distribution stack)**.** A compaction stack takes as input t control wires $c_1, c_2, \ldots, c_t$ and $t \cdot W$ data wires $x_{\tau,i}$ for $\tau \in [t]$ and $i \in [W]$. Let $t' = t - \sum_{\tau \in [t]} c_\tau$. The compaction stack outputs $t' \cdot W$ data wires $y_{\tau',i}$ for $\tau' \in [t']$ and $i \in [W]$, and implements the following state transitions:

$$c_1, \ldots, c_{\tau-1} \neq \mathcal{Z} \text{ and } c_\tau = 0 \;\Rightarrow\; y_{\tau',i} \leftarrow x_{\tau,i} \quad \text{where } \tau' = \tau - \sum_{j=1}^{\tau-1} c_j$$

A distribution stack has the same interface as the compaction stack, except that it treats $y_{\tau',i}$ as input and $x_{\tau,i}$ as output. Namely, the distribution stack implements the following state transitions:

$$c_1, \ldots, c_{\tau-1} \neq \mathcal{Z} \text{ and } c_\tau = 0 \;\Rightarrow\; x_{\tau,i} \leftarrow y_{\tau',i} \quad \text{where } \tau' = \tau - \sum_{j=1}^{\tau-1} c_j$$

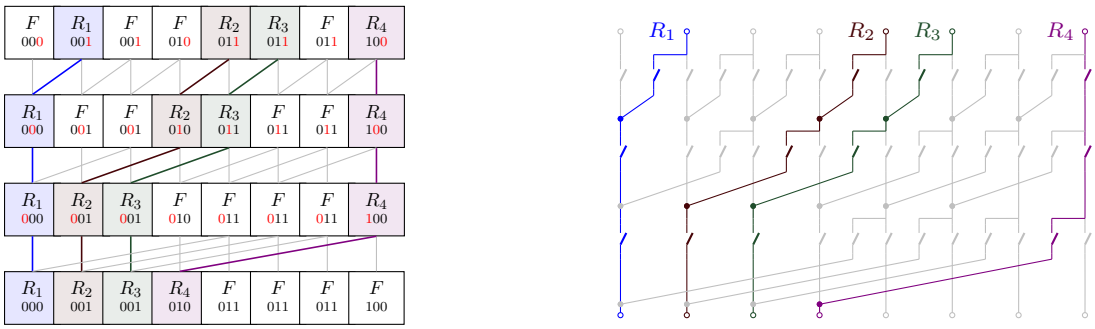
As mentioned in Section 3.2, any well-formed SIMD tri-state circuit supports incremental evaluation when inputs arrive gradually rather than all at once. The above syntax implies that once $\tau < t$ steps of inputs have arrived, the compaction stack can be eagerly evaluated, compacting all data elements that have arrived with flag 0. A similar observation applies to the distribution stack.

6.2 Stack Constructions from Oblivious Compaction

Intuition. Our stack construction builds on an oblivious compaction algorithm by Goodrich [Goo11]. The algorithm processes an array of t inputs (either real or filler) and compacts all the real elements to the front of the array via a compaction network (Figure 6a) of depth $d = \log t$. Consider

a real element e at input position τ. The goal is to route e to position $\tau - S_\tau$ in the output array, where S_τ is the number of input fillers preceding e. To achieve this, the algorithm shifts e forward by $b_\ell \cdot 2^{\ell-1}$ at each level ℓ, where b_ℓ is the ℓ-th bit of S_τ counting from the least significant bit. Goodrich [Goo11] showed that this routing scheme ensures no collisions in the network.

In our setting, real elements represent real stack calls and fillers represent fake calls. Since ORAM already masks the call types, we do not need to hide which calls are real. Meanwhile, we aim for incremental execution: each input element should be routed through the compaction network before the next input arrives. Goodrich's compaction network [Goo11] satisfies this requirement, as the routing schedule for each element depends only on the number of fillers preceding it.



(a) Goodrich's compaction network

(b) Adapted tri-state circuit

Figure 6: Goodrich's compaction algorithm routes elements through a compaction network (Figure 6a). An element at level ℓ can be routed either directly downward or to the left by $2^{\ell-1}$ offsets, depending on the ℓ-th bit in the binary representation of S_τ, where S_τ is the number of fillers among the first $\tau - 1$ input elements. In Figure 6b, we adapt the compaction network to a tri-state circuit, using Switch gates to control the routing directions. For clarity, we omit the control wires of the Switch gates.

To adapt the compaction algorithm to a SIMD tri-state circuit, we replace each cell with a cable and each edge with a Switch gate. The Switch controls which cable on the next level receives the data. A graphical illustration is provided in Figure 6b.

SIMD stack for data efficient routing. Next, we formally construct a core building block called *SIMD stack*, which assumes that the data wires are already grouped as cables. Assuming that the EagerPrefixSum circuit correctly sets the control wires, the SIMD stack implements bidirectional routing of data through the oblivious compaction network [Goo11]. Figure 8 illustrates our SIMD stack when $t = 4$. We defer formal analysis of the SIMD stack to Section B.3.

Eager evaluation of control wires. The remaining challenge is how to set the control wires for the Switch gates. A naïve approach is to run a counter that outputs the binary representation $\overline{s_{1,\tau} \cdots s_{d,\tau}}$ of $S_\tau = \sum_{i=1}^{i-1} c_\tau$ at every timestep τ. While this approach yields a stack that is both correct and strictly well-formed in a standalone setting, when the stack is plugged into the SIMD tristate circuit that implements the RAM, this bigger circuit does not satisfy strict well-formedness. This is because when the stack is part of the RAM's SIMD tristate circuit, its inputs do not arrive all at the same time — later inputs to the stack can depend on the earlier outputs of the stack, thus creating circular dependencies. Specifically, to achieve strict well-formedness, any wire set at timestep τ cannot depend on the input wire at timestep $\tau' > \tau$. Consider the example of Figure 8. Suppose that the stack receives a real element at timestep $\tau = 1$. Since $s_{1,1} = s_{1,2} = 0$, every subwire of $\mathbf{u}^W_{1,1}$, $\mathbf{u}^W_{2,1}$, $\mathbf{u}^W_{3,1}$ depends on the control wire $\neg s_{2,3}$ by our definition of dependency.

SIMD Stack

Notation: $\mathbf{x}_1^W, \ldots, \mathbf{x}_t^W \xleftarrow[\text{SIMD stack}]{c_1, \ldots, c_t} \mathbf{y}_1^W, \ldots, \mathbf{y}_t^W$

Parameter requirement: $t = 2^d$.

Construction:

Define cables $\mathbf{u}_{\ell,\tau}^W$ for $\ell \in [d+1]$. Let $\mathbf{u}_{1,\tau}^W$ and $\mathbf{u}_{d+1,\tau}^W$ be aliases of $\mathbf{x}_\tau^W$ and $\mathbf{y}_\tau^W$.

$$\begin{pmatrix} s_{1,1} & \cdots & s_{1,\tau} \\ \vdots & \ddots & \vdots \\ s_{d,1} & \cdots & s_{d,\tau} \end{pmatrix} \leftarrow \text{EagerPrefixSum}\,(c_1, \ldots, c_\tau)$$

$\mathbf{u}_{\ell,\tau}^W \xrightarrow{s_{\ell,\tau}} \mathbf{u}_{\ell+1,\tau}^W$ $\qquad \forall \ell \in [d], \tau \in [t]$

$\mathbf{u}_{\ell,\tau}^W \xrightarrow{\neg s_{\ell,\tau}} \mathbf{u}_{\ell+1,\tau-2^{\ell-1}}^W$ $\qquad \forall \ell \in [d],\ \tau \in (2^{\ell-1}, t]$

Figure 7: The SIMD stack circuit implements Goodrich's compaction network [Goo11], and calls the eager-prefix-sum building block to generate the routing schedule.

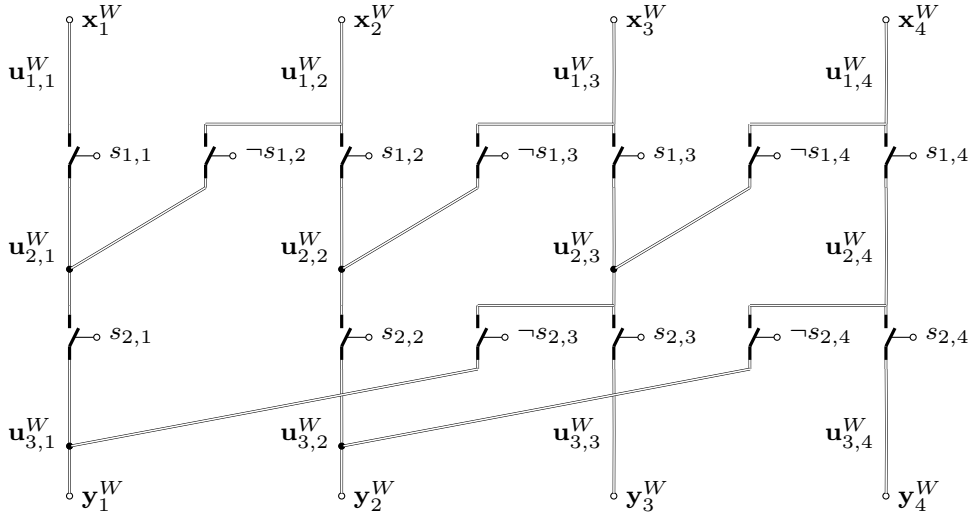


Figure 8: An example of SIMD stack in Figure 7 for $t = 4$. When $c_\tau = 0$, the cable $\mathbf{x}_\tau^W$ is dynamically connected to cable $\mathbf{y}_{\tau'}^W$, where $\tau' = \tau - \sum_{j=1}^{\tau-1} c_\tau$. The symbol $\neg s$ denotes the output wire of Boolean gate $\text{NAND}(s, s)$.

Eager-Prefix-Sum

Notation: $\begin{pmatrix} s_{1,1} & \cdots & s_{1,t} \\ \vdots & \ddots & \vdots \\ s_{d,1} & \cdots & s_{d,t} \end{pmatrix} \leftarrow \text{EagerPrefixSum}\,(c_1, \ldots, c_t)$

Parameter requirements: t is a power of two, and $d = \log t$.

Construction:

$z_{1,\tau}$ is an alias of c_τ	$\forall \tau \in [t]$
$s_{\ell,\tau} \leftarrow 0$	$\forall \ell \in [d], \tau \in [2^{\ell-1}]$
$s_{\ell,\tau+2^{\ell-1}} \leftarrow \text{XOR}\,(z_{\ell,\tau},\, s_{\ell,\tau})$	$\forall \ell \in [d], \tau \in [t-2^{\ell-1}]$
$a_{\ell,\tau} \leftarrow \text{OR}\,(z_{\ell,\tau},\, s_{\ell,\tau})$	$\forall \ell \in [d-1], \tau \in [t-2^{\ell-1}]$
$b_{\ell,\tau} \leftarrow \text{OR}\,\left(z_{\ell,\tau+2^{\ell-1}},\, \neg s_{\ell,\tau+2^{\ell-1}}\right)$	$\forall \ell \in [d-1], \tau \in [t-2^{\ell-1}]$
$z_{\ell+1,\tau} \overset{a_{\ell,\tau}}{\text{---}} 0$	$\forall \ell \in [d-1], \tau \in [t-2^{\ell-1}]$
$z_{\ell+1,\tau} \overset{\neg a_{\ell,\tau}}{\text{---}} b_{\ell,\tau}$	$\forall \ell \in [d-1], \tau \in [t-2^{\ell-1}]$

Figure 9: The eager-prefix-sum construction. We overload the symbol $x \overset{c}{\text{---}} y$ to denote three gates connected in series: $u_1 \overset{\text{grp}}{\longleftarrow} x$, $\mathbf{u}^1 \overset{c}{\text{---}} \mathbf{v}^1$, $y \overset{\text{ungrp}}{\longleftarrow} v_1$, where $\mathbf{u}^1$ and $\mathbf{v}^1$ are cables of width 1. Additionally, we introduce a constant wire 0 that always carries 0. Intuitively, $z_{\ell,\tau}$ is 0 iff there is a real element routed through the cable $\mathbf{u}^W_{\ell,\tau}$ in the SIMD stack, $a_{\ell,\tau}$ is zero iff there is an element routed directly downward from cable $\mathbf{u}^W_{\ell,\tau}$ to $\mathbf{u}^W_{\ell+1,\tau}$, and $b_{\ell,\tau}$ is zero iff there is an element routed leftward from cable $\mathbf{u}^W_{\ell,\tau+2^{\ell-1}}$ to $\mathbf{u}^W_{\ell+1,\tau}$. The last two lines essentially computes $z_{\ell+1,\tau} = \text{AND}(a_{\ell,\tau}, b_{\ell,\tau})$, but the use of Switch gates allows $z_{\ell+1,\tau}$ to be set to 0 eagerly when $a_{\ell,\tau} = 0$, i.e., when $\mathbf{u}^W_{\ell+1,\tau}$ receives a real element from $\mathbf{u}^W_{\ell,\tau}$.

However, $\neg s_{2,3}$ also depends on c_2 due to the Boolean circuit of the counter, which is input at timestep $\tau = 2$, resulting in a circular dependency.

To fix this circularity issue, we observe that it is possible to "predict" certain bits in future prefix sums. In the above example, since the first input control is $c_1 = 0$, we have $S_3 = \sum_{\tau=1}^{2} c_\tau \le 1$, and hence $s_{2,3}$ must equal 0. More generally, let $z_{\ell,\tau}$ denote whether there is a real element routed through the cable $\mathbf{u}^W_{\ell,\tau}$, where $z_{\ell,\tau} = 0$ means there is a real element routed through the cable. Then, as we show in Section B, $s_{\ell,\tau+2^{\ell-1}}$ can be set eagerly as the XOR of $z_{\ell,\tau}$ and $s_{\ell,\tau}$. We designed a SIMD tri-state circuit called *eager-prefix-sum* (Figure 9) that implements such eager evaluation behavior, and prove that it allows each subwire of a cable to be set only after all the control wires it depends on are set.

Compaction and distribution stacks. Finally, we obtain compaction and distribution stacks by encapsulating the SIMD stack with Group and Ungroup gates. To ensure well-formedness, we add t extra Switch gates in the compaction stack to filter dummy inputs, and pad the unused ports of the SIMD stack with constant input cables 0^W that carries zero.

Compaction and Distribution Stack

Parameter requirements: t is a power of two, and $t' = t - \sum_{\tau \in [t]} c_\tau$.
Construction:
Define $\mathbf{u}_1^W, \ldots, \mathbf{u}_t^W$, $\mathbf{v}_1^W, \ldots, \mathbf{v}_{t'}^W$, and a cable 0^W carrying zero values. Let

$$\mathbf{u}_1^W, \ldots, \mathbf{u}_t^W \xleftarrow[\text{SIMD stack}]{c_1, \ldots, c_t} \mathbf{v}_1^W, \ldots, \mathbf{v}_{t'}^W, \mathbf{0}^W, \ldots, \mathbf{0}^W$$

- For compaction stack where $x_{\tau,i}$ are inputs and $y_{\tau',i}$ are outputs:

$$\mathbf{r}_\tau^W \xleftarrow{c_\tau} \mathbf{u}_\tau^W \quad \text{for } \tau \in [t]$$

$$r_{\tau,i} \xleftarrow{\text{grp}} x_{\tau,i} \quad \text{for } \tau \in [t], i \in [W]$$

$$y_{\tau',i} \xleftarrow{\text{ungrp}} v_{\tau',i} \quad \text{for } \tau' \in [t'], i \in [W]$$

- For distribution stack where $y_{\tau',i}$ are inputs and $x_{\tau,i}$ are outputs:

$$v_{\tau',i} \xleftarrow{\text{grp}} y_{\tau',i} \quad \text{for } \tau' \in [t'], i \in [W]$$

$$x_{\tau,i} \xleftarrow{\text{ungrp}} u_{\tau,i} \quad \text{for } \tau \in [t], i \in [W]$$

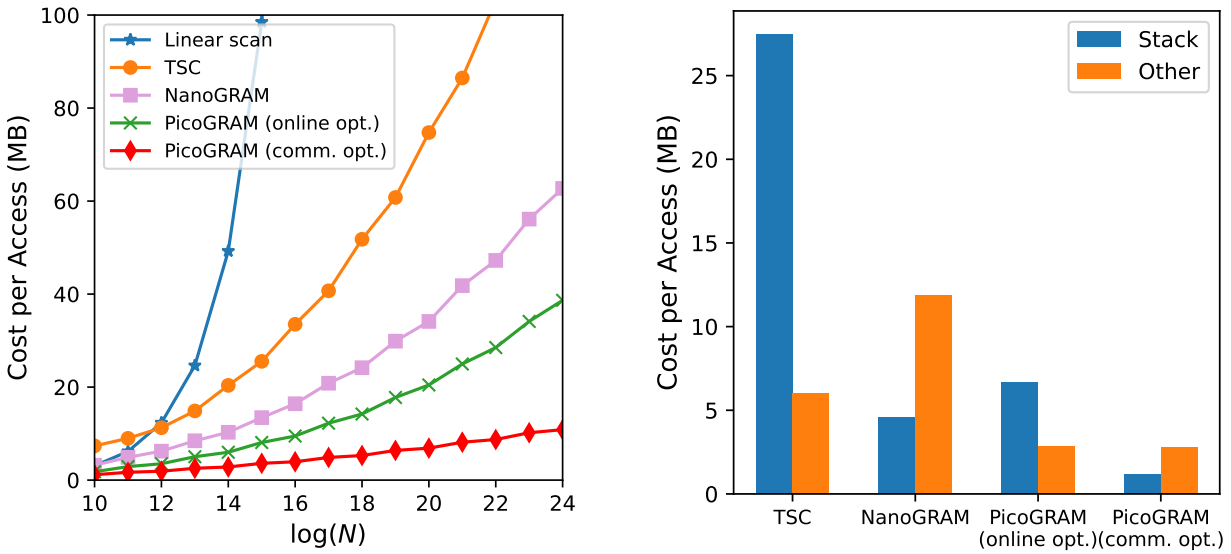
Figure 10: Construction of Compaction and Distribution stacks (Definition 6.1).

