2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00205

Efficient Actively Secure DPF and RAM-based 2PC with One-Bit Leakage

Wenhao Zhang
Northwestern University
wenhao.zhang@northwestern.edu

Xiaojie Guo
Nankai University
State Key Laboratory of Cryptology
xiaojie.guo@mail.nankai.edu.cn

Kang Yang*
State Key Laboratory of Cryptology
yangk@sklc.org

Ruiyu Zhu
rynzhu@gmail.com

Yu Yu*
Shanghai Jiao Tong University
Shanghai Qi Zhi Institute
yuyu@yuyu.hk

Xiao Wang*
Northwestern University
wangxiao@northwestern.edu

***Abstract*—Secure two-party computation (2PC) in the RAM model has attracted huge attention in recent years. Most existing results only support semi-honest security, with the exception of Keller and Yanai (Eurocrypt 2018) with very high cost. In this paper, we propose an efficient RAM-based 2PC protocol with active security and one-bit leakage.**

1) **We propose an actively secure protocol for distributed point function (DPF), with one-bit leakage, that is essentially as efficient as the state-of-the-art semi-honest protocol. Compared with previous work, our protocol takes about $50\times$ less communication for a domain with 2^{20} entries, and no longer requires actively secure generic 2PC.**
2) **We extend the dual-execution protocol to allow reactive computation, and then build a RAM-based 2PC protocol with active security on top of our new building blocks. The protocol follows the paradigm of Doerner and shelat (CCS 2017). We are able to prove that the protocol has end-to-end one-bit leakage.**
3) **Our implementation shows that our protocol is almost as efficient as the state-of-the-art semi-honest RAM-based 2PC protocol, and is at least two orders of magnitude faster than prior actively secure RAM-based 2PC without leakage, providing a realistic trade-off in practice.**

1. Introduction

Secure two-party computation (2PC) protocols [54] allow two parties each with a private input x, y respectively, to obtain $f(x, y)$ for some public function f but nothing else. There has been a huge amount of work to build efficient protocols and tools when f can be efficiently represented as a *circuit*; however, not all functions can be converted to a compact circuit since normal programs are in the *random-access machine (RAM) model*. To address this, secure 2PC in the RAM model [23] was proposed to support private accesses in 2PC protocols. It has found a lot of applications for building efficient and secure protocols for database queries [2], stable matching [15] and various graph algorithms [39].

The high-level approach of RAM-based 2PC is to combine oblivious RAMs (ORAMs) [22] and 2PC protocols. In more detail, one can use a 2PC protocol to emulate an ORAM client securely while having the parties act as the ORAM server(s): since the ORAM ensures that the server does not learn the private accesses, the parties cannot learn the accesses either. Although there has been huge progress in pushing the efficiency of RAM-based 2PC by means of optimized ORAM for secure computation [19], [23], [34], [50], [46], [56] and customized protocols leveraging the fact that there are two non-colluding ORAM servers with computational resources [24], [1], [16], [17], [40], [45], [28], all of them are in the semi-honest setting. The only exception is the work of Keller and Yanai [35] (dubbed KY18), where they proposed an optimized protocol based on the Circuit ORAM [46] and the SPDZ-BMR protocol [37]. When comparing its performance with state-of-the-art semi-honest protocols [16], we observe a huge gap of at least two orders of magnitude slowdowns, making it essentially infeasible to run any RAM-based 2PC applications in the malicious setting. When diving into the details, there are two main sources of slowdown.

1) **Actively secure circuit-based 2PC has a high overhead.** The generic approach of RAM-based 2PC can be done with malicious security by emulating the ORAM client in a reactive 2PC with malicious security. Indeed, this is the approach that KY18 took. However, due to the high depth of circuits needed to emulate ORAM circuits, a constant-round malicious 2PC is the only option. KY18 used the SPDZ-BMR protocol, which allows identification in the event of abort; this feature is crucial to enable their efficient representation of the server verifiable secret sharing, which can lead to two orders of magnitude improvements in memory usage. KY18 also posted an open problem on how to make it compatible with more efficient authenticated garbling [48] approach, which is still

*Corresponding authors

DOI 10.1109/SP54263.2024.00182

open to this date. Regardless, constant-round maliciously secure 2PC generally incurs a significant performance slowdown and this overhead will be amplified in a RAM protocol when emulating the ORAM algorithm in 2PC.

2) **Tricks in semi-honest protocols no longer work directly.** State-of-the-art RAM-based 2PC protocols use a crucial tool, namely distributed point function (DPF) [20], [8], which allows two parties with secret shares of α and β to homomorphically evaluate the point function $f_{(\alpha,\beta)}(x)$, that evaluates to β only when x equals α and 0 otherwise; recent DPFs [16], [27] let parties obtain secret shares of the output with communication sublinear to the number of evaluations. This implies an efficient protocol to read or write an array but not both at the same time. Doerner and shelat [16] first proposed a protocol, namely Floram, using DPF on top of the square-root ORAM, which was later improved in a sequence of works [28], [45]. However, bringing the same trick to malicious security is challenging: 1) it is not clear how to efficiently distribute DPF keys based on shares of α and β with malicious security; 2) it is unclear how to ensure the correctness of the local computation, an important feature of DPF-based ORAMs.