7 Evaluation

We implemented PicoGRAM in C++ and evaluated both its communication and computation costs. The implementation is open-sourced at https://github.com/picogramimpl/picogram. We compare PicoGRAM against four baselines, with a word width of 64-bit, an equivalent computational security parameter of approximately 128 bits, and a target statistical failure probability $\sigma = 2^{-40}$ throughout. We set the RAM's runtime T equal to its space N in the evaluation, and ignore the costs from the CPU circuitry of the RAM.

Our evaluation compares the following approaches.

- **Linear scan** naïvely scans the entire memory space for each access using state-of-the-art garbled circuit construction [RR21].
- **TSC** simulates the tri-state GRAM following the pseudocode in the paper [HKO23]. While AND gates are emulated with the tri-state gates in the original work [HKO23], we replace them with the state-of-the-art construction [RR21] to match our scheme.
- **NanoGRAM** runs the cost simulator of NanoGRAM [PLS23], a concretely efficient GRAM, despite its asymptotically higher communication than TSC [HKO23].
- **Interactive** captures state-of-the-art RAM-model 2PC [WCS15, LWN+15, RR21] performance. Specifically, we implemented the interactive baseline by removing the stacks in PicoGRAM and instead allowing interactions between the garbler and evaluator. This way, each access to a non-recursive ORAM tree incurs one round trip, and we assume a WAN setup with 100 ms round trip time (RTT).



(a) Amortized communication cost.

(b) Breakdown for $N = 2^{16}$.

Figure 11: Amortized communication cost of PicoGRAM compared to previous GRAM constructions. Since our PicoGRAM (online opt.) variant does not use SIMD garbling, it also shows a breakdown of the contribution from our two main techniques.

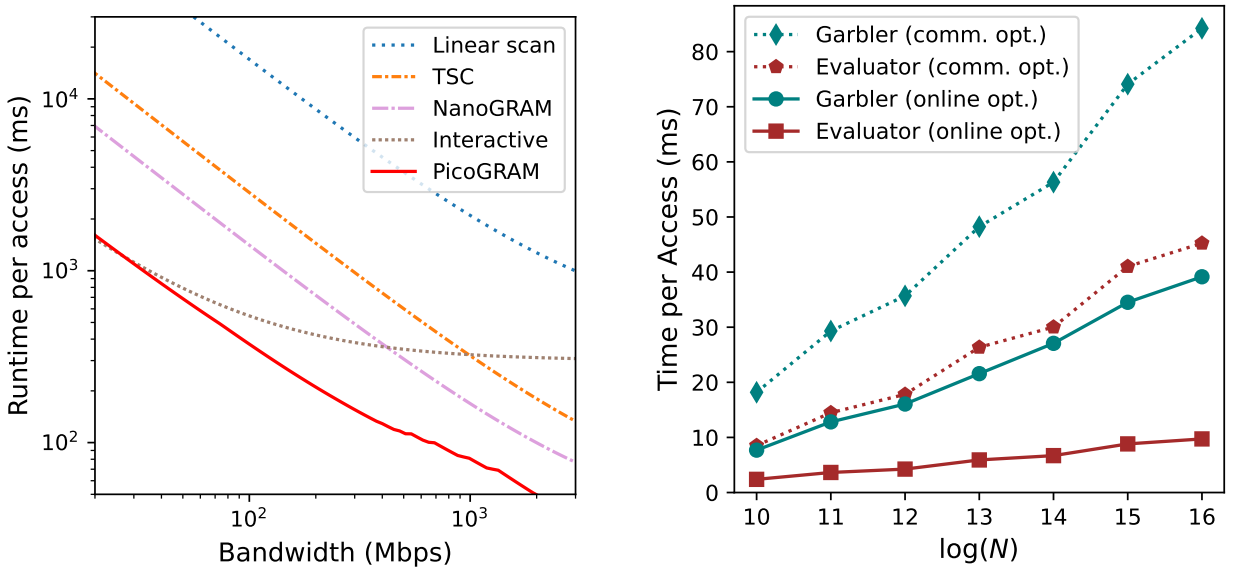
- **PicoGRAM** is our scheme. The evaluation considers three different variants of PicoGRAM optimized for different metrics.

  1. "PicoGRAM (comm. opt.)" is a variant optimized for communication overhead using the SIMD gates and DDH.
  2. "PicoGRAM (online opt.)" is a variant optimized for online computation. This variant is more suited in scenarios where the offline garbling overhead and the transmission of the garbled circuits is not on the critical path, and only the evaluation time is on the critical path. Therefore, this variant does not use the SIMD optimizations and the DDH assumption.
  3. "PicoGRAM" is a variant optimized for the end-to-end time, including garbling time, network transmission time, and evaluation time. Depending on the system configuration (e.g., network bandwidth available, ping latency, and CPU clock rate), we auto-tune the parameters as described in Section D.4.

For our end-to-end runtime evaluation, we use 8 cores at 2.2 GHz on an Intel Xeon Platinum 8352S CPU. Our communication cost results are platform-independent. Further details on the implementation and concrete performance of the baselines are provided in Section E.

Communication cost. Figure 11 compares the communication cost of PicoGRAM (comm. opt.) to other constant-round baselines. For RAM spaces ranging from 2^{10} to 2^{24}, PicoGRAM reduces communication by $5.3 \sim 12.6\times$ over TSC [HKO23], and by $2.8 \sim 5.8\times$ over NanoGRAM [PLS23]. A breakdown of the cost is shown in Figure 11b. At $N = 2^{16}$, we reduce the communication cost from stacks by $23.7\times$, with $5.7\times$ of this improvement attributed to our SIMD garbling.

End-to-end runtime for 2PC. Due to the use of elliptic curves, PicoGRAM incurs higher computational costs than prior state-of-the-art GRAMs [HKO23, PLS23]. For example, on our test



(a) End-to-end time for $N = 2^{16}$.

(b) Computation time of PicoGRAM.

Figure 12: (a) Amortized end-to-end time for 2PC at $N = 2^{16}$ excluding OT, simulated under varying network bandwidths, where the garbler's computation is assumed to be pipelined with communication. (b) Amortized computation time of PicoGRAM's garbler and evaluator when optimized for different metrics.

platform, OpenSSL's P-256 elliptic curve [CMR+23, Oom24] requires approximately 1.2×10^5 CPU cycles per scalar multiplication, whereas a 128-bit hash using fixed-key AES [BHKR13] takes only about 40 cycles.

Nevertheless, in a two-party computation (2PC) setting[5], our auto-tuned PicoGRAM achieves a 6.28× speedup in end-to-end time over tri-state GRAM [HKO23] and a 3.15× speedup over NanoGRAM [PLS23] at a bandwidth of 300 Mbps and for $N = 2^{16}$. As bandwidth increases, computation becomes the bottleneck and the speedup decreases; despite this, at 2,000 Mbps, PicoGRAM remains 3.62× faster than tri-state [HKO23] and 2.01× faster than NanoGRAM [PLS23]. Conversely, as bandwidth decreases, the interactive baseline becomes more competitive, since round-trip latency is less dominant. Our simulation shows that PicoGRAM is 2.46× faster than the interactive baseline at 300 Mbps, and remains faster until the bandwidth drops to 34 Mbps.

As our current implementation requires the garbled circuit stored in memory, we are not able to measure the runtime of larger experiments. With additional engineering, however, it would be feasible to store most of the garbled circuit on disk and load the required parts into memory with minimal overhead through pipelined pre-fetching. We leave this optimization for future work. Based on extrapolation, we estimate the amortized end-to-end runtime to be approximately 381 ms for $N = 2^{24}$ and 600 ms for $N = 2^{30}$ at a bandwidth of 300 Mbps. In comparison, the interactive baseline is estimated to take 919 ms for $N = 2^{24}$ and 1,365 ms for $N = 2^{30}$ under the same bandwidth conditions.

Online time. As an advantage over the interactive protocols, GRAMs can be deployed efficiently in a preprocessing setting, where the online phase is dominated by the evaluator's computation time (see Section E.3 for more discussions). Similar to the high-bandwidth scenario above, by

[5]We do not take into account the cost of oblivious transfers (OT), which depends on the program's input size.

tuning the parameters, we manage to reduce the evaluator's computation time to less than 10 ms per access for $N \leq 2^{16}$, as shown in Figure 12b. In comparison, the interactive baseline requires at least 100 to 300 ms online time per access due to the round trips. For $N = 2^{24}$ and $N = 2^{30}$, we estimate our online time to increase to 25 ms and 41 ms per access, respectively. In comparison, the interactive baseline is estimated to take 316 ms and 455 ms per access, even if it is also tuned to minimize the online time.

Acknowledgements

This work is in part support by NSF under award numbers 2128519, 2212746, and 2044679, a Packard Fellowship, an ONR grant, and a DARPA SIEVE grant under a subcontract from SRI, a JP Morgan Faculty Research Award, and a research grant from 0xPARC.

References

[ACC+16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating ram computations with adaptive soundness and privacy. In Martin Hirt and Adam Smith, editors, *Theory of Cryptography*, pages 3–30, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[AKL+23] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. Optorama: Optimal oblivious RAM. *J. ACM*, 70(1):4:1–4:70, 2023.

[AKLS23] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, and Elaine Shi. Oblivious RAM with worst-case logarithmic overhead. *J. Cryptol.*, 36(2):7, 2023.

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography*, pages 455–472, Cham, 2018. Springer International Publishing.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 439–448, New York, NY, USA, 2015. Association for Computing Machinery.

[BHHO08] Dan Boneh, Shai Halevi, Mike Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 108–125, Berlin, Heidelberg, 2008. Springer-Verlag.

[BHKO23] Cruz Barnum, David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Adaptive garbled circuits and garbled RAM from non-programmable random oracles. *IACR Cryptol. ePrint Arch.*, page 1527, 2023.

[BHKR13] Mihir Bellare, Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. pages 478–492, 05 2013.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*,

STOC '90, page 503–513, New York, NY, USA, 1990. Association for Computing Machinery.

[BN16] Elette Boyle and Moni Naor. Is there an oblivious ram lower bound? In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, page 357–368, New York, NY, USA, 2016. Association for Computing Machinery.

[Can01] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[CCC+16] Yu-Chi Chen, Sherman S.M. Chow, Kai-Min Chung, Russell W.F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel ram from indistinguishability obfuscation. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, page 179–190, New York, NY, USA, 2016. Association for Computing Machinery.

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Adaptive succinct garbled ram or: How to delegate your database. In *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986*, page 61–90, Berlin, Heidelberg, 2016. Springer-Verlag.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled ram. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, page 169–178, New York, NY, USA, 2016. Association for Computing Machinery.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for ram programs. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 429–437, New York, NY, USA, 2015. Association for Computing Machinery.

[CHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2022.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-xor" technique. In Ronald Cramer, editor, *Theory of Cryptography*, pages 39–53, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[CMR+23] Lily Chen, Dustin Moody, Andrew Regenscheid, Angela Robinson, and Karen Randall. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. Technical Report NIST Special Publication (SP) 800-186, National Institute of Standards and Technology, Gaithersburg, MD, 2023.

[Coh05] Henri Cohen. Analysis of the sliding window powering algorithm. *Journal of Cryptology*, 18(1):63–76, Jan 2005.

[DS17] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 523–535, New York, NY, USA, 2017. Association for Computing Machinery.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[FNR+15] Christopher W. Fletcher, Muhammad Naveed, Ling Ren, Elaine Shi, and Emil Stefanov. Bucket ORAM: single online roundtrip, constant bandwidth oblivious RAM. *IACR Cryptol. ePrint Arch.*, page 1065, 2015.

[GGH+13a] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 40–49, 2013.

[GGH+13b] Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova, and Daniel Wichs. Optimizing oram and using it efficiently for secure computation. In Emiliano De Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, pages 1–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 405–422, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[GKK+12] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 513–524. ACM, 2012.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled ram. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, pages 210–229, USA, 2015. IEEE Computer Society.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 449–458, New York, NY, USA, 2015. Association for Computing Machinery.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[Goo11] Michael T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 379–388, New York, NY, USA, 2011. Association for Computing Machinery.

[HCS17] T.-H. Hubert Chan and Elaine Shi. Circuit opram: Unifying statistically and computationally secure orams and oprams. In *Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, page 72–107, Berlin, Heidelberg, 2017. Springer-Verlag.

[Hea23] David Heath. Parallel RAM from cyclic circuits. *CoRR*, abs/2309.05133, 2023.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In *Advances in Cryptology - EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*, pages 3–31, Berlin, Heidelberg, 2024. Springer-Verlag.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure Two-Party computation using garbled circuits. In *20th USENIX Security Symposium (USENIX Security 11)*, San Francisco, CA, August 2011. USENIX Association.

[HKO22] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Epigram: Practical garbled ram. In *Advances in Cryptology - EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, pages 3–33, Berlin, Heidelberg, 2022. Springer-Verlag.

[HKO23] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Tri-state circuits: A circuit model that captures ram. In *Advances in Cryptology - CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, pages 128–160, Berlin, Heidelberg, 2023. Springer-Verlag.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. 2020.

[Kel20] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1575–1590, New York, NY, USA, 2020. Association for Computing Machinery.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 419–428, New York, NY, USA, 2015. Association for Computing Machinery.

[KS08] Vladimir Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages and Programming*, 2008.

[LHS+14] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. Automating efficient ram-model secure computation. In *2014 IEEE Symposium on Security and Privacy*, pages 623–638, 2014.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 595–608. ACM, 2023.

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious ram lower bound! In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 523–542, Cham, 2018. Springer International Publishing.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 719–734. Springer, 2013.

[LO15] Steve Lu and Rafail Ostrovsky. Black-box parallel garbled RAM. Cryptology ePrint Archive, Paper 2015/1068, 2015. https://eprint.iacr.org/2015/1068.

[LO17] Steve Lu and Rafail Ostrovsky. Black-box parallel garbled ram. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 66–92, Cham, 2017. Springer International Publishing.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, April 2009.

[LP14] Huijia Lin and Rafael Pass. Succinct garbling schemes and applications. *IACR Cryptol. ePrint Arch.*, 2014:766, 2014.

[LWN+15] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376, 2015.

[met] An implementation of oblivious ram by meta. https://github.com/facebook/oram.

[Mon85] Peter Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[MS04] Pradeep Kumar Mishra and Palash Sarkar. Application of montgomery's trick to scalar multiplication for elliptic and hyperelliptic curves using a fixed base point. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *Public Key Cryptography – PKC 2004*, pages 41–54, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, page 129–139, New York, NY, USA, 1999. Association for Computing Machinery.

[olaa] Oblivious key-value store by oblivious labs. https://github.com/obliviouslabs/oram.

[olab] Scalable oblivious accesses to blockchain data. https://writings.flashbots.net/scalable-oblivious-accesses-to-blockchain-data.

[Oom24] Jeroen Ooms. *openssl: Toolkit for Encryption, Signatures and Certificates Based on OpenSSL*, 2024. R package version 2.2.2.

[PLS23] Andrew Park, Wei-Kai Lin, and Elaine Shi. Nanogram: Garbled ram with o(logn) overhead. In *Advances in Cryptology - EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*, pages 456–486, Berlin, Heidelberg, 2023. Springer-Verlag.

[PPRY18] Sarvar Patel, Giuseppe Persiano, Mariana Raykova, and Kevin Yeo. Panorama: Oblivious RAM with logarithmic overhead. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 871–882. IEEE Computer Society, 2018.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I*, page 94–124, Berlin, Heidelberg, 2021. Springer-Verlag.

[SDS+18] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4), April 2018.

[Shi25] Elaine Shi. Private information retrieval and oblivious ram. Talk in Simons "Cryptography 10 Years Later: Obfuscation, Proof Systems, and Secure Computation Boot Camp", 2025.

[SHS+15] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428, 2015.

[sig] Technology deep dive: Building a faster oram layer for enclaves. https://signal.org/blog/building-faster-oram/.

[WCS15] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 850–861, New York, NY, USA, 2015. Association for Computing Machinery.

[WHC+14] Xiao Shaun Wang, Yan Huang, T-H. Hubert Chan, Abhi Shelat, and Elaine Shi. Scoram: Oblivious ram for secure computation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 191–202, New York, NY, USA, 2014. Association for Computing Machinery.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[YPHK23] Yibin Yang, Stanislav Peceny, David Heath, and Vladimir Kolesnikov. Towards generic mpc compilers via variable instruction set architectures (visas). In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 2516–2530, New York, NY, USA, 2023. Association for Computing Machinery.

[ZE13] Samee Zahur and David Evans. Circuit structures for improving efficiency of security and privacy tools. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, page 493–507, USA, 2013. IEEE Computer Society.

[ZPSZ24] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server PIR with sublinear server computation. In *IEEE Symposium on Security and Privacy*, 2024.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.

[ZWR+16] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting square-root oram: Efficient random access in multi-party computation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 218–234, 2016.

A Deferred Cost Analysis

In this section, we provide a detailed cost analysis of PicoGRAM, including additional optimizations to achieve our claimed bounds in Theorem 1.1. At a high level, we adopt the same garbled RAM construction as Heath et al. [HKO23], except that: 1) we replace the tri-state circuit stack with our optimized stack based on SIMD gates; 2) we generalize the GRAM to handle both cases $T > N$ and $T < N$; and 3) we perform additional optimizations to reduce the bucket sizes of the root and leaf nodes. We have discussed part 1) in Section 5.3, and next we explain how to achieve 2) and 3).

A.1 Handling Cases $T > N$ and $T < N$

In our previous analysis (Section 2.2), we assumed, as in prior work [HKO23], that the RAM's runtime T equals its space N. Here, we show that this assumption can be removed. Namely, we can construct a GRAM with amortized communication cost $O\left(\lambda \cdot \left(W \cdot \log N + \log^3 N\right) \cdot \omega(1)\right)$ for any T.

Case $T < N$: We provision $O(T)$ leaves in both the main ORAM tree and each of the recursive position maps, since there can be at most $O(T)$ elements inserted into each tree. The main ORAM tree thus has amortized communication cost $O(\lambda(W + \log T) \log T \cdot \omega(1))$, and each position map has cost $O(\lambda \log^2 T \cdot \omega(1))$. Since the RAM space is N, we still need $\log N$ position maps. As $T < N$, the total amortized cost is bounded by $O\left(\lambda \cdot \left(W \cdot \log N + \log^3 N\right) \cdot \omega(1)\right)$.

Case $T > N$: We construct $\lceil T/N \rceil$ GRAMs and connect them in series. Each GRAM is capable of handling $3N$ accesses: the first N accesses are for initializing the memory, the last N accesses are for extracting the data to initialize the next GRAM, and the remaining N accesses are for the actual memory operations. Asymptotically, the amortized communication cost remains the same as in the case $T = N$.

A.2 Improving Dependence on the $\omega(1)$ Factor

Next, we describe techniques that further improve the asymptotic performance of PicoGRAM to match our claimed results in Theorem 1.1. Specifically, we reduce the amortized communication cost from

$$O\left(\lambda \cdot \left(W \cdot \log N + \log^3 N\right) \cdot \omega(1)\right) \text{ to } O\left(\lambda \cdot \left(W \cdot \log N \cdot \omega(1) + \log^3 N\right)\right).$$

In other words, we eliminate the $\omega(1)$ factor for the case $W \in o(\log^2 N)$.

In our previous analysis (Section 2.2 and Table 3), the $\omega(1)$ factor arises from both the root buckets and the leaf buckets, as they are super-logarithmic in size. Below, we present two additional optimizations to remove the $\omega(1)$ factor from the leaf buckets and the root buckets, respectively.

Reducing leaf bucket size via better load balancing In Table 3, we cut off the ORAM tree at level $\log(N/B)$ and hope that no leaf node is invoked more than $2B$ times. To ensure that the tree can still store N elements, we have to set the leaf bucket size to $O(B)$. Moreover, we need $B \in \log N \cdot \omega(1)$ to achieve negligible failure probability, which leads to our first $\omega(1)$ inefficiency.[6]

Ideally, we want to avoid the cutoff and keep all bucket sizes constant except for the root, which means we need to ensure that each leaf is invoked only $O(1)$ times. While the tri-state circuit

[6] In interactive RAM-model 2PC, this is not an issue because the circuits can be generated on the fly. In Garbled RAM, however, the garbler has to provision sufficient circuitry for the worst case.

work [HKO23] assumes this property in their construction and cost analysis, they did not explain how to achieve it in practice.

Our key idea is to improve load balancing using a random permutation. Observe that during each access, the path traversed in the ORAM tree is determined by the leaf position previously assigned to the element. Consider an ORAM tree with capacity N and $T = N$ accesses; for simplicity, let the tree have $2N$ leaves indexed by positions $1, 2, \ldots, 2N$. We need to assign N random positions initially and N new random positions after each access. Instead of choosing these positions uniformly at random, we assign them according to a random permutation π of $[2N]$. Specifically, we set the initial positions to $\pi[1], \pi[2], \ldots, \pi[N]$ and the new positions to $\pi[N+1], \pi[N+2], \ldots, \pi[2N]$. This guarantees that each leaf is accessed at most once during the N accesses. While Circuit ORAM also requires performing eviction on two paths after every access, the eviction paths follow a reverse lexicographical order of the positions [WCS15], which only adds one more access to each leaf.

This approach preserves the security of the ORAM, as the positions of the accessed leaves (i.e., the leakage) can be simulated by sampling N elements without replacement from $2N$. Since we assume a semi-honest setting, we can let the garbler sample the random permutation, leveraging the fact that the garbler does not know which path is accessed at runtime.[7]

This optimization has been integrated into our implementation of PicoGRAM, as it also improves concrete performance. When comparing with the tri-state GRAM baseline in Section 7, we assume that they adopt the same optimization.

Reducing root size by merging stash. Chan et al. [HCS17] observed that the root buckets of all recursive position maps can be merged into a single stash of size $O(\log N) \cdot \omega(1)$ while maintaining negligible failure probability. This optimization ensures that only the main ORAM tree incurs the $\omega(1)$ factor, not the recursive trees (see Section 10 in their online full version). The same optimization can be applied to PicoGRAM.

During each GRAM access, oblivious routing is performed on the stash to distribute entries to each of the recursive trees [HCS17]. Since the stash size is $O(\log N) \cdot \omega(1)$, the oblivious routing can be implemented with a circuit of size $O(\log^2 N \cdot \mathsf{poly} \log \log N)$, as each position map has a word size of $O(\log N)$.

It remains an open question whether the $\omega(1)$ factor can also be eliminated from the main ORAM tree. For Circuit ORAM [WCS15], the failure probability is bounded by $\exp(-c \cdot R)$, where R is the stash size and c is a constant, so R must be set to be super-logarithmic to achieve negligible failure probability.

A.3 Final Cost Analysis

Finally, we combine the optimizations above and obtain the following theorem on the communication cost of PicoGRAM.

Theorem A.1 (Communication cost of PicoGRAM (restatement of Theorem 1.1))**.** PicoGRAM *achieves an amortized communication cost of* $O(\lambda \cdot (W \cdot \log N \cdot \omega(1) + \log^3 N))$ *bits per instruction, where* N *is the RAM's space,* W *is the word width,* λ *is the security parameter, and* $\omega(1)$ *is an arbitrarily small super-constant factor in* N*.*

Proof. In Section A.1, we showed that it suffices to consider the case $T = N$.

[7] In practice, it may also be desirable to periodically reveal to the garbler the timestep of each node at runtime, so that the circuit can be garbled and sent on demand. In such scenarios, the garbler and evaluator can jointly compute and secret share the random permutation.

As established in Section 5.3, the communication cost from stacks sums to $O(\lambda \cdot (W + \log^2 N) \log N)$ bits per access.

Next, consider the costs from the Circuit ORAM buckets. With the first optimization in Section A.2, every bucket except the root bucket has constant size. Since the circuit size for each read and eviction operation is linear in the bucket size and word width in Circuit ORAM, the communication costs from all non-root buckets sum to $O(\lambda \cdot W \cdot \log^2 N)$. The root bucket of the main ORAM tree costs $O(\lambda \cdot W \cdot \log N \cdot \omega(1))$ bits per access, and, applying the second optimization in Section A.2, the root bucket (i.e., the unified stash) of all recursive position maps costs $O(\lambda \cdot W \cdot \log^2 N \cdot \mathsf{poly} \log \log N)$. Therefore, the overall communication cost from the Circuit ORAM buckets is $O(\lambda \cdot (W \cdot \log N \cdot \omega(1) + \log^3 N))$ bits per access.

Summing the costs from both parts yields the claimed result. □

Remarks on computation cost. In prior works on practical garbled RAM [HKO23, PLS23], only communication cost is measured, as it typically dominates computation in most practical scenarios. However, since PicoGRAM significantly reduces communication cost while introducing more expensive DDH group operations, we also analyze the asymptotic computation cost in terms of the number of multiplications in $\mathbb{G}_q$.

As shown in our garbling scheme (Figure 3 and 4), exponentiation operations are required only when the garbler invokes a Group or Ungroup gate, or when the evaluator invokes an Ungroup gate. For each Switch gate, the evaluator performs a single modular multiplication or division in the exponent, deferring the actual heavy-weight exponentiations to the Ungroup gates. Each Group and Ungroup gate involves $O(1)$ exponentiations, or equivalently, $O(\lambda)$ multiplications in $\mathbb{G}_q$. Therefore, both the garbler's and evaluator's computation costs are upper-bounded by $O\left(\lambda \left(W \cdot \log N + \log^3 N\right)\right)$ multiplications per instruction.

Furthermore, since the garbler only needs to exponentiate the generator g, we can leverage precomputation as described in Section D.2 to reduce the garbler's computation cost to $O(\lambda \cdot (W + \log^2 N))$ multiplications per instruction.

B Deferred Analysis of Improved Stack Circuitry

B.1 Preliminaries

First, we review some properties of Goodrich's compaction network [Goo11] that will facilitate our analysis of the stack circuitry.

Definition B.1 (Goodrich's Compaction Network [Goo11])**.** In Goodrich's compaction network [Goo11], elements are routed through a $(d+1) \times t$ grid of cells from level 1 to level $d+1$, where $d = \log t$. A cell is **used** if an element is routed through it, and **unused** otherwise. Let e be a real element input to the τ-th cell at level 1, and let S_τ be the number of unused cells before e at level 1. Then, the network routes e to the τ_ℓ-th cell at level $\ell \in [2, d+1]$, where $\tau_\ell = \tau - \left(S_\tau \bmod 2^{\ell-1}\right)$.

Lemma B.2. *Goodrich's compaction network [Goo11] preserves the order of the elements at every level.*

Proof. Let e and e' be real elements input at cells τ and τ', respectively, at level 1, with $\tau' < \tau$. Let S_τ and $S_{\tau'}$ be the number of unused cells before e and e' at level 1. By Definition B.1, e and e' are routed to the τ_ℓ-th and τ'_ℓ-th cells at level ℓ, where $\tau_\ell = \tau - (S_\tau \bmod 2^{\ell-1})$ and $\tau'_\ell = \tau' - (S_{\tau'} \bmod 2^{\ell-1})$. Observe that

$$(S_{\tau'} \bmod 2^{\ell-1}) - (S_\tau \bmod 2^{\ell-1}) \leq S_{\tau'} - S_\tau \leq \tau - \tau' - 1.$$

Therefore, $\tau'_\ell + 1 \le \tau_\ell$, which shows that the order of the elements is preserved at every level. □

Corollary B.3. *Let* e *be a real element input to the τ-th cell at level 1 of Goodrich's compaction network [Goo11], and S_τ be the number of unused cells before* e *at level 1. Then, the number of unused cells at level ℓ before* e *is*

$$S_{\ell,\tau_\ell} = \left\lfloor S_\tau / 2^{\ell-1} \right\rfloor \cdot 2^{\ell-1}.$$

Proof. By Lemma B.2, the number of used cells before e remains $\tau - 1 - S_\tau$ at every level. The compaction network routes e to the τ_ℓ-th cell at level ℓ, where $\tau_\ell = \tau - (S_\tau \bmod 2^{\ell-1})$. Thus, the number of unused cells before e at level ℓ is

$$\begin{aligned} S_{\ell,\tau_\ell} &= (\tau_\ell - 1) - (\tau - 1 - S_\tau) \\ &= S_\tau - (S_\tau \bmod 2^{\ell-1}) \\ &= \left\lfloor S_\tau / 2^{\ell-1} \right\rfloor \cdot 2^{\ell-1}. \end{aligned}$$

□

B.2 Analysis of Eager-Prefix-Sum

We first show that the eager-prefix-sum building block (Figure 9) is well-formed; that is, every wire in the circuit can be set to a unique value given the input values $c_1, \ldots, c_t$.

Lemma B.4. *Eager-prefix-sum (Figure 9) is strictly well-formed.*

Proof. Note that only one of the Switches in $z_{\ell+1,\tau} \xrightarrow{a_{\ell,\tau}} 0$ and $z_{\ell+1,\tau} \xrightarrow{\neg a_{\ell,\tau}} b_{\ell,\tau}$ can be active and set $z_{\ell+1,\tau}$. The rest of the construction consists of Boolean circuits, which uniquely set each wire. It is straightforward to check that every wire is set after all its dependencies are set, since the circuit is mostly Boolean, and we can always set $a_{\ell,\tau}$ and $\neg a_{\ell,\tau}$ before setting the subwires converted from 0 and $b_{\ell,\tau}$. □

Next, we show that eager-prefix-sum correctly outputs the routing schedule for Goodrich's compaction network [Goo11].

Lemma B.5. *Given the values of $c_1, \ldots, c_t$, eager-prefix-sum (Figure 9) sets*

$$s_{\ell,\tau_\ell} = \left\lfloor S_\tau / 2^{\ell-1} \right\rfloor \bmod 2 \tag{1}$$

for every $\ell \in [d]$ and $\tau \in [t]$, where $S_\tau = \sum_{i=1}^{\tau-1} c_i$ and $\tau_\ell = \tau - \left(S_\tau \bmod 2^{\ell-1}\right)$.

Proof. We prove the lemma by induction on ℓ, together with an additional induction hypothesis:

- In Goodrich's compaction network (Definition B.1), with an input element at every timestep $\tau \in [t]$ such that $c_\tau = 0$, the wire $z_{\ell,\tau}$ can be set to 0 if and only if there is an element routed through the τ-th cell at level ℓ.

Base case: when $\ell = 1$, we indeed have $z_{1,\tau} = c_\tau$ is set to 0 if and only if there is an element routed through the τ-th cell at level 1. Moreover, the recurrence relations $s_1 = 0$ and $s_{1,\tau+1} \leftarrow \mathrm{XOR}(z_{1,\tau}, s_{1,\tau})$ imply

$$s_{1,\tau} = \sum_{i=1}^{\tau-1} z_{1,i} \bmod 2 = \sum_{i=1}^{\tau-1} c_i \bmod 2 = \left\lfloor S_\tau / 2^{1-1} \right\rfloor \bmod 2 \tag{2}$$

Inductive step from ℓ to $\ell+1$: Notice that there is an element routed through the τ-th cell at level $\ell+1$ if and only if either of the following cases hold:

- $z_{\tau,\ell}=0$ and $s_{\ell,\tau}=0$. Namely, there is an element reaching the τ-th cell on level ℓ and routed directly downward to level $\ell+1$.
- $z_{\tau,\ell}=0$ and $s_{\ell,\tau+2^{\ell-1}}=1$. Namely, there is an element reaching the $\left(\tau+2^{\ell-1}\right)$-th cell on level ℓ, and the element is routed leftward to level $\ell+1$.

This matches the circuitry for computing $z_{\ell+1,\tau}$ in the eager-prefix-sum construction. Therefore, the additional induction hypothesis holds for level $\ell+1$.

Next, we show that Equation (1) also holds for level $\ell+1$. In the proof below, we define the shorthand notation

$$Z_{\ell+1,\tau}=\left\lfloor\left(\sum_{i=1}^{\tau-1}z_{\ell+1,i}\right)/2^{\ell}\right\rfloor, \tag{3}$$

and our first goal is to show that

$$s_{\ell+1,\tau}=Z_{\ell+1,\tau} \bmod 2. \tag{4}$$

Intuitively, Corollary B.3 shows that the number of unused cells between every two consecutive used cells is always a multiple of 2^{ℓ} at level $\ell+1$, so we can pack the unused cells into chunks of length 2^{ℓ}, and $Z_{\ell+1,\tau}$ counts the number of full chunks before timestep τ.

When $\tau\le 2^{\ell}$, Equation (4) trivially holds, since $Z_{\ell+1,\tau}$ can only be 0, and $s_{\ell+1,\tau}$ is set to 0 by definition. Therefore, we can prove the equality inductively for every $\tau\in[t]$ if the following recurrence relation holds:

$$s_{\ell+1,\tau}=Z_{\ell+1,\tau} \bmod 2 \;\Rightarrow\; s_{\ell+1,\tau+2^{\ell}}=Z_{\ell+1,\tau+2^{\ell}} \bmod 2. \tag{5}$$

Since $s_{\ell+1,\tau+2^{\ell}}=s_{\ell+1,\tau}\oplus z_{\ell+1,\tau}$, Equation (5) holds if

$$Z_{\ell+1,\tau+2^{\ell}}=Z_{\ell+1,\tau}+z_{\ell+1,\tau}. \tag{6}$$

To prove Equation (6), we discuss both the case $z_{\ell+1,\tau}=0$ and $z_{\ell+1,\tau}=1$.