Contribution. In this paper, we design and implement a maliciously secure RAM-based 2PC protocol with high concrete performance. The protocol would leak one bit of information to the adversary but enjoys performance essentially the same as state-of-the-art semi-honest RAM 2PC protocols.

1) We design an efficient and maliciously secure protocol for distributed point functions (DPFs). Compared to previous malicious protocols, our protocol follows a different route in generating the DPF correlation and no longer needs generic malicious 2PC. As a result, our protocol improves the communication by a factor of $50\times$. What's more, the cost of this protocol is almost the same as the state-of-the-art semi-honest DPF protocols [27]. It also has huge applications beyond RAM-based 2PC, e.g., in malicious pseudorandom correlation generators.
2) We extend the normal dual-execution with one-bit leakage protocol [30] to support reactive 2PC. Then we incorporate both building blocks to build a malicious RAM-based 2PC based on the blueprint of Floram. Although DPF is invoked repeatedly, we show an optimization that allows end-to-end leakage to be a single bit by carefully controlling the abort event.
3) We implement all of the protocols and hook them with generic malicious 2PC for end-to-end applications. Our benchmark shows that the performance of our active-secure one-bit leakage protocol is almost as fast as semi-honest protocols in common network settings and is two orders of magnitude faster than prior full malicious RAM-based 2PC [35].

Paper organization. Section 2 provides an overview of our techniques and improvements. In Section 3, we introduce preliminaries. In Section 4, we provide details of our reactive 2PC protocol; in Section 5, we show our efficient DPF protocol details. We combine them together to build a RAM-based 2PC protocol in Section 6. Finally, in Section 7, we discuss the concrete performance of our protocols.

2. Technical Overview

2.1. Recap of Floram

First, we review the high-level ideas of Floram [16], one of the state-of-the-art semi-honest RAM-based 2PC protocols. The protocol has a read-only memory (ROM), a write-only memory (WOM), and a stash (S) supporting both read and write. Suppose that the initial values are in both the ROM and WOM; the protocol will ensure that 1) WOM always contains the most recent data (but we cannot read from it) and 2) ROM and S as a whole also contain the most recent data where the version in S takes priority. For a read operation, one just needs to query from the ROM structure and then linearly scan all elements in S; for a write operation, one first updates the WOM, and then appends this update to S. Both ROM and WOM can be efficiently built using DPFs. When S reaches σ elements, a refresh protocol will be executed that copies over the data in WOM to ROM and clears the stash S. Due to the advances in DPF, the communication cost of an operation on ROM and WOM is $O(\log N)$ for an array of size N; the stash is instead implemented using generic 2PC protocols. Thus the amortized communication cost is $O(\log N + \sigma + N/\sigma)$, which minimizes to $O(\sqrt{N})$.

In order to bring this idea to malicious security, we need to make all building blocks maliciously secure and allow them to be composed without causing inconsistency. Below, we discuss the details of each component.

2.2. Reactive 2PC with One-Bit Leakage

Next, we briefly discuss the intuition in our reactive 2PC protocol. Active 2PC with one-bit leakage was studied before [30], [41], but was only assumed as two parties evaluate a function for one shot. Their intuition is to run Yao's garbled circuit protocol twice with opposite directions along with malicious oblivious transfer and run a check protocol in the end to ensure the consistency of two executions. Either the output is correct, or the protocol will abort; thus the adversary can only learn one bit of information from the fact that the protocol aborts or proceeds. However, in our setting, two parties need to hold a "state" (e.g., stash) that is fed to a reactive 2PC and gets updated by the protocol.

To enable this upgrade, we hook the idea of dual execution with BDOZ authenticated shares [4]. Recall that to authenticate a secret sharing of a bit b as BDOZ share (namely $\langle\langle b \rangle\rangle$), party P_0 holds $(b_0, \mathsf{M}_0[b_0], \mathsf{K}_0[b_1])$ and P_1 holds $(b_1, \mathsf{M}_1[b_1], \mathsf{K}_1[b_0])$, such that $\mathsf{M}_0[b_0] = \mathsf{K}_1[b_0] \oplus b_0\Delta_1$ and $\mathsf{M}_1[b_1] = \mathsf{K}_0[b_1] \oplus b_1\Delta_0$ where Δ_0, Δ_1 are private MAC keys held by P_0 and P_1 respectively. When P_0 is the garbler, we let it produce a garbled circuit (GC) where the free-XOR delta is Δ_0. For an input bit b in BDOZ share, P_0 can define a zero garbled key as $L^0 = \mathsf{K}_0[b_1] \oplus b_0\Delta_0$ and P_1 defines

$L^* = \mathsf{M}_1[b_1]$, we can see that

$$L^* = \mathsf{M}_1[b_1] = \mathsf{K}_0[b_1] \oplus (b_0 \oplus b) \cdot \Delta_0 = L^0 \oplus b\Delta_0.$$

This means that L^* held by P_1 as an evaluator and L^0 held by P_0 as a garbler have a correct relationship needed for GC generation/evaluation. In summary, this is an approach where two parties can *locally* convert BDOZ shares to garbled labels compatible with dual execution. There is a similar process making dual-execution garbled labels back to BDOZ shares *locally*, although the shares may not be valid if one of the parties cheats during GC execution. To obtain the output with guaranteed correctness, two parties need first to check the validity of the authenticated share and only reveal it if it is valid. One bit of leakage is due to the validity check.

With this intuition, two parties store any state in BDOZ shares and convert them to garbled labels when they need to run 2PC, where the results can be converted back to BDOZ shares. The overhead compared to semi-honest Yao's protocol is exactly twice, but it can be parallelized easily.

2.3. Efficient DPF with Malicious Security

Prior protocol. Recall that in a DPF protocol, two parties have secret sharing of $\alpha \in [n]$ and $\beta \in \mathbb{F}$ and should get secret sharings of a size-n vector $\boldsymbol{x} \in \mathbb{F}^n$, which is all zeros except that $x^{(\alpha)} = \beta$. To make the DPF protocol maliciously secure, there are two important tasks: 1) use authenticated sharing for the input and output of the DPF protocol, and 2) prevent the parties from cheating during the execution of the protocol. Ensuring DPF to output authenticated sharing can be done via appending β with $\beta \cdot \Delta$, where Δ is the secret shared MAC key; this works as long as the DPF scheme allows any ring element as β. However, ensuring input authentication, consistency, and protocol security is much more complicated, as the state-of-the-art DPF protocol involves $\log N$ rounds and extensive local computation. The only maliciously secure protocol was proposed by Boyle et al. [7]. Their protocol works by first generating additive shares of vector in the form of $([0], \ldots, [0], [r], [0], \ldots, [0])$, where the share of a random value r is in the α-th location, following the classical semi-honest DPF protocol but replace all joint computation using a generic malicious 2PC. Then, two parties further expand a level to obtain shares of $2n$ elements: $([0], \ldots, [0], [L], [R], [0], \ldots, [0])$, where L, R are random values and $[L]$ is the 2α-th element. Two parties then again use generic malicious 2PC to compute authenticated shares of L^{-1} and R^{-1} while only revealing R^{-1}. Next, two parties pick a public random value χ and compute two linear combinations on their secret sharings which will end up being $X_L = \chi^\alpha \cdot L$ and $X_R = \chi^\alpha \cdot R$. Finally, they can check whether $[X_L] \cdot [L^{-1}] = [X_R] \cdot R^{-1}$ in malicious 2PC.

Their analysis shows that this protocol unfortunately leaks one bit of information about α to the adversary. In terms of the cost, this protocol requires heavy use of generic malicious 2PC and, in particular, needs to compute three field multiplications in 2PC, which is very expensive. For example, MASCOT [32] requires about 33,000 bytes of communication to compute one such multiplication even without counting the cost of underlying oblivious transfer, while the rest part of this protocol only needs $2(\log N + 1)\kappa$ bits of communication. This means that the cost of this field multiplication is going to be the main bottleneck of the whole DPF protocol for any reasonable size of N. Another potential issue is that this malicious DPF protocol requires β to be a field element (so that inverse exists), and thus it is not immediately clear how to efficiently support output authentication, where β has two field elements.

Our protocol. The prior protocol is costly and also heavily relies on malicious generic 2PC, making it complicated to implement. In this work, we propose a completely different way to generate DPF with malicious security without using generic malicious 2PC or field multiplication, while still maintaining the same level of security. As a result, the protocol is much easier to implement and is almost as efficient as state-of-the-art semi-honest DPF protocols.

Different from the prior work that first generates the whole vector of shares and then checks the relationship in a modular way, our protocol maintains the invariance that after each level of expansion, two parties hold authenticated sharing of partial prefix expansion. To be more specific, we assume that the two parties start with a SPDZ authenticated share of 1, namely $[\![1]\!]$. A SPDZ authentication is similar to BDOZ but instead parties hold secret shares of the value b and its MAC $b \cdot \Delta$ (along with secret shares of the MAC key), i.e., (b_0, M_0) and (b_1, M_1) such that $M_1 \oplus M_0 = (b_0 \oplus b_1) \cdot (\Delta_0 \oplus \Delta_1)$. See Section 3.3 for complete details. Two parties use one level of expansion to either get $([\![0]\!], [\![1]\!])$ or $([\![1]\!], [\![0]\!])$, depending on the most significant bit of α. This process can be iteratively executed to obtain $([\![0]\!], \ldots, [\![0]\!], [\![1]\!], [\![0]\!], \ldots, [\![0]\!])$, where $[\![1]\!]$ is at the α-th location. Finally, a correction word is used to correct $[\![1]\!]$ to $[\![\beta]\!]$ while maintaining $[\![0]\!]$ unchanged.