- **Case $z_{\ell+1,\tau}=0$:** then there is an element routed through the τ-th cell on level $\ell+1$ by the additional induction hypothesis. Moreover, by Corollary B.3, the number of unused cells before the τ-th cell at level $\ell+1$ is a multiple of 2^{ℓ}. Therefore, we can rewrite Equation (3) as

  $$Z_{\ell+1,\tau}=\left(\sum_{i=1}^{\tau-1}z_{\ell+1,i}\right)/2^{\ell} \tag{7}$$

  Thus, we have

  $$\sum_{i=1}^{\tau+2^{\ell}-1}z_{\ell+1,i}=\left(\sum_{i=1}^{\tau-1}z_{\ell+1,i}\right)+0+\left(\sum_{i=\tau+1}^{\tau+2^{\ell}-1}z_{\ell+1,i}\right)\le Z_{\ell+1,\tau}\cdot 2^{\ell}+2^{\ell}-1. \tag{8}$$

  Hence, $Z_{\ell+1,\tau+2^{\ell}}$ is bounded by

  $$Z_{\ell+1,\tau}\le Z_{\ell+1,\tau+2^{\ell}}\le\left\lfloor\frac{Z_{\ell+1,\tau}\cdot 2^{\ell}+2^{\ell}-1}{2^{\ell}}\right\rfloor=Z_{\ell+1,\tau} \tag{9}$$

  and $Z_{\ell+1,\tau+2^{\ell}}=Z_{\ell+1,\tau}+z_{\ell+1,\tau}$ when $z_{\ell+1,\tau}=0$.

- **Case $z_{\ell+1,\tau} = 1$:** First, by Equation (3), we have the upper-bound

$$Z_{\ell+1,\tau+2^\ell} \le \left\lfloor \left(\sum_{i=1}^{\tau-1} z_{\ell+1,i} \right) / 2^\ell \right\rfloor + \left\lceil \left(\sum_{i=\tau}^{\tau+2^\ell-1} z_{\ell+1,i} \right) / 2^\ell \right\rceil = Z_{\ell+1,\tau} + 1 \tag{10}$$

It remains to show the lower-bound $Z_{\ell+1,\tau+2^\ell} \ge Z_{\ell+1,\tau} + 1$. Again, we consider two cases:

  - If $z_{\ell+1,\tau+1} = \cdots = z_{\ell+1,\tau+2^\ell-1} = 1$, then

$$\sum_{i=1}^{\tau+2^\ell-1} z_{\ell+1,i} = \left(\sum_{i=1}^{\tau-1} z_{\ell+1,i} \right) + 2^\ell \tag{11}$$

    which implies $Z_{\ell+1,\tau+2^\ell} = Z_{\ell+1,\tau} + 1$

  - If there exists $\tau' \in [\tau+1, \tau+2^\ell-1]$ such that $z_{\ell+1,\tau'} = 0$, then by Corollary B.3, the number of unused cells before the τ'-th cell at level $\ell+1$ is a multiple of 2^ℓ. Therefore,

$$Z_{\ell+1,\tau+2^\ell} \ge \left\lfloor \left(\sum_{i=1}^{\tau'-1} z_{\ell+1,i} \right) / 2^\ell \right\rfloor = \left(\sum_{i=1}^{\tau'-1} z_{\ell+1,i} \right) / 2^\ell \tag{12}$$

    Furthermore, since $z_{\ell+1,\tau} = 1$,

$$\left(\sum_{i=1}^{\tau'-1} z_{\ell+1,i} \right) / 2^\ell > \left(\sum_{i=1}^{\tau-1} z_{\ell+1,i} \right) / 2^\ell \ge Z_{\ell+1,\tau} \tag{13}$$

    Combining Equation (12) and (13), we get $Z_{\ell+1,\tau+2^\ell} > Z_{\ell+1,\tau}$. Since both $Z_{\ell+1,\tau+2^\ell}$ and $Z_{\ell+1,\tau}$ are integers, we have $Z_{\ell+1,\tau+2^\ell} \ge Z_{\ell+1,\tau} + 1$. Therefore, Equation (6) also holds when $z_{\ell+1,\tau} = 1$.

This concludes the proof of Equation (4).

By the additional induction hypothesis, the number of unused cells before the $\tau_{\ell+1}$-th cell at level $\ell+1$ can be expressed as

$$S_{\ell+1,\tau_{\ell+1}} = \sum_{i=1}^{\tau_\ell} z_{\ell,i} \tag{14}$$

Combining Equation (4), Equation (14), and Corollary B.3, we get our lemma's claim for level $\ell+1$:

$$\begin{aligned} s_{\ell+1,\tau_{\ell+1}} &= \left\lfloor \left(\sum_{i=1}^{\tau_{\ell+1}-1} z_{\ell+1,i} \right) / 2^\ell \right\rfloor \bmod 2 \\ &= \left\lfloor S_{\ell+1,\tau_{\ell+1}} / 2^\ell \right\rfloor \bmod 2 \\ &= \left\lfloor S_\tau / 2^\ell \right\rfloor \bmod 2 \end{aligned} \tag{15}$$

Thus, we have shown both induction hypotheses, which imply the lemma. □

Finally, we show that eager-prefix-sum can indeed set the output wires eagerly, resolving the circular dependency problem in the RAM circuit.

Lemma B.6. *For $\tau \le t$, if $c_1, \ldots, c_\tau$ are set, then eager-prefix-sum (Figure 9) can further set*

- *every wire $s_{\ell,\tau'}$ for $\ell \in [d]$ and $\tau' \le \tau$, and*
- *if $c_\tau = 0$, also every wire $s_{\ell,\tau_\ell+2^{\ell-1}}$ for $\ell \in [d]$, where $\tau_\ell = \tau - \left(S_\tau \bmod 2^{\ell-1}\right)$.*

Moreover, the wires in eager-prefix-sum are set following the dependency order in Definition 3.3.

Proof. We first show that $s_{\ell,\tau'}$ can be set for every $\ell \in [d]$ and $\tau' \le \tau$. We prove this by induction on ℓ, together with an extra induction hypothesis that z_{ℓ,τ'_ℓ} can be set for every $\ell \in [d]$ and $\tau'_\ell \le \tau - 2^{\ell-1}$.

For $\ell = 1$, this is trivial: $z_{1,1}, \ldots, z_{1,\tau}$ are set to the input values, $s_{1,1}$ is set to constant 0, and $s_{1,2}, \ldots, s_{1,\tau}$ can be set by the XOR gates.

If the invariant holds for level ℓ, then we can set both $a_{\ell,\tau'_{\ell+1}}$ and $b_{\ell,\tau'_{\ell+1}}$ for every $\tau'_{\ell+1} \le \tau - 2^\ell$, which further lets us set $z_{\ell+1,\tau'_{\ell+1}}$ using the Switch gates. Using the XOR gates, we can then set $s_{\ell+1,\tau'}$ for every $\tau' \le \tau$, completing the induction.

Now, for the eager evaluation property: by Lemma B.5, we have

$$\begin{aligned}
\tau_\ell - \tau_{\ell+1} &= \left(S_\tau \bmod 2^\ell\right) - \left(S_\tau \bmod 2^{\ell-1}\right) \\
&= \left(\left\lfloor S_\tau / 2^{\ell-1} \right\rfloor \bmod 2\right) \cdot 2^{\ell-1} \\
&= s_{\ell,\tau_\ell} \cdot 2^{\ell-1}. \qquad (16)
\end{aligned}$$

We prove by induction that z_{ℓ,τ_ℓ} can be set to 0 for every $\ell \in [d]$.

For $\ell = 1$, $z_{1,\tau_\ell} = c_{\tau_\ell} = 0$.

If the induction hypothesis holds for $\ell \in [d-1]$, then we discuss the following two cases on the value of s_{ℓ,τ_ℓ} (we have shown that s_{ℓ,τ_ℓ} can be set since $\tau_\ell \le \tau$):

- If $s_{\ell,\tau_\ell} = 0$, then a_{ℓ,τ_ℓ} can be set to 0, and $\tau_{\ell+1} = \tau_\ell$. Therefore, $z_{\ell+1,\tau_{\ell+1}} = z_{\ell+1,\tau_\ell}$ can be set to 0 using the Switch gate.
- If $s_{\ell,\tau_\ell} = 1$, then $\tau_{\ell+1} = \tau_\ell - 2^{\ell-1}$. Since $z_{\ell,\tau_\ell} = \neg s_{\ell,\tau_\ell} = 0$, we can set $b_{\ell,\tau_{\ell+1}} = b_{\ell,\tau_\ell - 2^{\ell-1}} = 0$ using the OR gate. Moreover, since $\tau_{\ell+1} = \tau_\ell - 2^{\ell-1} \le \tau - 2^{\ell-1}$, we have shown that both $z_{\ell,\tau_{\ell+1}}$ and $s_{\ell,\tau_{\ell+1}}$ can be set. Therefore, $a_{\ell,\tau_{\ell+1}}$ can be set. No matter $a_{\ell,\tau_{\ell+1}}$ takes 0 or 1, we can hence set $z_{\ell+1,\tau_{\ell+1}} = 0$ using one of the Switch gates.

This concludes the induction that z_{ℓ,τ_ℓ} can be set to 0. Thus, with the XOR gates, we can set $s_{\ell,\tau_\ell+2^{\ell-1}}$ for every $\ell \in [d]$.

Finally, as shown in Lemma B.4, each wire is set after all its dependencies are set. □

B.3 Analysis of SIMD Stack

Since the eager-prefix-sum circuit is well-formed, each wire $s_{\ell,\tau}$ in the SIMD stack is set to a unique value, which in turn uniquely determines whether each Switch in the SIMD stack is active. This allows us to abstract the runtime state of the SIMD stack as a graph, simplifying the analysis.

Definition B.7 (Meta-graph of SIMD Stack)**.** The meta-graph of a SIMD Stack (Figure 7) with respect to input controls $c_1, \ldots, c_t$ is an undirected graph (V, E), where the vertices V are all the cables $\mathbf{u}^W_{\ell,\tau}$, and there is an edge between two vertices (cables) if they are connected by an active Switch gate.

We now establish several properties of the meta-graph of the SIMD stack, which will be used in subsequent proofs.

Lemma B.8. *In the meta-graph of the SIMD stack with respect to input controls $c_1, \ldots, c_t$, the vertex $\mathbf{u}^W_{1,\tau}$ is connected to $\mathbf{u}^W_{\ell,\tau_\ell}$, where $\tau_\ell = \tau - \left(\sum_{i=1}^{\tau-1} c_i \bmod 2^{\ell-1}\right)$, for every $\ell \in [d+1]$.*

Proof. By the construction of SIMD stack (Figure 7), $\mathbf{u}^W_{\ell,\tau_\ell}$ is connected to $\mathbf{u}^W_{\ell+1,\tau_\ell}$ if $s_{\ell,\tau_\ell} = 0$, and $\mathbf{u}^W_{\ell+1,\tau_\ell - 2^{\ell-1}}$ if $s_{\ell,\tau_\ell} = 1$. By Equation (16), $\mathbf{u}^W_{\ell,\tau_\ell}$ is connected to $\mathbf{u}^W_{\ell+1,\tau_{\ell+1}}$ for every $\ell \in [d]$, and hence $\mathbf{u}^W_{1,\tau}$ is connected to $\mathbf{u}^W_{\ell,\tau_\ell}$ for every $\ell \in [d+1]$.

□

Lemma B.9. *In the meta-graph of the SIMD stack with respect to input controls $c_1, \ldots, c_t$, the vertex $\mathbf{u}^W_{d+1,\tau^{(1)}}$ is not connected to $\mathbf{u}^W_{d+1,\tau^{(2)}}$ for any $\tau^{(1)} \neq \tau^{(2)}$.*

Proof. First, we show that the meta-graph always has $t \cdot d$ edges. Note that the number of edges equals the number of control wires with value 0. Since $s_{\ell,\tau} = 0$ for $\tau \in [2^{\ell-1}]$, we can treat $\neg s_{\ell,\tau}$ as a control wire for $\tau \in [2^{\ell-1}]$ in Figure 7, which does not affect the count. Then, there are $2td$ control wires, half of which are 0, so the number of edges is td.

Meanwhile, the meta-graph has $t(d+1)$ vertices, so there are at least t connected components. Since there is always one active Switch connecting each cable at level $\ell \in [d]$ with a cable on the next level, every vertex is connected to a vertex on the last level. Thus, no pair of vertices on the last level can be connected, or there would be fewer than t connected components. □

Corollary B.10. *In the meta-graph of the SIMD stack with respect to input controls $c_1, \ldots, c_t$, every vertex $\mathbf{u}^W_{\ell,\tau}$ is connected to a unique vertex $\mathbf{u}^W_{d+1,\tau'}$ on the last level.*

Proof. As shown in Lemma B.9, every vertex $\mathbf{u}^W_{\ell,\tau}$ is connected to at least one vertex on the last level, which must also be unique since there cannot be two vertices on the last level both connected to $\mathbf{u}^W_{\ell,\tau}$. □

Lemma B.11. *In the meta-graph of the SIMD stack with respect to input controls $c_1, \ldots, c_t$, the vertex $\mathbf{u}^W_{\ell,\tau^{(1)}}$ is not connected to $\mathbf{u}^W_{\ell,\tau^{(2)}}$ for any $0 < \tau^{(1)} - \tau^{(2)} < 2^{\ell-1}$, where $\ell \in [d+1]$ and $\tau^{(1)}, \tau^{(2)} \in [t]$.*

Proof. Suppose that $\mathbf{u}^W_{\ell,\tau^{(1)}}$ and $\mathbf{u}^W_{\ell,\tau^{(2)}}$ are connected. Then, by Corollary B.10 and Lemma B.9, both vertices must be connected to a unique last-level vertex $\mathbf{u}^W_{d+1,\tau'}$. However, by the topology of the compaction network, each vertex $\mathbf{u}^W_{j,\tau}$ can only connect to either $\mathbf{u}^W_{j+1,\tau}$ or $\mathbf{u}^W_{j+1,\tau-2^{j-1}}$ on the next level, so we have $\tau^{(1)} \equiv \tau' \equiv \tau^{(2)} \bmod 2^{\ell-1}$, which contradicts $0 < \tau^{(1)} - \tau^{(2)} < 2^{\ell-1}$.

□

Lemma B.12. *In the meta-graph of the SIMD stack with respect to input controls $c_1, \ldots, c_t$, for each vertex $\mathbf{y}^W_{\tau'}$ with $\tau' \leq t' = t - \sum_{\tau=1}^{t} c_\tau$, there exists a unique τ such that $c_\tau = 0$ and $\mathbf{x}^W_\tau$ is connected to $\mathbf{y}^W_{\tau'}$.*

Proof. Define $U_x = \left\{\mathbf{x}^W_\tau \mid c_\tau = 0\right\}$ and $U_y = \left\{\mathbf{y}^W_{\tau'} \mid \tau' \in [t']\right\}$. By Corollary B.10, every vertex in U_x is connected to a unique vertex in U_y. Now we prove every vertex in U_y is also connected to at most one vertex in U_x. It suffices to show that $\mathbf{x}^W_{\tau^{(1)}}$ and $\mathbf{x}^W_{\tau^{(2)}}$ are not connected to the same vertex in U_y if $\tau^{(1)} \neq \tau^{(2)}$ and $c_{\tau^{(1)}} = c_{\tau^{(2)}} = 0$. By Lemma B.8, we need to show:

$$\tau^{(1)} < \tau^{(2)} \text{ and } c_{\tau^{(1)}} = c_{\tau^{(2)}} = 0 \;\Rightarrow\; \tau^{(1)} - \sum_{i=1}^{\tau^{(1)}-1} c_i \neq \tau^{(2)} - \sum_{i=1}^{\tau^{(2)}-1} c_i \tag{17}$$

And Equation (17) holds because $\tau^{(2)} - \tau^{(1)} > \sum_{i=\tau^{(1)}+1}^{\tau^{(2)}-1} c_i = \sum_{i=\tau^{(1)}}^{\tau^{(2)}-1} c_i$.

Finally, we prove that each vertex in U_y is connected to at least one vertex in U_x. Suppose that there exists a vertex in U_y that is not connected to any vertex in U_x. Then, since $|U_x| = |U_y|$ and each vertex in U_y is connected to at most one vertex in U_x, there must exist a vertex in U_x that is not connected to any vertex in U_y by the pigeonhole principle. This contradicts the fact that each vertex in U_x is connected to a vertex in U_y. □

Lemma B.13. *In the meta-graph of the SIMD stack with respect to input controls $c_1, \ldots, c_t$, if $c_\tau = 0$, then the vertex $\mathbf{u}^W_{1,\tau}$ is not connected to $\mathbf{u}^W_{\ell,\tau'_\ell}$ if $\tau'_\ell > \tau_\ell - 2^{\ell-1}$ and $\tau'_\ell \neq \tau_\ell$, where $\ell \in [d]$ and $\tau_\ell = \tau - \left(S_\tau \bmod 2^{\ell-1}\right)$.*

Proof. Let's split the vertices into four disjoint sets:

- $U_1 = \left\{\mathbf{u}^W_{\ell,\tau'} \mid \tau' \in \left[0,\, \tau_\ell - 2^{\ell-1}\right]\right\}$;
- $U_2 = \left\{\mathbf{u}^W_{\ell,\tau'} \mid \tau' \in \left(\tau_\ell - 2^{\ell-1},\, \tau_\ell\right)\right\}$;
- $U_3 = \left\{\mathbf{u}^W_{\ell,\tau'} \mid \tau' = \tau_\ell\right\}$;
- $U_4 = \left\{\mathbf{u}^W_{\ell,\tau'} \mid \tau' \in (\tau_\ell,\, t]\right\}$.

We want to prove that $\mathbf{u}^W_{1,\tau}$ is connected to vertices only in U_1 and U_3.

By Lemma B.8, $\mathbf{u}^W_{1,\tau}$ is connected to every vertex in U_3, and by Lemma B.11, vertices in U_3 are not connected to vertices in U_2. Therefore, it suffices to show that U_3 and U_4 are disconnected.