Given this high-level approach, the key is to expand one level of the tree. Our high-level idea follows a semi-honest optimization of DPF, namely Half-Tree [27]. Suppose two parties hold (x_0, X_0) and (x_1, X_1) respectively as their SPDZ share of 1 at the root such that $X_0 \oplus X_1 = \Delta_0 \oplus \Delta_1$ To obtain $([\![a]\!], [\![a \oplus 1]\!])$ for some private $a \in \{0, 1\}$, with a hash function $\mathcal{H}$ the correction word CW would be

$$\mathsf{CW} := \mathcal{H}(x_0 \| X_0) \oplus \mathcal{H}(x_1 \| X_1) \oplus (a \oplus 1) \cdot (\Delta_0 \oplus \Delta_1).$$

Each party can locally expand the left-child node as $(l_b, L_b) := \mathcal{H}(x_b \| X_b) \oplus x_b \cdot \mathsf{CW}$ and the right-child node as $(r_b, R_b) := \mathcal{H}(x_b \| X_b) \oplus X_b \oplus x_b \cdot \mathsf{CW}$. In addition, computing CW boils down to compute $a \cdot \Delta$ efficiently; when a is authenticated, their shares can be used to reconstruct shares of $a \cdot \Delta$ locally; thus computing shares of CW can all be done via local computation. Our crucial observation is that, the adversary can only cheat by corrupting CW with an additive value. However, if such corruption happens, the only type of change is to make the authenticated shares on the next level ((l_b, L_b), (r_b, R_b) in the above example) invalid, which can be easily discovered by an almost-free MAC check protocol. Unfortunately, the adversary can still learn one-bit information since its cheat could lead to an abort event or not, depending on the bit a. However, it is sufficient in our application and many other applications in pseudorandom correlation generators. We refer to Section 5 for more details.

2.4. Putting Everything Together

Given the above two important building blocks already optimized with high efficiency, we can now build an efficient RAM-based 2PC protocol with active security. We follow the blueprint of Floram [16] and use authenticated shares, either in BDOZ or SPDZ, to connect various building blocks. Here the main challenge is to avoid secure computation of pseudorandom functions (PRFs) during refresh protocols, which would be prohibitive. It is clear that for WOM, two parties would store the authenticated shares, but the design of ROM is more complicated (as we elaborate below). Our final solution in the end only requires 2 PRF computations in 2PC for each operation.

Write-only memory. Suppose elements stored in the RAM model are represented as an array $\boldsymbol{D}$ with totally N elements. For WOM, two parties need to hold authenticated shares of $D^{(i)}$. To update the α-th value to D^*, two parties first read from ROM to obtain $[\![D^{(\alpha)}]\!]$ and then use DPF to obtain an authenticated vector of field elements $(\ldots, [\![0]\!], [\![D^{(\alpha)} \oplus D^*]\!], [\![0]\!], \ldots)$, where the non-zero element is at location α, and then locally XOR each element in the list to the authenticated shares of $D^{(0)}, \ldots, D^{(N-1)}$ corresponding. Although this version requires two separate DPFs, one can apply the optimization in Floram to reduce it to call the DPF protocol only once. We provide full details in Section 6.

Read-only memory: First attempt. Two parties hold authenticated sharing of two PRF keys $[\![k_0]\!]$ and $[\![k_1]\!]$. For ROM, we can think of a scheme where the i-th data block $D^{(i)}$ is encrypted as $E^{(i)} = \mathrm{PRF}(k_0, i) \oplus \mathrm{PRF}(k_1, i) \oplus D^{(i)}$ and is public to both parties. For a read operation at α, two parties would use the above malicious DPF protocol to obtain a unit vector $(\ldots, [\![0]\!], [\![1]\!], [\![0]\!], \ldots)$, where $[\![1]\!]$ is specified by α. Then two parties can compute $[\![E^{(\alpha)}]\!]$ by computing the inner product between the vector $\boldsymbol{E}$ and the authenticated unit vector, and then use 2PC to decrypt it to obtain the authenticated share $[\![D^{(\alpha)}]\!]$. So far, everything works great, but the challenge appears when connecting WOM to ROM via a refresh procedure. Essentially, the problem setup is that two parties have $[\![k_0]\!], [\![k_1]\!]$ and $[\![D^{(i)}]\!]$; we need a protocol so that they obtain $E^{(i)} = \mathrm{PRF}(k_0, i) \oplus \mathrm{PRF}(k_1, i) \oplus D^{(i)}$. To defend against a malicious adversary, the values held by the honest party should be correct even if the adversary cheats in some way. One way to ensure this property is to mask all PRFs in a 2PC protocol, but this would require $2N$ PRF computation in 2PC. This computation would blow up the cost since it can only cover about $\sqrt{N}$ writes efficiently, leading to perform PRF computation $O(\sqrt{N})$ times in 2PC per access. Alternatively, two parties can compute PRF locally, supply them to 2PC to compute the masking step, and then reveal the result; however, this approach would allow parties to change the value as the adversary can claim any value as their PRF evaluation. In summary, it is not clear how to ensure consistency between WOM and ROM.

Read-only memory: Our approach. Our alternative method is to put the value and its SPDZ MAC together into the ROM. Since data are doubly encrypted by both parties, no information can be revealed. Furthermore, the additional MAC allows us to ensure consistency. In more detail, we now have $E^{(i)} = \mathrm{PRF}(k_0, i) \oplus \mathrm{PRF}(k_1, i) \oplus (D^{(i)}, D^{(i)} \cdot \Delta)$, where $\Delta = \Delta_0 \oplus \Delta_1$ is the SPDZ MAC key and the output length of PRF is sufficient for two elements. To read the α-th element from the array, two parties first compute $[\mathrm{PRF}(k_0, \alpha) \oplus \mathrm{PRF}(k_1, \alpha)]$ in 2PC, and use malicious DPF to obtain XOR-secret sharing of a unit vector $[\boldsymbol{u}] = (\ldots, [0], [1], [0], \ldots)$, where the non-zero value is at index α. Two parties locally compute $\left(\bigoplus_i [u^{(i)}] \cdot E^{(i)}\right) \oplus [\mathrm{PRF}(k_0, \alpha) \oplus \mathrm{PRF}(k_1, \alpha)] = [(D^{(\alpha)}, D^{(\alpha)} \cdot \Delta)]$, which is essentially the SPDZ authenticated sharing $[\![D^{(\alpha)}]\!]$. Here we no longer need MACs on the output of DPF as long as DPF is maliciously secure: if any party cheats in any way, SPDZ shares as the output will be invalid independent of the underlying data.

Back to refresh procedure: now two parties have sharings $[\![k_0]\!], [\![k_1]\!]$, $[\![D^{(i)}]\!]$ and need to obtain $E^{(i)} = \mathrm{PRF}(k_0, i) \oplus \mathrm{PRF}(k_1, i) \oplus (D^{(i)}, D^{(i)} \cdot \Delta)$. Two parties can treat the SPDZ sharing $[\![D^{(i)}]\!]$ as additive sharing $[(D^{(i)}, D^{(i)} \cdot \Delta)]$. Since P_0 can compute $\mathrm{PRF}(k_0, i)$ while P_1 can compute $\mathrm{PRF}(k_1, i)$, they effectively have additive shares $[\mathrm{PRF}(k_0, i) \oplus \mathrm{PRF}(k_1, i) \oplus (D^{(i)}, D^{(i)} \cdot \Delta)]$. To reveal the underlying value, we can just allow them to exchange the shares. Since the public values themselves will eventually be used as authenticated values, any change of values will cause abort.

Bounding the leakage. With the above changes, the protocol is essentially as cheap as its semi-honest counterpart. However, a naive argument would lead to an amount of leakage linear to the number of RAM access operations, since every operation requires outputting some value, where checks are needed, leaving an opportunity to leak a bit. To reduce the amount of leakage, we batch all checks since they all verify consistency between values and their MACs, and defer these checks right before revealing the designated output (i.e., $f(x, y)$ where f is the function in the RAM model to be evaluated). For any intermediate values, we will open them without a check. This will not leak any information because all opened intermediate values are masked by authenticated shares of random values as how we design the protocol. This way, all intermediate values can be simulated while the only abort end is in the end.

3. Preliminaries

3.1. Notation

We use κ to denote the computational security parameter. We denote by $\log(\cdot)$ the logarithm in base 2. We write $x \leftarrow S$ to denote sampling x uniformly at random from a set S. We define $[a, b) := \{a, \ldots, b-1\}$ and $[a, b] := \{a, \ldots, b\}$. For an n-bit integer x, we denote

Functionality $\mathcal{F}_{\mathsf{aBit}}$

Initialize: This command is called only once. Upon receiving $(\mathsf{init}, \Delta_0)$ from P_0 and $(\mathsf{init}, \Delta_1)$ from P_1, where $\Delta_0, \Delta_1 \in \mathbb{F}_{2^\kappa}$, abort if $\mathsf{lsb}(\Delta_0 \oplus \Delta_1) \neq 1$; otherwise store (Δ_0, Δ_1).

Authenticate bits: This command can be called multiple times. For $b \in \{0,1\}$, upon receiving $(\mathsf{auth}, b, \boldsymbol{x}, \ell)$ from P_b and (auth, b, ℓ) from P_{1-b}, where $\boldsymbol{x} \in \mathbb{F}_2^\ell$, do the following:

- Sample $\mathsf{K}_{1-b}[\boldsymbol{x}] \leftarrow \mathbb{F}_{2^\kappa}^\ell$. If P_{1-b} is corrupted, instead receive $\mathsf{K}_{1-b}[\boldsymbol{x}] \in \mathbb{F}_{2^\kappa}^\ell$ from the adversary.
- Compute $\mathsf{M}_b[\boldsymbol{x}] := \mathsf{K}_{1-b}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta_{1-b} \in \mathbb{F}_{2^\kappa}^\ell$. If P_b is corrupted, receive $\mathsf{M}_b[\boldsymbol{x}] \in \mathbb{F}_{2^\kappa}^\ell$ from the adversary, and then recompute $\mathsf{K}_{1-b}[\boldsymbol{x}] := \mathsf{M}_b[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta_{1-b} \in \mathbb{F}_{2^\kappa}^\ell$.
- Output $\mathsf{M}_b[\boldsymbol{x}]$ to P_b and $\mathsf{K}_{1-b}[\boldsymbol{x}]$ to P_{1-b}.