Since $\tau_\ell - \tau_{\ell+1} \leq 2^{\ell-1}$, by the topology of the compaction network, there can be no edge between U_1 and U_4, so it only remains to show that there is no edge between U_3 and U_4.

Suppose that there is an edge between U_3 and U_4. Then, by the topology of the compaction network, the edge can only match one of the following two cases:

- Case vertical: The edge is between $\mathbf{u}^W_{\ell,\tau_\ell} \in U_3$ and $\mathbf{u}^W_{\ell+1,\tau_\ell} \in U_4$.
- Case diagonal: The edge is between $\mathbf{u}^W_{\ell+1,\tau_{\ell+1}} \in U_3$ and $\mathbf{u}^W_{\ell,\tau_{\ell+1}+2^{\ell-1}} \in U_4$.

For the vertical case, since $\mathbf{u}^W_{\ell+1,\tau_\ell} \notin U_3$, we have $\tau_{\ell+1} \neq \tau_\ell$. By Equation (16), we have $s_{\ell,\tau_\ell} = 1$, which means the Switch between $\mathbf{u}^W_{\ell,\tau_\ell}$ and $\mathbf{u}^W_{\ell+1,\tau_\ell}$ is inactive, so the edge cannot exist.

For the diagonal case, since $\mathbf{u}^W_{\ell,\tau_{\ell+1}+2^{\ell-1}} \notin U_3$, we have $\tau_\ell \neq \tau_{\ell+1} + 2^{\ell-1}$. By Equation (16), we have $s_{\ell,\tau_\ell} = 0$ and thus $\neg s_{\ell,\tau_\ell} = 1$, which means the Switch between $\mathbf{u}^W_{\ell+1,\tau_{\ell+1}}$ and $\mathbf{u}^W_{\ell,\tau_{\ell+1}+2^{\ell-1}}$ is inactive, so the edge cannot exist.

To conclude, U_2 is disconnected to U_3, and $U_1 \cup U_3$ is disconnected to U_4. Therefore, $U_2 \cup U_4$ is disconnected to U_3. □

B.4 Analysis of Compaction and Distribution Stack

Now we are ready to analyze our construction of the compaction and distribution stacks. We first prove their correctness in Theorem B.14.

Theorem B.14. *The compaction and distribution stack constructions in Figure 10 correctly implement the state transitions defined in Definition 6.1.*

Proof. When $c_\tau = 0$, $\mathbf{r}_\tau^W$ is connected to $\mathbf{u}_\tau^W$ by an active Switch gate. Furthermore, by Lemma B.8, the cables $\mathbf{u}_\tau^W$ and $\mathbf{v}_{\tau'}^W$ are connected in the SIMD stack's meta-graph, where $\tau' = \tau_{\ell+1} = \tau - \sum_{j=1}^{\tau-1} c_j$. Finally, by Lemma B.6, the control wires of all the active Switches on the path can be set when $c_1, \ldots, c_\tau$ are set. Therefore, if either $x_{\tau,i}$ or $y_{\tau',i}$ is set, we can set the other to the same value using Group, Ungroup, and active Switch gates. □

Next, we show that our compaction and distribution stacks can be used to build a strictly well-formed ORAM circuit. We adopt the ORAM construction as described in prior work [HKO23]. Since the ORAM buckets can be implemented purely as Boolean circuits, it suffices to show that the stack constructions can achieve the state transitions as defined in Definition 6.1 by setting every wire following the dependency order.[8]

Theorem B.15. *If $\sum_{\tau=1}^{t} c_\tau = t - t'$, the compaction stack construction in Figure 10 is strictly well-formed, and moreover, during the state transitions of the compaction stack, each wire can be set after all its dependencies are set.*

Proof. First, we show that the compaction stack is well-formed. By Lemma B.12, every cable $\mathbf{v}_{\tau'}^W$ in the compaction stack is connected to a unique cable $\mathbf{r}_\tau^W$ through a chain of active Switch gates for $\tau' \in [t']$. Moreover, by Lemma B.11, $\mathbf{v}_{\tau'}^W$ is not connected to the constant zero cables. Therefore, every subwire $v_{\tau',i}$ is set to a unique value. Since the cables $\mathbf{v}_\tau^W$ are mutually disconnected for $\tau \in [t]$, every cable $\mathbf{u}_{\ell,\tau}^W$ in the SIMD stack is connected to a unique cable $\mathbf{v}_\tau^W$, and hence every subwire can also be set uniquely. Finally, since the eager-prefix-sum circuit is well-formed, every wire in the compaction stack can be set.

Next, we show that each wire can be set after all its dependencies are set during the state transitions of the compaction stack. In the rest of the proof, when we say a wire can be set, we mean the wire can be set after all its dependencies. Crucially, we need to show that the Group gate can set each subwire of $\mathbf{r}_\tau^W$ in the compaction stack. If $c_\tau = 1$, then $\mathbf{r}_\tau^W$ is not connected to other cables by active Switch gates, so its subwire only depends on c_τ and the input wire $x_{\tau,i}$. If $c_\tau = 0$, then by Lemma B.13, $\mathbf{r}_\tau^W$ is not connected to any cable in U_2 and U_4 (as defined in the proof of Lemma B.13) through a chain of active Switch gates. By the SIMD stack construction,

- If $s_{\ell,\tau'}$ or $\neg s_{\ell,\tau'}$ is the control wire of a Switch connected to any cable in U_1, then we have $\tau' \le \tau_{\ell+1} - 2^{\ell-1} + 2^{\ell-1} \le \tau$.
- If $s_{\ell,\tau'}$ or $\neg s_{\ell,\tau'}$ is the control wire of a Switch connected to any cable in U_3, then we have either $\tau' = \tau_\ell \le \tau$, or $\tau' = \tau_\ell + 2^{\ell-1}$. Since $c_\tau = 0$.

For either case, by Lemma B.6, we can set $s_{\ell,\tau'}$ and $\neg s_{\ell,\tau'}$ once we obtain $c_1, \ldots, c_\tau$.

Therefore, the Group gate can set each subwire of $\mathbf{r}_\tau^W$ in the compaction stack after all its dependencies are set. Once we set $\mathbf{r}_\tau^W$, we can further set every subwire in the cable connected to it through the active Switch gates, and achieve the state transition as defined in Definition 6.1. Finally, we can set the subwires of cable $\mathbf{0}^W$ and other cables connected to it through active Switch gates, since all the control wires are set.

By Lemma B.4, every internal wire in the eager-prefix-sum circuit can be set as well. Therefore, the compaction stack is also strictly well-formed. □

[8] A minor technicality is that we cannot directly use Boolean circuits to merge the outputs of the distribution stacks from different child nodes, since the state transition rule of the distribution stack ensures $x_{\tau,i}$ is set only if $c_\tau = 0$. However, we can convert $x_{\tau,i}$ back to cables and use Switch gates to select the output from the distribution stack with $c_\tau = 0$. For better concrete efficiency, we can also avoid this back-and-forth conversion between subwires and independent wires.

Theorem B.16. *If $\sum_{\tau=1}^{t} c_\tau = t - t'$, the distribution stack construction in Figure 10 is strictly well-formed, and moreover, during the state transitions of the distribution stack, each wire can be set after all its dependencies are set.*

Proof. By Corollary B.10, every cable $\mathbf{u}_{\ell,\tau}^{W}$ in the SIMD stack is connected to a unique cable $\mathbf{y}_{\tau'}^{W}$ through a chain of active Switch gates, and hence can be set to a unique value.

The rest of the proof is similar to that of Theorem B.15. The Group gate can set each subwire of $\mathbf{y}_{\tau'}^{W}$ if $c_\tau = 0$ and $\tau' = \tau - \sum_{i=1}^{\tau-1} c_i$, since all the control wires the subwire depends on are set. The rest of the subwires can be set through the active Switch gates. Finally, since eager-prefix-sum is strictly well-formed, the distribution stack is also strictly well-formed. □

Using the following lemma, we can show that the RAM construction in Theorem 5.2 is also strictly well-formed.

Lemma B.17 (Strict well-formedness of RAM)**.** *The SIMD tri-state circuit $\mathcal{C}$ that $\mathsf{negl}(N)$-obliviously simulates P, as stated in Theorem 5.2, is strictly well-formed if both its compaction and distribution stacks are strictly well-formed, and moreover, if each wire is set after all its dependencies during the state transitions defined in Definition 6.1.*

Proof. Heath et al. [HKO23] have shown that every wire in $\mathcal{C}$ can be set uniquely following the evaluation rules, assuming the stack and distribution stacks implement the declared state transitions. We further show that during this evaluation process, every wire is set after all its dependencies are set. Note that except for the stacks, the rest parts of $\mathcal{C}$ are simply Boolean circuits. For each Boolean gate, the evaluation rule guarantees that the output wire of the gate can be set only after both the input wires are set, which means all the dependencies of the output wire have been set. Since every wire of the stacks can also be set after all their dependencies during the evaluation, we conclude that every wire in $\mathcal{C}$ is set after all its dependencies are set. □

Finally, we analyze the cost of our compaction and distribution stacks.

Theorem B.18. *The compaction and distribution stack constructions in Figure 10 include $O(t \cdot (W + \log t))$ gates.*

Proof. The eager-prefix-sum circuit consists of $d = \log t$ levels, each with $O(t)$ Boolean, Switch, Group, and Ungroup gates. The SIMD stack additionally includes $O(d \cdot t)$ Switch gates. The compaction and distribution stacks further include $O(t)$ Switch gates and $O(d \cdot t)$ Group and Ungroup gates. Summing these, we obtain the theorem statement. □

C Garbling SIMD Tri-State Circuits in Random Oracle Model

In this section, we present a concretely more efficient garbling scheme and prove its security under the Decisional Diffie-Hellman (DDH) assumption in the random oracle (RO) model. Our scheme incorporates several concrete optimizations [KS08, NPS99, BMR90] and leverages the state-of-the-art half-gate technique [RR21] for garbling Boolean gates. Furthermore, the circular security provided by the random oracle allows us to relax the requirement from strictly well-formed circuits to merely well-formed circuits. This relaxation enables further simplification of the stack construction and improves concrete performance.

C.1 Our Construction from DDH and Random Oracle

Labels of an independent wire. As in prior works that use the Free-XOR optimization [KS08, ZRE15, RR21, HKO23, Hea24], we set the zero and one labels of each independent wire x so that $\mathsf{L}_{x=0} \oplus \mathsf{L}_{x=1} = \Delta$, where $\oplus$ denotes bit-wise XOR and Δ is a global key held by the garbler. Each label is $\lambda + 1$ bits, where λ is the security parameter, and the least significant bit is called the permutation bit. We ensure the zero and one labels of each wire have different permutation bits by setting the least significant bit of Δ to 1. This allows the garbler to reveal the value of an independent wire to the evaluator by sending only the permutation bit of the wire's zero label.

Labels of a cable. For cables, we assign the label of each subwire to be an element of a prime-order DDH group $\mathbb{G}_q$ with security parameter λ. Let $W_{\max}$ be the maximum width of any cable in the circuit. The garbler samples $W_{\max}$ global keys $\Gamma_1, \ldots, \Gamma_{W_{\max}} \in \mathbb{Z}_q^*$. For each cable $\mathbf{x}^W$, the garbler also holds a secret key $\mathsf{K}_{\mathbf{x}} \in \mathbb{Z}_q^*$, and defines the labels of the i-th subwire as $\mathsf{L}_{x_i=b} = g^{\mathsf{K}_{\mathbf{x}} \cdot (\Gamma_i + b)}$ for $b \in \{0, 1\}$, where g is a public generator of the group.[9]

Next, we explain how our garbling scheme with random oracle improves concretely over the construction in Section 4.

Optimizing Boolean gates. Since Yao's original garbled circuit [Yao86], a series of works [BMR90, NPS99, KS08, ZRE15, RR21] have improved the concrete efficiency of garbled Boolean circuits. We adopt the state-of-the-art scheme by Rosulek and Roy [RR21], where XOR gates are free and each AND gate incurs only $1.5\lambda + 5$ bits of communication.

Optimizing Group and Ungroup gates. Group and Ungroup gates are essentially two-row garbled truth tables, and can be optimized using standard techniques. Specifically, we apply the Point-and-Permute technique [BMR90] to Group gates, leveraging the permutation bit in the labels of independent wires. For Ungroup gates, we apply the GRR3 row reduction technique [NPS99], since the output labels of Ungroup gates can be freely chosen.

Optimizing Switch. To further reduce the cost, we apply a similar row reduction technique to the Switch gates. Specifically, for a Switch $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$, if the cable keys of $\mathbf{x}^W$ and $\mathbf{y}^W$ satisfy $\mathsf{K}_{\mathbf{y}} = H_{\mathbb{Z}}(\mathsf{L}_{c=0}, \mathsf{gid}) \cdot \mathsf{K}_{\mathbf{x}}$, then the evaluator can convert between the labels of $\mathbf{x}^W$ and $\mathbf{y}^W$ given the zero label of the control wire, and the garbler only needs to send a single bit to reveal the value of the control wire. However, since each cable may be connected to multiple Switch gates, we can only apply this optimization to a subset of Switch gates, which we call the **spanning Switch gates**. The remaining Switch gates are garbled using the standard technique, where we leverage the random oracle to encrypt the ratio of $\mathsf{K}_{\mathbf{x}}$ and $\mathsf{K}_{\mathbf{y}}$.[10]

Definition C.1 (Root Cables and Spanning Switch Gates)**.** For a SIMD tri-state circuit $\mathcal{C}$, let G be the undirected graph where each cable in $\mathcal{C}$ maps to a vertex, and each Switch gate maps to an edge between the two cables (i.e., vertices) it connects to. We select a subset of cables in $\mathcal{C}$ as **root cables** following the rules below:

- A cable is a root cable if it has a subwire input to an Ungroup gate.
- For each connected component in G without a root cable, we select an arbitrary cable in the component as the root cable.

[9] Compared to the construction in Section 4, here we further correlate the zero and one labels of subwires. This enables "free" addition across the subwires of the same cable, as detailed in Section D.1.

[10] Except for the SIMD optimizations, our spanning Switch gates are similar to the Buffer gates in the original work of tri-state circuits [HKO23], and the non-spanning Switch gates each combine a Buffer and a Join gate as in [HKO23].

Moreover, we define the **spanning Switch gates** of $\mathcal{C}$ to be a maximum subset of Switch gates so that the induced sub-graph of G is acyclic and disconnects all the root cables with each other.

Full construction. The definition above provides the garbler with a topological ordering to compute all the cable keys and labels of independent wires. The garbler can first sample the cable keys of all the root cables, which lets her compute the labels of the output wire of every Ungroup gate. Then, combining these labels with the labels of the circuit's input wires, the garbler can compute the labels of every independent wire in the circuit using a garbling scheme for Boolean circuits. Finally, with the labels of all the control wires, the garbler can compute the cable keys of the remaining non-root cables according to the constraints imposed by the spanning Switch gates.

The evaluator, on the other hand, still evaluates the circuit following the topological ordering specified by the gates' evaluation rules. By well-formedness, the evaluator is guaranteed to obtain a unique active label of each wire corresponding to the wire's value. Finally, we let the garbler reveal the value of each output wire to the evaluator by sending the permutation bit of the wire's zero label. We present the full garbling scheme in Figure 13 and Figure 14.

C.2 Analysis

Efficiency. Each Boolean gate incurs $1.5\lambda+5$ bits of communication, as shown in [RR21], where λ is the bit-length of the CCRH. Each Group gate incurs $2 \cdot \lambda_{\mathrm{DDH}}$ bits, where λ_{DDH} is the bit-length of a DDH group element. Each Ungroup gate incurs $2\lambda + 2$ bits. Each Switch gate incurs either 1 bit (if it is a spanning Switch) or $\lceil \log q \rceil + 1$ bits (if not), where q is the order of the DDH group. In our implementation, we instantiate CCRH using 128-bit fixed-key AES [BHKR13] and use the P-256 elliptic curve with compressed points for the DDH group. Thus, we set $\lambda = 127$, $\lambda_{\mathrm{DDH}} = 257$, and $\lceil \log q \rceil = 256$.

Correctness. We first verify that the garbler can indeed garble all gates. Every Ungroup gate can be garbled because its input wire always belongs to a root cable, whose key is independently sampled. Consequently, the garbler also obtains the labels of all output wires of Ungroup gates. Next, the garbler can garble all Boolean gates and obtain the labels of every independent wire. Since Boolean gates are garbled and evaluated in the same order, if a gate could not be garbled, the evaluator would not be able to evaluate it and obtain the output wire, which would contradict the well-formedness requirement. As all control wires are independent, the garbler can compute $H_{\mathbb{Z}}(\mathsf{L}_{c=0}, \mathsf{gid})$ for each Switch gate with control wire c. By Definition C.1, every cable is connected to a root cable through a unique chain of spanning Switch gates, so the garbler can compute the cable keys and labels for all cables in the circuit. Finally, the garbler can compute all remaining gate materials using these keys and labels.

We can now check that Eval correctly recovers the label for each wire corresponding to the wire's value, given the correctness of [RR21]. Unlike the construction in Section 4, the evaluator now learns the values of each control and output wire via the permutation bit. By well-formedness, the evaluator can obtain the label of every wire and thus learn the correct output of the circuit.