Figure 1: Functionality for authenticated bits.

by $(x^{(0)}, \ldots, x^{(n-1)})$ its bit decomposition, that is, $x^{(i)} \in \{0,1\}$ for $i \in [0,n)$ and $x = \sum_{i \in [0,n)} x^{(i)} \cdot 2^i$. We use bold lower-case letters like $\boldsymbol{x}$ to denote a vector and $x^{(i)}$ to denote the i-th component of $\boldsymbol{x}$ with $x^{(0)}$ the first component. We use $\mathsf{lsb}(x)$ to denote the least significant bit (LSB) of a string x (i.e., $x^{(0)}$). We write $\mathbb{F}_{2^\kappa} \cong \mathbb{F}_2[X]/f(X)$ for a monic irreducible polynomial $f(X)$ of degree κ. We use $\mathsf{X} \in \mathbb{F}_{2^\kappa}$ to denote the element corresponding to $X \in \mathbb{F}_2[X]/f(X)$. Depending on the context, we use $\{0,1\}^\kappa$, $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$ interchangeably, and thus addition in $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$ corresponds to XOR in $\{0,1\}^\kappa$. We use $\mathbf{unit}(N, \alpha) \in \mathbb{F}_{2^\kappa}^N$ for a vector with exact one non-zero entry 1 at position $\alpha \in [0,N)$.

3.2. Security Model and Ideal Functionalities

We use the standard ideal/real paradigm [10], [21] to prove security of our two-party protocols in the presence of a *malicious, static* adversary. In the *ideal-world* execution, two parties P_0 and P_1 interact with an ideal functionality $\mathcal{F}$, and one of them may be corrupted by an *ideal-world adversary* (a.k.a., *simulator*) $\mathcal{S}$. In the *real-world* execution, P_0 interacts with P_1 via executing a protocol Π, and one of them may be corrupted by a *real-world adversary* $\mathcal{A}$. We say that a protocol Π securely realizes an ideal functionality $\mathcal{F}$, if the real-world execution is computationally indistinguishable from the ideal-world execution.

Our protocols call the standard two-party functionalities: the coin-tossing functionality $\mathcal{F}_{\mathsf{coin}}$ and the commitment functionality $\mathcal{F}_{\mathsf{com}}$, which can be securely realized using a random oracle [12].

3.3. Authenticated Secret Sharings

We consider two kinds of authenticated secret sharings in the two-party setting, i.e., SPDZ style [13], [12] and BDOZ style [4]. Suppose that P_0 (resp., P_1) holds a global key $\Delta_0 \in \mathbb{F}_{2^\kappa}$ (resp., $\Delta_1 \in \mathbb{F}_{2^\kappa}$).

We use $[\![x]\!]$ to denote a SPDZ-style authenticated secret sharing on $x \in \mathbb{F}_{2^\kappa}$. In particular, we have $[\![x]\!] = ([\![x]\!]_0, [\![x]\!]_1)$ and, for each $b \in \{0,1\}$, P_b holds

$$[\![x]\!]_b := (x_b, \mathsf{M}_b[x]) \in \mathbb{F}_{2^\kappa}^2$$

such that $x = x_0 + x_1$ and $\mathsf{M}_0[x] + \mathsf{M}_1[x] = x \cdot (\Delta_0 + \Delta_1) \in \mathbb{F}_{2^\kappa}$. We use $[x] = ([x]_0, [x]_1)$ to denote an unauthenticated additive sharing, i.e., $[x]_0 + [x]_1 = x$. So, we have $[\![x]\!] = ([x], [x \cdot \Delta])$ with $\Delta = \Delta_0 + \Delta_1$. Note that SPDZ-style authenticated sharings are additively homomorphic, i.e., two parties can locally compute $[\![a \cdot x + b \cdot y]\!] = a \cdot [\![x]\!] + b \cdot [\![y]\!]$ for any public constants $a, b \in \mathbb{F}_{2^\kappa}$. Besides, for any public constant c, both parties can locally compute $[\![c]\!]$ by setting $x_0 := c$, $x_1 := 0$, $\mathsf{M}_0[x] := c \cdot \Delta_0$ and $\mathsf{M}_1[x] := c \cdot \Delta_1$. For a vector $\boldsymbol{x} \in \mathbb{F}_{2^\kappa}^\ell$, we write $[\![\boldsymbol{x}]\!] = ([\![x^{(0)}]\!], \ldots, [\![x^{(\ell-1)}]\!])$. In Figure 2, we describe the batch-check protocol with essentially no communication, which can verify the correctness of multiple values opened in a batch.

For a bit $x \in \mathbb{F}_2$, we write $\langle\!\langle x \rangle\!\rangle$ to denote a BDOZ-style authenticated secret sharing. In particular, we have $\langle\!\langle x \rangle\!\rangle := (\langle\!\langle x \rangle\!\rangle_0, \langle\!\langle x \rangle\!\rangle_1)$ and, for each $b \in \{0,1\}$, P_b holds

$$\langle\!\langle x \rangle\!\rangle_b = (x_b, \mathsf{K}_b[x_{1-b}], \mathsf{M}_b[x_b]) \in \mathbb{F}_2 \times \mathbb{F}_{2^\kappa}^2$$

such that secret bit $x = x_0 \oplus x_1$ and MAC tag $\mathsf{M}_b[x_b] = \mathsf{K}_{1-b}[x_b] + x_b \cdot \Delta_{1-b} \in \mathbb{F}_{2^\kappa}$. The BDOZ-style authenticated sharings can be generated by calling the functionality $\mathcal{F}_{\mathsf{aBit}}$ (shown in Figure 1). In this figure, for the sake of simplicity, we write $\boldsymbol{x} = (x^{(0)}, \ldots, x^{(\ell-1)})$, $\mathsf{K}_{1-b}[\boldsymbol{x}] = (\mathsf{K}_{1-b}[x^{(0)}], \ldots, \mathsf{K}_{1-b}[x^{(\ell-1)}])$ and $\mathsf{M}_b[\boldsymbol{x}] = (\mathsf{M}_b[x^{(0)}], \ldots, \mathsf{M}_b[x^{(\ell-1)}])$. This functionality has been used in previous works [48], [49], [29], [52]. Functionality $\mathcal{F}_{\mathsf{aBit}}$ can be securely realized against malicious adversaries by executing a correlated oblivious transfer (COT) protocol [31], [6], [53], [51], [44], [5], [27]. To guarantee $\mathsf{lsb}(\Delta_0 \oplus \Delta_1) = 1$, the consistency check in [11] can be adopted (particularly, κ random authenticated sharings need to be sacrificed). It is clear that BDOZ-style authenticated sharings are also additively homomorphic. For a bit vector $\boldsymbol{x} \in \mathbb{F}_2^\ell$, we write $\langle\!\langle \boldsymbol{x} \rangle\!\rangle = (\langle\!\langle x^{(0)} \rangle\!\rangle, \langle\!\langle x^{(1)} \rangle\!\rangle, \ldots, \langle\!\langle x^{(\ell-1)} \rangle\!\rangle)$.

Both parties can locally compute an authenticated sharing on a field element $x \in \mathbb{F}_{2^\kappa}$ from κ authenticated sharings $\langle\!\langle x^{(0)} \rangle\!\rangle, \ldots, \langle\!\langle x^{(\kappa-1)} \rangle\!\rangle$ where $x^{(i)} \in \{0,1\}$ for each $i \in [0,\kappa)$. In particular, both parties are able to locally compute $\langle\!\langle x \rangle\!\rangle := \sum_{i \in [0,\kappa)} \langle\!\langle x^{(i)} \rangle\!\rangle \cdot \mathsf{X}^i$. We denote by $\langle\!\langle x \rangle\!\rangle := \mathsf{B2F}(\langle\!\langle x^{(0)} \rangle\!\rangle, \ldots, \langle\!\langle x^{(\kappa-1)} \rangle\!\rangle)$ this local computation. Besides, we can transform a BDOZ-style authenticated sharing to a SPDZ-style authenticated sharing without any interaction [9]. Specifically, given $\langle\!\langle x \rangle\!\rangle = (\langle\!\langle x \rangle\!\rangle_0, \langle\!\langle x \rangle\!\rangle_1)$, both parties locally compute $[\![x]\!]$ by setting $[\![x]\!]_b := (x_b, \mathsf{K}_b[x_{1-b}] \oplus \mathsf{M}_b[x_b] \oplus x_b \Delta_b)$ for each $b \in \{0,1\}$. We write $[\![x]\!] := \mathsf{Convert}(\langle\!\langle x \rangle\!\rangle)$ for this computation.

3.4. Garbling Scheme

Following the previous work [3], we give the definition of garbling schemes, which is specified for our usage. For a bit $x \in \{0,1\}$, we use $\mathsf{K}[x] \in \{0,1\}^\kappa$ to denote the 0-label and $\mathsf{M}[x] \in \{0,1\}^\kappa$ to denote the garbled label on bit x. We always consider that the free-XOR technique [36] is adopted, which is the case for the state-of-the-art garbling schemes [55], [43]. In this case, a random global key $\Delta \in \{0,1\}^\kappa$ is sampled, and $\mathsf{M}[x] = \mathsf{K}[x] \oplus x\Delta$ for any bit $x \in \{0,1\}$. We observe that garbled labels have the same form of BDOZ-style authenticated bits (modeled in functionality $\mathcal{F}_{\mathsf{aBit}}$). In our 2PC protocol shown in Section 4, we will

Protocol $\Pi_{\mathsf{BatchCheck}}$

Input: Two parties P_0 and P_1 hold ℓ SPDZ-style authenticated sharings $[\![y^{(0)}]\!], \dots, [\![y^{(\ell-1)}]\!]$ along with their opened values $y^{(i)} \in \mathbb{F}_{2^\kappa}$ for each $i \in [0, \ell)$.

Batch check: Two parties do the following.