Security. We prove the security of the garbling scheme, as formalized in Theorem C.4. At a high level, we first define a set of oracles that embed the garbler's secret keys, and show that, without additional knowledge of these keys, the outputs of the oracles are computationally indistinguishable from random functions. We then construct a hybrid simulator that uses these oracles to generate a garbled circuit with a distribution identical to the real one, where the simulator is also given the wire values from the real-world circuit evaluation. Finally, we obtain the ideal simulator by

$\mathsf{Gen}(1^\lambda, |\mathsf{inp}|)$:

- Sample the global key $\Delta \in \{0,1\}^{\lambda+1}$ where the least significant bit is set to 1.
- Sample the zero label $\mathsf{L}_{x=0} \xleftarrow{\$} \{0,1\}^{\lambda+1}$ for each input wire x.
- Select a prime-order DDH group $\mathbb{G}_q$ with security parameter λ, and a generator g of the group.
- Return $\mathsf{sk} = (\Delta, \mathsf{L}, \mathbb{G}_q, g)$.

$\mathsf{Encode}(\mathsf{sk} = (\Delta, \mathsf{L}, \_, \_), \mathsf{inp})$:

- Return $\widetilde{\mathsf{inp}}$ where $\widetilde{\mathsf{inp}}[i] \leftarrow \mathsf{L}_{x=0} \oplus \mathsf{inp}[i] \cdot \Delta$ for each i-th input wire x.

$\mathsf{Garble}(\mathsf{sk} = (\Delta, \mathsf{L}, \mathbb{G}_q, g), \mathcal{C})$:

- Sample the global keys $\Gamma_i \xleftarrow{\$} \mathbb{Z}_q^*$ for $i \in [W_{\max}]$.
- Sample the cable key $\mathsf{K}_{\mathbf{x}} \xleftarrow{\$} \mathbb{Z}_q^*$ for each root cable $\mathbf{x}^W$.
- For each Ungroup gate $y \xleftarrow{\text{ungrp}} x_i$, set the gate's garbled material

$$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (h_0 \oplus h_1 \oplus \Delta,\ \mu_\beta)$$

  and the output wire's zero label

$$\mathsf{L}_{y=0} \leftarrow h_\beta \oplus \beta \cdot \Delta$$

  where $\beta \xleftarrow{\$} \{0,1\}$, $h_b \leftarrow H(\mathsf{L}_{x_i=b}, \mathsf{gid} \| \text{“}h\text{”})$, and $\mu_b \leftarrow H(\mathsf{L}_{x_i=b}, \mathsf{gid} \| \text{“}\mu\text{”})$.
- Garble all the Boolean gates $z \leftarrow f(x,y)$ following the Garble scheme in [RR21], and store the zero label $\mathsf{L}_{x=0}$ of every independent wire x.
- Solve the keys of the remaining cables so that for each spanning Switch gate $\mathbf{x}^W \overset{c}{\text{—}\!\!\diagup\!\!\text{—}} \mathbf{y}^W$, the equation $\mathsf{K}_{\mathbf{y}} = H_{\mathbb{Z}}(\mathsf{L}_{c=0}, \mathsf{gid}) \cdot \mathsf{K}_{\mathbf{x}}$ holds.
- For every Switch gate $\mathbf{x}^W \overset{c}{\text{—}\!\!\diagup\!\!\text{—}} \mathbf{y}^W$, if it is a spanning Switch gate, set its garbled material

$$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \mathsf{LSB}(\mathsf{L}_{c=0})$$

  where $\mathsf{LSB}(\cdot)$ is the least significant bit of the input; otherwise

$$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left(\mathsf{LSB}(\mathsf{L}_{c=0}),\ H_{\mathbb{Z}}(\mathsf{L}_{c=0}, \mathsf{gid}) \cdot \mathsf{K}_{\mathbf{x}} \cdot \mathsf{K}_{\mathbf{y}}^{-1}\right)$$

- For each Group gate $y_i \xleftarrow{\text{grp}} x$, set the gate's garbled material as

$$\widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left((\mathsf{L}_{y_i=\beta})^{h_\beta},\ (\mathsf{L}_{y_i=\neg\beta})^{h_{\neg\beta}}\right)$$

  where $\beta \leftarrow \mathsf{LSB}(\mathsf{L}_{x=0})$ and $h_b \leftarrow H_{\mathbb{Z}}(\mathsf{L}_{x=b}, \mathsf{gid})$.
- Let $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{LSB}(\mathsf{L}_{y=0})$ for each i-th output wire y.
- Return $\widetilde{\mathcal{C}} = \left(\mathcal{C}, \mathbb{G}_q, \widetilde{\mathsf{Gates}}, \widetilde{\mathsf{Out}}\right)$

Figure 13: Our garbling scheme based on DDH and random oracle (continued in Figure 14).

$\mathsf{Eval}(\widetilde{\mathcal{C}} = \left(\mathcal{C}, \mathbb{G}_q, \widetilde{\mathsf{Gates}}, \widetilde{\mathsf{Out}}\right), \widetilde{\mathsf{inp}})$:

- Let $\mathsf{L}_x \leftarrow \widetilde{\mathsf{inp}}[i]$ for each i-th input wire x.
- Compute the labels of the rest of the wires following the order defined by the evaluation rules and using the garbled material of the gates:
  - **Boolean** $z \leftarrow f(x, y)$**:** Compute L_z from L_x and L_y following the Eval scheme in [RR21].
  - **Group** $y_i \xleftarrow{\textbf{grp}} x$**:** Parse $\widetilde{\mathsf{Gates}}[\mathsf{gid}]$ as (e_0, e_1), and set
    $$\mathsf{L}_{y_i} \leftarrow (e_\beta)^{h^{-1}} \text{ where } \beta \leftarrow \mathsf{LSB}(\mathsf{L}_x) \text{ and } h \leftarrow H_{\mathbb{Z}}(\mathsf{L}_x, \mathsf{gid})$$
  - **Ungroup** $y \xleftarrow{\textbf{ungrp}} x_i$: Parse $\widetilde{\mathsf{Gates}}[\mathsf{gid}]$ as (K, μ).
    $$\text{If } \mu = H(\mathsf{L}_{x_i}, \mathsf{gid} \| \text{“}\mu\text{”}),\ \mathsf{L}_y \leftarrow H(\mathsf{L}_{x_i}, \mathsf{gid} \| \text{“}h\text{”})$$
    $$\text{Else } \mathsf{L}_y \leftarrow H(\mathsf{L}_{x_i}, \mathsf{gid} \| \text{“}h\text{”}) \oplus K$$
  - **Switch** $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$: If it is a spanning Switch, set $\beta \leftarrow \widetilde{\mathsf{Gates}}[\mathsf{gid}]$ and $R \leftarrow H_{\mathbb{Z}}(\mathsf{L}_c,\ \mathsf{gid})$; otherwise, parse $\widetilde{\mathsf{Gates}}[\mathsf{gid}]$ as (β, R).
    If $\beta \neq \mathsf{LSB}(\mathsf{L}_c)$, skip the gate. Otherwise, for each $i \in [W]$, set $\mathsf{L}_{y_i} \leftarrow (\mathsf{L}_{x_i})^R$ if L_{x_i} is known, and set $\mathsf{L}_{x_i} \leftarrow (\mathsf{L}_{y_i})^{R^{-1}}$ if L_{y_i} is known.†
- For each i-th output wire y, set $\mathsf{out}[i] \leftarrow \mathsf{LSB}(\mathsf{L}_y) \oplus \widetilde{\mathsf{Out}}[i]$.
- Return out

† *The exponentiation can be computed lazily, same as in Figure 4.*

Figure 14: Our garbling scheme based on DDH and random oracle (continued from Figure 13).

replacing oracle calls with random sampling, and show that the ideal simulator only needs access to the values of the output and control wires.

Lemma C.2. *Let g be a public generator of a DDH group $\mathbb{G}_q$ with prime order q with respect to security parameter λ. Let $W, t \in \mathsf{poly}\lambda$ be positive integers, and $b \in \{0,1\}^{W \times t}$ be a $W \times t$ matrix of binary values. Then,*

$$\left\{ M_{\gamma,k,b} \mid \gamma \xleftarrow{\$} (\mathbb{Z}_q^*)^W, k \xleftarrow{\$} (\mathbb{Z}_q^*)^t \right\}_\lambda \overset{c}{\approx} \text{Uniform}\left(\mathbb{G}_q^{W \times k}\right)_\lambda$$

where

$$M_{\gamma,k,b} = \begin{pmatrix} g^{k_1 \cdot (\gamma_1 + b_{1,1})} & \dots & g^{k_t \cdot (\gamma_1 + b_{1,t})} \\ \vdots & \ddots & \vdots \\ g^{k_1 \cdot (\gamma_W + b_{W,1})} & \dots & g^{k_t \cdot (\gamma_W + b_{W,t})} \end{pmatrix}$$

and $\overset{c}{\approx}$ denotes computational indistinguishability.

Proof. We first show that

$$\mathcal{D}_\lambda := \left\{ \begin{pmatrix} g^{k_1\cdot\gamma_1} & \dots & g^{k_t\cdot\gamma_1} \\ \vdots & \ddots & \vdots \\ g^{k_1\cdot\gamma_W} & \dots & g^{k_t\cdot\gamma_W} \\ g^{k_1} & \dots & g^{k_t} \end{pmatrix} \mid \gamma \xleftarrow{\$} (\mathbb{Z}_q^*)^W, k \xleftarrow{\$} (\mathbb{Z}_q^*)^t \right\}_\lambda \overset{c}{\approx} \text{Uniform} \left(\mathbb{G}_q^{(W+1)\times t}\right)_\lambda$$

By DDH, the tuple $\left(g, g^{k_1}, g^{\gamma_1}, g^{k_1\cdot\gamma_1}\right)$ is computationally indistinguishable from $\left(g, g^{k_1}, g^{\gamma_1}, R_{1,1}\right)$ where $R_{1,1}$ is a random group element. Moreover, given g, g^{k_1}, and g^{γ_1}, an algorithm can sample $k_2, \dots, k_t$ and $\Gamma_2, \dots, \Gamma_W$, and compute the rest of the matrix efficiently. This shows that

$$\mathcal{D}_\lambda \overset{c}{\approx} \left\{ \begin{pmatrix} R_{1,1} & g^{k_2\cdot\gamma_1} & \dots & g^{k_t\cdot\gamma_1} \\ g^{k_2\cdot\gamma_1} & g^{k_2\cdot\gamma_2} & \dots & g^{k_t\cdot\gamma_1} \\ \vdots & \vdots & \ddots & \vdots \\ g^{k_1\cdot\gamma_W} & g^{k_2\cdot\gamma_W} & \dots & g^{k_t\cdot\gamma_W} \\ g^{k_1} & g^{k_2} & \dots & g^{k_t} \end{pmatrix} \mid \gamma \xleftarrow{\$} R_{1,1} \xleftarrow{\$} \mathbb{G}_q, (\mathbb{Z}_q^*)^W, k \xleftarrow{\$} (\mathbb{Z}_q^*)^t \right\}_\lambda$$

Following the same arguments, we can replace the matrix's entries one by one with random group elements, and hence

$$\begin{aligned} \mathcal{D}_\lambda &\overset{c}{\approx} \left\{ \begin{pmatrix} R_{1,1} & \dots & R_{1,t} \\ \vdots & \ddots & \vdots \\ R_{W,1} & \dots & R_{W,t} \\ g^{k_1} & \dots & g^{k_t} \end{pmatrix} \mid R \xleftarrow{\$} \mathbb{G}_q^{W\times t}, k \xleftarrow{\$} (\mathbb{Z}_q^*)^t \right\}_\lambda \\ &= \left\{ \begin{pmatrix} R_{1,1} & \dots & R_{1,t} \\ \vdots & \ddots & \vdots \\ R_{W,1} & \dots & R_{W,t} \\ r_1 & \dots & r_t \end{pmatrix} \mid R \xleftarrow{\$} \mathbb{G}_q^{W\times t}, r \xleftarrow{\$} \mathbb{G}_q^t \right\}_\lambda \end{aligned}$$

Therefore,

$$\begin{aligned} &\left\{ M_{\gamma,k,b} \mid \gamma \xleftarrow{\$} (\mathbb{Z}_q^*)^W, k \xleftarrow{\$} (\mathbb{Z}_q^*)^t \right\}_\lambda \\ &\overset{c}{\approx} \left\{ \begin{pmatrix} R_{1,1}\cdot r_1^{b_{1,1}} & \dots & R_{1,t}\cdot r_t^{b_{1,t}} \\ \vdots & \ddots & \vdots \\ R_{W,1}\cdot r_1^{b_{W,1}} & \dots & R_{W,t}\cdot r_t^{b_{W,t}} \end{pmatrix} \mid R \xleftarrow{\$} \mathbb{G}_q^{W\times t}, r \xleftarrow{\$} \mathbb{G}_q^t \right\}_\lambda = \text{Uniform}\left(\mathbb{G}_q^{W\times t}\right)_\lambda \end{aligned}$$

□

Lemma C.3 (DDH-based circular correlation robustness)**.** *Assume that H is a random oracle that outputs a $\lambda+1$ bits string and $H_{\mathbb{Z}}$ is a random oracle that outputs an element of $\mathbb{Z}_q^*$. Consider a prime-order group $\mathbb{G}_q$ with generator g. Let $\mathsf{sk} \in \{0,1\}^\lambda$, $\Gamma \in (\mathbb{Z}_q^*)^{W_{\max}}$, and $k \in (\mathbb{Z}_q^*)^t$ be secret keys sampled uniformly at random, and $\Delta = \mathsf{sk} \parallel 1$. Define two sets of oracles as follows,*
$\mathcal{O}^{\Delta,\Gamma,k} = \{\mathcal{O}^{\Delta}_{\text{bool}}, \mathcal{O}^{\Delta,k}_{\text{switch}}, \mathcal{O}^{\Delta,\Gamma}_{\text{grp}}, \mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}, \mathcal{O}^{\Gamma,k}_{\text{ddh}}\}$ *where*

- $\mathcal{O}^{\Delta}_{\text{bool}}(X,\nu,L) = H(X \oplus \Delta, \nu) \oplus L(\Delta)$ *where X is a $\lambda+1$ bits string, ν is a nonce, and $L \in \mathcal{L}$ is a linear function from $\{0,1\}^{\lambda+1}$ to $\{0,1\}^{\lambda+1}$.*[11]

[11] [RR21] sets the output length of the oracle to be $\lambda/2$ bits due to their slicing technique. We use a longer output for simplicity, since one can always pad the extra bits with 0 in the linear function L and truncate them when using the output of the oracle.

- $\mathcal{O}^{\Delta,k}_{\text{switch}}(X,\nu,\tau_1,\tau_2) = H_{\mathbb{Z}}(X \oplus \Delta,\nu)\cdot(k_{\tau_1}\cdot k_{\tau_2}^{-1})$ *where* X *is a* $\lambda+1$ *bits string,* ν *is a nonce, and* $b \in \{0,1\}$.
- $\mathcal{O}^{\Delta,\Gamma}_{\text{grp}}(X,\nu,i,r) = H_{\mathbb{Z}}(X \oplus \Delta,\nu)\cdot(1+\Gamma_i^{-1})^r$, *where* X *is a* $\lambda+1$ *bits string,* ν *is a nonce, and* $r \in \{-1,1\}$.
- $\mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}(X,i,r,\nu,b) = H(X^{(1+\Gamma_i^{-1})^r},\nu)\oplus b\cdot\Delta$ *where* $X \in \mathbb{G}_q$, $r \in \{-1,1\}$, *and* $b \in \{0,1\}$.
- $\mathcal{O}^{\Gamma,k}_{\text{ddh}}(i,\tau,b) = g^{k_\tau\cdot(\Gamma_i+b)}$ *where* $i \in [W_{\max}]$, $\tau \in [t]$, *and* $b \in \{0,1\}$.

$\mathcal{R} = \{\mathcal{R}_{\text{bool}}, \mathcal{R}_{\text{switch}}, \mathcal{R}_{\text{grp}}, \mathcal{R}_{\text{ungrp}}, \mathcal{R}_{\text{ddh}}\}$ *where*

- $\mathcal{R}_{\text{switch}}(X,\nu,\tau_1,\tau_2)$ *and* $\mathcal{R}_{\text{grp}}(X,\nu,i,r)$ *are truly random functions to* $\mathbb{Z}_q^*$.
- $\mathcal{R}_{\text{bool}}(X,i,r,\nu,b)$ *and* $\mathcal{R}_{\text{ungrp}}(X,i,r,\nu,b)$ *are truly random functions to* $\{0,1\}^{\lambda+1}$.
- $\mathcal{R}_{\text{ddh}}$ *is a truly random function to* $\mathbb{G}_q$.

A sequence of queries to the oracles in $\mathcal{O}^{\Delta,\Gamma,k}$ *is legal if the same nonce* ν *is never called twice across the oracles, and moreover, each pair* (i,τ) *is never queried with different values of* b *to* $\mathcal{O}^{\Gamma,k}_{\text{ddh}}$. *If the decisional Diffie-Hellman assumption holds in* $\mathbb{G}_q$ *with security parameter* λ, *then no poly-time adversary* $\mathcal{A}$ *issuing legal queries can distinguish* $\mathcal{O}^{\Delta,\Gamma,k}$ *and* $\mathcal{R}$ *except with negligible probability. I.e.:*

$$\left|\Pr_{\Delta,\Gamma,k}\left[\mathcal{A}^{H,\,H_{\mathbb{Z}},\,\mathcal{O}^{\Delta,\Gamma,k}}\left(1^\lambda\right)=1\right]-\Pr_{\mathcal{R}}\left[\mathcal{A}^{H,\,H_{\mathbb{Z}},\,\mathcal{R}}\left(1^\lambda\right)=1\right]\right| < \mathsf{negl}(\lambda)$$

Proof. We prove the lemma by constructing the following sequence of hybrids:

- **Real:** The adversary interacts with H, $H_{\mathbb{Z}}$, and $\mathcal{O}^{\Delta,\Gamma,k}$.
- **Hyb$_1$:** Compared to **Real**, Hyb$_1$ adds a check on the first argument of each query to oracles H, $H_{\mathbb{Z}}$, $\mathcal{O}^{\Delta}_{\text{bool}}$, $\mathcal{O}^{\Delta,k}_{\text{switch}}$, $\mathcal{O}^{\Delta,\Gamma}_{\text{grp}}$, and $\mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}$. Hyb$_1$ aborts if the first argument is $X\oplus\Delta$ or $X^{(1+\Gamma_i^{-1})^r}$ for $r=\pm1$, where X is the first argument of a previous query to any oracle.
- **Hyb$_2$:** We replace the output of $\mathcal{O}^{\Delta,k}_{\text{switch}}$, $\mathcal{O}^{\Delta,\Gamma}_{\text{grp}}$, $\mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}$ in Hyb$_1$ with the output of $\mathcal{R}_{\text{switch}}$, $\mathcal{R}_{\text{grp}}$, and $\mathcal{R}_{\text{ungrp}}$ respectively.
- **Hyb$_3$:** We replace the output of $\mathcal{O}^{\Gamma,k}_{\text{ddh}}$ in Hyb$_2$ with the output of $\mathcal{R}_{\text{ddh}}$.
- **Ideal:** Finally, we remove the check on the first argument of the queries, and get the ideal world view where the adversary interacts with H, $H_{\mathbb{Z}}$, and $\mathcal{R}$.

Next, we prove the indistinguishability of hybrids in a reverse order.

- Ideal $\approx$ Hyb$_3$: Since Hyb$_3$ does not involve any secret key Δ or Γ, the probability that a poly-time adversary input $X\oplus\Delta$ or $X^{(1+\Gamma_i^{-1})^r}$ for $r=\pm1$ is negligible. Therefore, Hyb$_3$ aborts with negligible probability.
- Hyb$_3$ $\approx$ Hyb$_2$: By Lemma C.2, $\mathcal{O}^{\Gamma,k}_{\text{ddh}}$ is computationally indistinguishable from $\mathcal{R}_{\text{ddh}}$.

- $\text{Hyb}_2 \approx \text{Hyb}_1$: First, consider the replacement of $\mathcal{O}^{\Delta}_{\text{bool}}$ with $\mathcal{R}_{\text{bool}}$. If $X \oplus \Delta$ is input as the first argument and X is queried before, both Hyb_1 and Hyb_2 abort, and are hence indistinguishable. Otherwise, $H(X \oplus \Delta, \nu)$ is identical to a random function, and moreover, serves as a one-time pad for $L(\Delta)$, since the nonce ν can only be queried once. Through the same argument, we can replace $\mathcal{O}^{\Delta,k}_{\text{switch}}$ and $\mathcal{O}^{\Delta,\Gamma}_{\text{grp}}$. Similarly, for $\mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}$, if $X^{(1+\Gamma_i^{-1})^r}$ for $r = \pm 1$ is input as the first parameter, both Hyb_1 and Hyb_2 abort. Otherwise, $H_{\mathbb{Z}}(X^{(1+\Gamma_i^{-1})^r})$ is indistinguishable from a random function and serves as a one time pad for $\mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}$.
- $\text{Hyb}_1 \approx \text{Real}$: So far, we have shown $\text{Hyb}_1 \approx \text{Ideal}$. Therefore, the probability that Hyb_1 aborts must be negligible. Since the only difference between Hyb_1 and Real is that Hyb_1 may abort on certain inputs, $\text{Hyb}_1 \approx \text{Real}$.

□

Theorem C.4 (Security of the garbling scheme from RO)**.** *Assuming DDH and Random Oracles, the construction in Figure 13 and 14 is a secure garbling scheme for the family of SIMD tri-state circuits that are well-formed, w.r.t. the leakage function* $\mathsf{controls}(\cdot,\cdot)$ *as defined below: given a well-formed SIMD tri-state circuit* $\mathcal{C}$ *and input* inp*,* $\mathsf{controls}(\mathcal{C}, \mathsf{inp})$ *outputs the values on all the control wires of the Switch gates when evaluating* $\mathcal{C}$ *over* inp*.*

Proof. We construct the following experiments:

- **Real:** The real experiment outputs the garbled circuit $\widetilde{\mathcal{C}}$ and the active labels of the input wires $\widetilde{\mathsf{inp}}$, as defined in Definition 4.2.
- **Hyb:** The Hyb experiment simulates $\widetilde{\mathcal{C}}$ and $\widetilde{\mathsf{inp}}$ with the hybrid simulator $\mathsf{Sim}_{\text{Hyb}}$ (Figure 15). $\mathsf{Sim}_{\text{Hyb}}$ is given $\mathbb{G}_q$ in the real experiment and can access the oracle $\mathcal{O}^{\Delta,\Gamma,k}$ as defined in Lemma C.3, where Δ and Γ are the global secret keys used by Garble in the real experiment, and $k = [k_1, \ldots, k_t]$ is randomly sampled from $\left(\mathbb{Z}_q^*\right)^t$, with t being the total number of cables. Moreover, we let $\mathsf{Sim}_{\text{Hyb}}$ access the value val_x of each wire in the real experiment. By Definition 3.1, val_x is uniquely determined by the circuit's input. Our $\mathsf{Sim}_{\text{Hyb}}$ simulator calls the $\mathsf{Hybrid1}$ simulator in the prior work [RR21] to generate the garbled circuit materials of all the boolean gates, as well as the active labels of the gates' output wires.
- **Ideal:** The ideal experiment replaces $\mathsf{Sim}_{\text{Hyb}}$ with the ideal simulator Sim (Figure 16). Specifically, we replace all the oracle calls to $\mathcal{O}^{\Delta,\Gamma,k}$ in with random sampling. The replacement also reduces the $\mathsf{Hybrid1}$ simulator in [RR21] to their privacy simulator $\mathcal{S}_{\text{priv}}$. At this stage, Sim only uses the values of the output wires and the control wires, so we replace val with out and ctrl in the simulator's input.

We now show that the real and the ideal views are computationally indistinguishable.

- **Real = Hyb:** First, the active labels of the input wires sampled by $\mathsf{Sim}_{\text{Hyb}}$ is indistinguishable from Real, since each active label in $\widetilde{\mathsf{inp}}$ is masked by an independently sampled zero label. Then, we show that $\widetilde{\mathsf{Gates}}$ are also indistinguishable in the Real and Hyb for each type of gates.

  For the Boolean gates, we utilize the hybrid simulator $\mathsf{Hybrid1}$ in [RR21] to generate the garbled materials and the active labels of the output wires with identical joint distribution as in Real. This is feasible because our simulator $\mathsf{Sim}_{\text{Hyb}}$ has access to the values of all the wires, and moreover, the oracle $\mathcal{O}^{\Delta}_{\text{bool}}$ needed by $\mathsf{Hybrid1}$.

For each Group gate, we call the oracle $\mathcal{O}^{\Delta,\Gamma}_{\text{grp}}$ to compute the inactive row of the gate's garbled truth table, and we permute the two rows according the least significant bit of the active input label. It is straightforward to check that the rows are correctly permuted for both the case $\mathsf{val}_x = 0$ and $\mathsf{val}_x = 1$.

For each Ungroup gate $y \xleftarrow{\text{ungrp}} x_i$, we similarly query the oracle $\mathcal{O}^{\Delta,\Gamma}_{\text{ungrp}}$ to compute $h_{1-\mathsf{val}_{x_i}} \oplus \Delta$ and $\mu_{1-\mathsf{val}_{x_i}}$ in the real experiment. Additionally, the hybrid simulator samples β' to substitute $\beta \oplus \mathsf{val}_{x_i}$. After the substitution, we can check that both the garbled material and the output wire's label are computed identically to the real experiments.

Finally, we need to simulate the labels of each subwire and the garbled material of the Switch gates. A main challenge here is that we cannot directly reveal the cable keys to the simulator as they will undermine the pseudo-randomness of the active subwire labels. On the other hand, the non-spanning Switch gates cannot be garbled merely with the subwire labels, since this would require discrete logarithm in $\mathbb{G}_q$.

To resolve this dilemma, we factorize each cable key into two parts: $\mathsf{K}_{\mathbf{x}} = k_{\tau_{\mathbf{x}}} \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}}$, where $k_{\tau_{\mathbf{x}}}$ is called the union-wise key and $\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}$ is called the relative key of cable $\mathbf{x}^W$. We require that two cables connected by active Switch gates share the same union-wise key. In other words, the cables are assigned to unions such that cables connected by active Switch gates are grouped into the same union, and we let $\tau_{\mathbf{x}}$ denote the index of the union that contains the cable $\mathbf{x}^W$. The simulator directly possesses the relative keys, but the union-wise keys are only embedded in the oracles $\mathcal{O}^{\Delta,k}_{\text{switch}}$ and $\mathcal{O}^{\Gamma,k}_{\text{ddh}}$.

Note that given the relative label and the oracle $\mathcal{O}^{\Gamma,k}_{\text{ddh}}$, the simulator can compute the active label of a subwire x_i as

$$\mathsf{L}_{x_i} = g^{\mathsf{K}_{\mathbf{x}} \cdot (\Gamma_i + b)} = g^{k_{\tau_x} \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \cdot (\Gamma_i + b)} = \left(\mathcal{O}^{\Gamma,k}_{\text{ddh}}(\tau_x,\, i,\, \mathsf{val}_{x_i})\right)^{\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}}.$$

The simulator first samples the relative key $\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}$ for each root cable $\mathbf{x}^W$, which serves as a one time pad and ensures that the simulated cable keys have the same distribution as in Real. Then, the simulator solves the relative keys of the remaining cables using the constraints of the spanning Switch gates. If a spanning Switch gate $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$ is active, then $\mathbf{x}^W$ and $\mathbf{y}^W$ share the same union-wise labels, and $\mathsf{val}_c = 0$. Therefore, the constraint $\mathsf{K}_{\mathbf{y}} = H_{\mathbb{Z}}(\mathsf{L}_{c=0},\, \mathsf{gid}) \cdot \mathsf{K}_{\mathbf{x}}$ can be simply rewritten as $\mathsf{K}^{\mathsf{r}}_{\mathbf{y}} = H_{\mathbb{Z}}(\mathsf{L}_c,\, \mathsf{gid}) \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}}$. If the spanning Switch gate is inactive, then we instead have the constraint

$$\frac{\mathsf{K}^{\mathsf{r}}_{\mathbf{y}}}{\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}} = \frac{\mathsf{K}_{\mathbf{y}}}{\mathsf{K}_{\mathbf{x}}} \cdot \frac{k_{\tau_{\mathbf{x}}}}{k_{\tau_{\mathbf{y}}}} = H_{\mathbb{Z}}(\mathsf{L}_c \oplus \Delta,\, \mathsf{gid}) \cdot \frac{k_{\tau_{\mathbf{x}}}}{k_{\tau_{\mathbf{y}}}} = \mathcal{O}^{\Delta,k}_{\text{switch}}(\mathsf{L}_c,\, \mathsf{gid},\, \tau_{\mathbf{x}},\, \tau_{\mathbf{y}})$$

With all the relative labels, the simulator can then compute the garbled material of the Switch gates. For each spanning Switch gate, we only need to simulate the permutation bit of the zero label of the control wire, that is, $\mathsf{LSB}(\mathsf{L}_c) \oplus \mathsf{val}_c$.

For the non-spanning Switch gates, we also need to simulate the second term $R = H_{\mathbb{Z}}(\mathsf{L}_{c=0},\, \mathsf{gid}) \cdot \mathsf{K}_{\mathbf{x}} \cdot (\mathsf{K}_{\mathbf{y}})^{-1}$. If the Switch gate is active, then $R = H_{\mathbb{Z}}(\mathsf{L}_c,\, \mathsf{gid}) \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \cdot (\mathsf{K}^{\mathsf{r}}_{\mathbf{y}})^{-1}$. Otherwise, we have

$$R = H_{\mathbb{Z}}(\mathsf{L}_c \oplus \Delta,\, \mathsf{gid}) \cdot \frac{\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}}{\mathsf{K}^{\mathsf{r}}_{\mathbf{y}}} \cdot \frac{k_{\tau_{\mathbf{x}}}}{k_{\tau_{\mathbf{y}}}} = \frac{\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}}{\mathsf{K}^{\mathsf{r}}_{\mathbf{y}}} \cdot \mathcal{O}^{\Delta,k}_{\text{switch}}(\mathsf{L}_c,\, \mathsf{gid},\, \tau_{\mathbf{x}},\, \tau_{\mathbf{y}}).$$

Finally, $\widetilde{\mathsf{Out}}$ can be simulated the same way as the spanning Switch gates. Namely, $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{LSB}(\mathsf{L}_y) \oplus \mathsf{val}_y$ for each i-th output wire y.

- **Hyb $\overset{c}{\approx}$ Sim:** First, the queries to $\mathcal{O}^{\Delta,\Gamma,k}$ in Figure 15 are legal as defined in Lemma C.3 since

  1. The nonce ν always relates to the unique identifier of the gate and is never repeated, and
  2. By well-formedness, if two cables $\mathbf{x}^W$ and $\mathbf{y}^W$ are in the same union, then for every offset i, we have $\mathsf{val}_{x_i} = \mathsf{val}_{y_i}$. Otherwise, we can apply the evaluation rules of the active Switch gates to alter the value of either x_i or y_i, which contradicts the requirement of unique total state. Therefore, the simulator never queries $\mathcal{O}^{\Gamma,k}_{\mathrm{ddh}}(i,\tau,b)$ with the same pair (i,τ) but different values of b.

  By Lemma C.3, the oracles are computationally indistinguishable from the random functions, so we can substitute the oracles with random sampling in Figure 15, which also get rids of all the access to wire values except for the output and control wires. Hence, we obtain the ideal simulator Sim in Figure 16 that produces indistinguishable views from $\mathsf{Sim}_{\mathrm{Hyb}}$.

Combining the above hybrid arguments, we conclude that the real and ideal views are computationally indistinguishable, and therefore the garbling scheme is secure. □

D Additional Concrete Optimizations

In our implementation of PicoGRAM, we incorporate several optimizations to enhance concrete performance beyond the theoretical asymptotics. These refinements are detailed below.

D.1 Optimizing Cryptographic Primitives

We refine the cryptographic primitives in our garbling scheme to minimize both communication and computation overhead.

Half gates. While we employ the techniques from [RR21] to reduce the communication to $1.5\lambda+5$ bits per AND gate, when either the garbler or evaluator knows one input wire's value, we can further use the half-gate technique from [ZRE15] to reduce the cost to λ bits.

Homomorphic addition of subwires. We observe that the labels of two subwires within the same cable can be added homomorphically, similar to the Free-XOR technique [KS08]:

$$\mathsf{L}_{x_i=a} \cdot \mathsf{L}_{x_j=b} = g^{\mathsf{K}_{\mathbf{x}}\cdot(\gamma_i+a)} \cdot g^{\mathsf{K}_{\mathbf{x}}\cdot(\gamma_j+b)} = g^{\mathsf{K}_{\mathbf{x}}\cdot(\gamma_i+\gamma_j+(a+b))}.$$

Since the addition is performed in $\mathbb{Z}_p$, we need to ensure that the sum $a+b$ remains small enough so that the wire can be decoded efficiently.

In PicoGRAM, we apply this technique to aggregate the return values from different levels of the ORAM tree during each access. Specifically, at each tree level, we skip Ungrouping the output from the child node, and instead directly add it with the data element read out from the current level in cable form. Since only one of the levels contains the data element, the value of each subwire is always bounded in $\{0,1\}$. This optimization reduces both the communication cost and the garbler's computational overhead associated with the Ungroup gate.

$\mathsf{Sim}_{\mathrm{Hyb}}(1^\lambda, \mathcal{C}, \mathsf{val}, \mathcal{O}^{\Delta,\Gamma,k})$:

- For each input wire x of $\mathcal{C}$, sample the active label $\mathsf{L}_x \xleftarrow{\$} \{0,1\}^{\lambda+1}$.
- Assign cables to unions such that cables connected by active Switch gates are in the same union. Let $\tau_{\mathbf{x}}$ denote the index of the union containing $\mathbf{x}^W$.
- For each root cable $\mathbf{x}^W$, sample a "relative" key $\mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \xleftarrow{\$} \mathbb{Z}_q^*$. Let the active label of each subwire x_i be
$$\mathsf{L}_{x_i} \leftarrow \left(\mathcal{O}^{\Gamma,k}_{\mathrm{ddh}}(\tau_x,\, i,\, \mathsf{val}_{x_i})\right)^{\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}}$$
- For each Ungroup gate $y \xleftarrow{\mathrm{ungrp}} x_i$, sample $\beta' \xleftarrow{\$} \{0,1\}$ and let
$$h \leftarrow H(\mathsf{L}_{x_i}, \mathsf{gid}\|\text{“}h\text{”}),\ h' \leftarrow \mathcal{O}^{\Delta,\Gamma}_{\mathrm{ungrp}}\left(\mathsf{L}_{x_i}, i, (-1)^{\mathsf{val}_{x_i}}, \mathsf{gid}\|\text{“}h\text{”}, 1\right),$$
$$\mu \leftarrow H(\mathsf{L}_{x_i}, \mathsf{gid}\|\text{“}\mu\text{”}),\ \mu' \leftarrow \mathcal{O}^{\Delta,\Gamma}_{\mathrm{ungrp}}(\mathsf{L}_{x_i}, i, (-1)^{\mathsf{val}_{x_i}}, \mathsf{gid}\|\text{“}\mu\text{”}, 0).$$
$$\begin{aligned} &\text{If } \beta' = 0, && \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (h \oplus h',\, \mu), \quad \mathsf{L}_y \leftarrow h; \\ &\text{Else,} && \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (h \oplus h',\, \mu'), \quad \mathsf{L}_y \leftarrow h'. \end{aligned}$$
- Garble all the Boolean gates $z \leftarrow f(x,y)$ with the oracle $\mathcal{O}_{bool}$ following the Hybrid1 scheme in [RR21]. Store the active label L_x of every independent wire x.
- Solve the relative keys of the remaining cables so that for each spanning Switch gate $\mathbf{x}^W \overset{c}{\text{—}\!\!/\!\!\text{—}} \mathbf{y}^W$:
$$\text{If } \mathsf{val}_c = 0, \text{ then } \mathsf{K}^{\mathsf{r}}_{\mathbf{y}} = H_{\mathbb{Z}}(\mathsf{L}_c,\, \mathsf{gid}) \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}}. \qquad \text{Else, } \mathsf{K}^{\mathsf{r}}_{\mathbf{y}} = \mathcal{O}^{\Delta,k}_{\mathrm{switch}}(\mathsf{L}_c,\, \mathsf{gid},\, \tau_{\mathbf{x}},\, \tau_{\mathbf{y}}) \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}}$$
- Set the gate material of each Switch gate $\mathbf{x}^W \overset{c}{\text{—}\!\!/\!\!\text{—}} \mathbf{y}^W$:
$$\text{If it is a spanning Switch gate, } \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \mathsf{LSB}(\mathsf{L}_c) \oplus \mathsf{val}_c.$$
$$\text{If not,} \quad \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left(\mathsf{LSB}(\mathsf{L}_c) \oplus \mathsf{val}_c,\ h \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \cdot (\mathsf{K}^{\mathsf{r}}_{\mathbf{y}})^{-1}\right)$$
where $h \leftarrow H(\mathsf{L}_c, \mathsf{gid})$ if $\mathsf{val}_c = 0$, and $h \leftarrow \mathcal{O}^{\Delta,k}_{\mathrm{switch}}(\mathsf{L}_c, \mathsf{gid}, \tau_{\mathbf{x}}, \tau_{\mathbf{y}})$ otherwise.
- For each Group gate $y_i \xleftarrow{\mathrm{grp}} x$, let
$$h \leftarrow H_{\mathbb{Z}}(\mathsf{L}_x, \mathsf{gid}),\ h' \leftarrow \mathcal{O}^{\Delta,\Gamma}_{\mathrm{grp}}\left(\mathsf{L}_{x_i}, \mathsf{gid}, i, (-1)^{\mathsf{val}_{x_i}}\right),\ e \leftarrow (\mathsf{L}_{y_i})^h,\ e' \leftarrow (\mathsf{L}_{y_i})^{h'}.$$
$$\text{If } \mathsf{LSB}(\mathsf{L}_x) = 0,\ \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (e,\, e'); \quad \text{Else, } \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (e',\, e).$$
- Let $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{LSB}(\mathsf{L}_y) \oplus \mathsf{val}_y$ for each i-th output wire y.
- Return $\widetilde{\mathcal{C}} = \left(\mathcal{C},\, \mathbb{G}_q,\, \widetilde{\mathsf{Gates}},\, \widetilde{\mathsf{Out}}\right)$

Figure 15: The hybrid simulator $\mathsf{Sim}_{\mathrm{Hyb}}$ has access to the oracle $\mathcal{O}^{\Delta,\Gamma,k}$ and the plaintext value of every wire in the unique total state given by the input. gid is a unique nonce of the gate being discussed.

$\mathsf{Sim}(1^\lambda, \mathcal{C}, \mathsf{out}, \mathsf{ctrl})$: *// Select a prime-order group $\mathbb{G}_q$ with security parameter λ*

- For each input wire x of $\mathcal{C}$, sample the active label $\mathsf{L}_x \xleftarrow{\$} \{0,1\}^{\lambda+1}$.
- Assign cables to unions such that cables connected by active Switch gates are in the same union. Let $\tau_{\mathbf{x}}$ denote the index of the union containing $\mathbf{x}^W$.
- For each root cable $\mathbf{x}^W$, sample a "relative" key $\mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \xleftarrow{\$} \mathbb{Z}_q^*$. Sample $U \xleftarrow{\$} (\mathbb{G}_q)^{t \times W_{\max}}$ and let the active label of each subwire x_i be
$$\mathsf{L}_{x_i} \leftarrow (U_{\tau,i})^{\mathsf{K}^{\mathsf{r}}_{\mathbf{x}}}$$
- For each Ungroup gate $y \xleftarrow{\text{ungrp}} x_i$, sample $\beta' \xleftarrow{\$} \{0,1\}$ and let
$$h \leftarrow H(\mathsf{L}_{x_i}, \mathsf{gid} \| \text{“}h\text{”}),\ h' \xleftarrow{\$} \{0,1\}^{\lambda+1},$$
$$\mu \leftarrow H(\mathsf{L}_{x_i}, \mathsf{gid} \| \text{“}\mu\text{”}),\ \mu' \xleftarrow{\$} \{0,1\}^{\lambda+1}.$$
$$\begin{aligned} &\text{If } \beta' = 0, && \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (h \oplus h',\ \mu), \quad \mathsf{L}_y \leftarrow h; \\ &\text{Else}, && \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (h \oplus h',\ \mu'), \quad \mathsf{L}_y \leftarrow h'. \end{aligned}$$
- Garble all the Boolean gates $z \leftarrow f(x,y)$ using the simulator $\mathcal{S}_{\text{priv}}$ in [RR21]. Store the active label L_x of every independent wire x.
- Solve the relative keys of the remaining cables so that for each spanning Switch gate $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$:
$$\text{If } \mathsf{ctrl}_c = 0, \text{ then } \mathsf{K}^{\mathsf{r}}_{\mathbf{y}} = H_{\mathbb{Z}}(\mathsf{L}_c, \mathsf{gid}) \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}}. \qquad \text{Else, } \mathsf{K}^{\mathsf{r}}_{\mathbf{y}} = R \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \text{ where } R \xleftarrow{\$} \mathbb{Z}_q^*.$$
- Set the gate material of each Switch gate $\mathbf{x}^W \overset{c}{\text{---}} \mathbf{y}^W$:
$$\text{If it is a spanning Switch gate, } \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \mathsf{LSB}(\mathsf{L}_c) \oplus \mathsf{val}_c.$$
$$\text{If not, } \quad \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow \left(\mathsf{LSB}(\mathsf{L}_c) \oplus \mathsf{val}_c,\ h \cdot \mathsf{K}^{\mathsf{r}}_{\mathbf{x}} \cdot (\mathsf{K}^{\mathsf{r}}_{\mathbf{y}})^{-1}\right)$$
where $h \leftarrow H(\mathsf{L}_c, \mathsf{gid})$ if $\mathsf{val}_c = 0$, and $h \xleftarrow{\$} \mathbb{Z}_q^*$ otherwise.
- For each Group gate $y_i \xleftarrow{\text{grp}} x$, let
$$h \leftarrow H_{\mathbb{Z}}(\mathsf{L}_x, \mathsf{gid}),\ h' \xleftarrow{\$} \mathbb{Z}_q^*,\ e \leftarrow (\mathsf{L}_{y_i})^h,\ e' \leftarrow (\mathsf{L}_{y_i})^{h'}.$$
$$\text{If } \mathsf{LSB}(\mathsf{L}_x) = 0,\ \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (e, e'); \quad \text{Else, } \widetilde{\mathsf{Gates}}[\mathsf{gid}] \leftarrow (e', e).$$
- Let $\widetilde{\mathsf{Out}}[i] \leftarrow \mathsf{LSB}(\mathsf{L}_y) \oplus \mathsf{val}_y$ for each i-th output wire y.
- Return $\widetilde{\mathcal{C}} = \left(\mathcal{C}, \mathbb{G}_q, \widetilde{\mathsf{Gates}}, \widetilde{\mathsf{Out}}\right)$

Figure 16: The simulator Sim of our garbling scheme based on random oracle. gid is a unique nonce of the gate being discussed. Compared to $\mathsf{Sim}_{\text{Hyb}}$, we replace the oracles $\mathcal{O}^{\Delta,\Gamma,k}$ with random sampling due to Lemma C.3.

D.2 Optimizing and Parallelizing Group Operations

We apply several techniques to further reduce the computation time of group operations.

Precomputation. We optimize exponentiation on the generator using window-based methods [Coh05], which reduce the number of multiplications required by precomputing a table of values for small exponents and then combining them to obtain larger powers. This optimization also improves the computational complexity of the garbler asymptotically by an $O(\log N)$ factor. Specifically, the garbler can precompute a table for all exponents $x \cdot N^y$ on the generator for $x \in [N-1]$ and $y \in [\frac{\log q}{\log N}]$. When garbling the circuit, the garbler can compute a generator's power more efficiently by first converting the exponent to base N and then multiplying $O(\frac{\lambda}{\log N})$ corresponding entries in the precomputed table. Unfortunately, the evaluator cannot benefit from this precomputation because exponentiation is always performed on random group elements. In our implementation, we use the OpenSSL library for elliptic curve operations, which already employs optimized precomputation.

Faster modular operations. We accelerate multiplication in $\mathbb{Z}_q^*$ using the Montgomery method [Mon85], which transforms numbers into a representation that allows for faster modular multiplication. In our garbled stacks, exponents remain in the Montgomery domain and are converted back to the standard domain only when processing an Ungroup gate. For modular inversion, we apply the Montgomery batch inversion technique [MS04], which replaces n individual inversions with a single inversion and $3n-3$ multiplications. As a result, the garbler computes only one inversion to garble the stack, and the evaluator performs one inversion per stack access, regardless of word width.

Parallelization. Finally, our implementation parallelizes elliptic curve (EC) operations in the Group and Ungroup gates to accelerate computation. On our test platform, a scalar multiplication on a 256-bit curve requires approximately 120,000 cycles. Since each task is large enough, we can efficiently distribute the work across multiple cores. It is difficult to achieve similar optimization in previous practical GRAM constructions [HKO22, PLS23, HKO23], as those schemes rely only on symmetric key cryptography, which is already extremely fast on modern CPUs—for example, computing a 128-bit hash with fixed-key AES takes only about 40 CPU cycles on our platform. While the garbler's computation is highly parallelizable, the evaluator's operations are mostly sequential, so they must rely on parallel ORAM constructions [HCS17, AKLS23], which introduces additional overhead to both communication and computation costs.

D.3 Circuit-level Optimizations

We also perform several optimizations to reduce the number of gates in the ORAM circuit.

Simplifying computation of controls. Since our garbling scheme based on the random oracle only requires the circuit to be well-formed (rather than strictly well-formed), we replace the eager-prefix-sum circuit in Section 6 with a naïve prefix-sum circuit in our concretely efficient implementation.

Sharing gates across stacks. When instantiating stacks to connect a parent node with its child nodes, we can reuse the same set of Group and Ungroup gates at the parent's end. Rather than sharing the independent wires $x_{\tau,i}$ in Figure 10 as inputs and outputs to multiple stacks, we convert the wires into cables with a single set of Group and Ungroup gates, and then reuse the cables for multiple stacks. Moreover, each pair of compaction and distribution stacks can also share the same internal SIMD stack, since they receive the same control inputs $c_1, \ldots, c_t$, and our Switch gates support bidirectional partial evaluation.

Optimizing bucket circuitry. We merge the "read" and "evict" sub-circuits in each Circuit ORAM bucket [WCS15]. The optimized circuit only needs to perform a single conditional swap for each element stored in the bucket at every timestep.

Efficient eviction on the read path. In addition to the deterministic evictions after every access to the Circuit ORAM [WCS15], we also perform an eviction on the read path. This eviction is slightly weaker than the regular eviction for two reasons:

1. The read path is random, and eviction on random paths has been shown to be slightly less effective than eviction in reverse-lexicographic order [WCS15];
2. No element is further evicted from a bucket if the bucket contains the element to be read during the access.

Nonetheless, we are able to implement this additional eviction with little extra communication cost, and it allows us to use smaller buckets and stashes while maintaining the same empirical failure probability.

D.4 Parameter Tuning.

Avoid SIMD garbling for small stacks. When a garbled stack is provisioned for a relatively small number of accesses, we utilize standard tri-state gates [HKO23] to construct the stack, thereby avoiding the computational overhead associated with Group and Ungroup gates. By adjusting a threshold on the number of accesses, we can balance communication and computation costs effectively.

Flattening the ORAM tree. We increase the fan-out of the Circuit ORAM tree from 2 to 4, so each parent node directly connects to 4 children through stacks. We maintain the same success probability by also increasing the bucket size. As the ORAM tree becomes shallower, we reduce the number of Group and Ungroup gates by half (assuming the previous optimization has been applied).

E Remarks on Concrete Performance of Baselines

E.1 Comparing NanoGRAM and TSC

Although Tri-state [HKO23] is asymptotically more efficient than NanoGRAM [PLS23], it incurs about twice the communication cost in our concrete evaluation. Below, we analyze the reasons for this discrepancy.

Both NanoGRAM and Tri-state GRAM operate in two phases: read and eviction. In the read phase, both schemes access $O(\log N)$ buckets and use stacks for routing. In Tri-state GRAM, the bucket size is constant, while NanoGRAM reads from buckets of size $O(\log N)$ (and an additional stash of size $\omega(\log N)$). However, since neither work provides an efficient stack instantiation, the performance difference is masked by the cost of the stack. Specifically, NanoGRAM is only 1.6× slower than Tri-state in the read phase at $N = 2^{16}$.

In the eviction phase, Tri-state evicts along two paths and requires additional stacks for routing, whereas NanoGRAM evicts buckets only at the end of their life cycle, avoiding the need for stacks. This makes NanoGRAM 10.2× faster in the eviction phase.

Two additional factors contribute to this performance gap. First, in Tri-state, a bucket can be invoked due to either a read or an eviction, so both circuits must be provisioned for the bucket at

every timestep. Since read and eviction require routing data along different directions, the stack must also be provisioned for both directions, resulting in a 2× overhead. NanoGRAM avoids this inefficiency since eviction only occurs at the end of the bucket's life cycle. Second, NanoGRAM offers more tunable parameters to balance the bucket and stack costs, whereas Tri-state cannot fully benefit from such balancing, partly because its bucket size is already constant.

Meanwhile, as our work significantly reduces the cost from garbled stacks, it becomes concretely more efficient to build PicoGRAM based on Tri-state, as NanoGRAM suffers higher communication from its Bucket ORAM circuitry [FNR+15, PLS23].

E.2 EpiGRAM and VISAs

EpiGRAM [HKO22] is another practical GRAM construction and has been implemented in the VISAs framework [YPHK23]. However, it has been both concretely and asymptotically outperformed by the subsequent NanoGRAM [PLS23], so we did not include it in our evaluation section.

Below, we list the amortized communication cost of each scheme under the largest benchmark presented in VISAs [YPHK23] (i.e., $W = 32$, $N = 8192$, $T = 23001$):

EpiGRAM (VISAs)	7.70 MB
NanoGRAM	6.71 MB
Tri-state	9.65 MB
Interactive (ORAM atop GC)	1.61 MB
PicoGRAM	2.00 MB

E.3 Additional Comparison with Interactive RAM-model 2PC

Impact of different parameters. Compared to PicoGRAM, the RAM-model 2PC baseline [WCS15, LWN+15] uses interactive protocols to achieve dynamic label translation, rather than garbled stack. This approach requires less communication (in bytes) and computation. However, it incurs a round-trip time (RTT) for each ORAM tree access. As shown in Section 7, PicoGRAM outperforms the interactive RAM-model 2PC construction by a factor of 2.93× when the RAM space is $N = 2^{16}$, the RTT is 100 ms, and the bandwidth is 200 Mbps. The speedup increases as RTT, bandwidth, or computational power increase. Conversely, the speedup decreases as the RAM space N grows, since the additional communication and computation of PicoGRAM scale with the ORAM tree depth, while the round-trip time for RAM-model 2PC remains constant.

Online versus offline. A key advantage of Garbled RAM over the interactive RAM-model 2PC is that both garbling and communication can be performed in a preprocessing phase, before the input is known. As a result, the online runtime depends only on the computation time of the evaluation algorithm. PicoGRAM remains secure under the standard semi-honest 2PC definition [Can01], where the adversary cannot adaptively choose inputs based on the garbled circuit. As shown in Section 7, we can reduce online time by replacing the SIMD gates with standard tri-state gates from prior works [HKO23,Hea24]. In this setting, we may also adopt the garbling scheme from [BHKO23], which is proven adaptively secure for tri-state circuits.

In contrast, interactive protocols allow circuits to be garbled on the fly, greatly reducing both initialization time and storage overhead for the parties. As a result, interactive protocols may be preferable for applications where inputs arrive incrementally in a streaming fashion. Additionally, if the RAM is pre-filled and the number of accesses $T \ll N$, only a small subset of ORAM tree paths are accessed. In such cases, interactive protocols can significantly outperform garbled RAM

schemes like PicoGRAM, as they do not require the garbler to provision the entire ORAM tree in advance.