1) Two parties call $\mathcal{F}_{\mathsf{coin}}$ to sample a random $\chi \in \mathbb{F}_{2^\kappa}$.
2) Two parties locally compute $[\![z]\!] := \sum_{i \in [0,\ell)} \chi^i \cdot [\![y^{(i)}]\!]$ and $z := \sum_{i\in[0,\ell)} \chi^i \cdot y^{(i)} \in \mathbb{F}_{2^\kappa}$.
3) For each $b \in \mathbb{F}_2$, P_b computes $V_b := \mathsf{M}_b[z] + z \cdot \Delta_b$ and calls $\mathcal{F}_{\mathsf{com}}$ to commit to V_b.
4) For each $b \in \mathbb{F}_2$, P_b calls $\mathcal{F}_{\mathsf{com}}$ to open V_b. Then, two parties check $V_0 = V_1$ and abort if the check fails.

Figure 2: Protocol for batch-checking the values authenticated by SPDZ-style MACs in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.

call functionality $\mathcal{F}_{\mathsf{aBit}}$ to generate garbled labels on input wires. Thus, Δ and 0-labels corresponding to input bits have been defined by the BDOZ-style authenticated bits, and are able to be used as the input of garbling algorithm Garble. Similarly, the garbled labels on input bits are defined by the MAC tags in the authenticated bits, and can be used as the input of evaluation algorithm Eval. We will transform the garbled labels on output bits into authenticated bits, instead of decoding them to obtain the output bits. Overall, our 2PC protocol only needs two algorithms Garble and Eval, where the encoding and decoding algorithms are not required.

Definition 1. *A garbling scheme $\mathcal{GS} = (\mathsf{Garble}, \mathsf{Eval})$, which is specific to our application, consists of the following two algorithms.*

- $(GC, \mathsf{K}[\boldsymbol{y}]) \leftarrow \mathsf{Garble}(\mathsf{K}[\boldsymbol{x}], \Delta, \mathcal{C})$*: Given a vector of* 0*-labels* $\mathsf{K}[\boldsymbol{x}]$ *on input wires, a global key* Δ *and a Boolean circuit* $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$*, this algorithm outputs a garbled circuit* GC *along with a vector of* 0*-labels* $\mathsf{K}[\boldsymbol{y}]$ *on output wires.*
- $\mathsf{M}[\boldsymbol{y}] \leftarrow \mathsf{Eval}(GC, \mathsf{M}[\boldsymbol{x}])$*: Given a garbled circuit* GC *and a vector of garbled labels* $\mathsf{M}[\boldsymbol{x}]$ *on input vector* $\boldsymbol{x}$*, this algorithm outputs a vector of garbled labels* $\mathsf{M}[\boldsymbol{y}]$ *on output vector* $\boldsymbol{y}$*.*

For security, we assume that the garbling scheme satisfies obliviousness [3]. That is, there exists a simulator $\mathcal{S}$, given a circuit $\mathcal{C}$, that can simulate a garbled circuit GC and a vector of garbled labels $\mathsf{M}[\boldsymbol{x}]$, which are computationally indistinguishable from the real values.

4. Constant-Round 2PC with Active Security

In Figure 3, we give a 2PC functionality $\mathcal{F}_{\mathsf{2PC}}$ in the active setting. This functionality allows two parties to input bits via the (input) command and generate random elements in $\mathbb{F}_{2^\kappa}$ via the (rand). By calling the (eval), two parties can compute any Boolean circuit. Two parties are able to call the (open) command to open some elements in $\mathbb{F}_{2^\kappa}$ to both of them. We do not consider the (output) command to output values to only one party, as it is not required for our RAM-based 2PC protocol (shown in Section 6). In addition, we define the (pack) and (unpack) commands to realize the conversion between κ bits and one element in $\mathbb{F}_{2^\kappa}$. Finally, a

Functionality $\mathcal{F}_{\mathsf{2PC}}$

This functionality initializes two identifier-value lists Bit and Val, where each value in Bit (resp., Val) is an element in $\mathbb{F}_2$ (resp., $\mathbb{F}_{2^\kappa}$). It interacts with two parties P_0 and P_1.

Input: Upon receiving $(\mathsf{input}, \mathsf{id}, b, x)$ from P_b and $(\mathsf{input}, \mathsf{id}, b)$ from P_{1-b}, where $b, x \in \mathbb{F}_2$, set $\mathsf{Bit}[\mathsf{id}] := x$.

Eval: Upon receiving $(\mathsf{eval}, \{\mathsf{id}^{(x_i)}\}_{i\in[0,n)}, \{\mathsf{id}^{(y_i)}\}_{i\in[0,m)}, \mathcal{C})$ from both parties, where $\mathcal{C} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is a boolean circuit, compute $(\mathsf{Bit}[\mathsf{id}^{(y_0)}], \mathsf{Bit}[\mathsf{id}^{(y_1)}], \dots, \mathsf{Bit}[\mathsf{id}^{(y_{m-1})}]) := \mathcal{C}(\mathsf{Bit}[\mathsf{id}^{(x_0)}], \mathsf{Bit}[\mathsf{id}^{(x_1)}], \dots, \mathsf{Bit}[\mathsf{id}^{(x_{n-1})}])$.

Rand: Upon receiving $(\mathsf{rand}, \mathsf{id})$ from both parties, sample $\mathsf{Val}[\mathsf{id}] \leftarrow \mathbb{F}_{2^\kappa}$.

Pack: Upon receiving $(\mathsf{pack}, \{\mathsf{id}^{(i)}\}_{i\in[0,\kappa)}, \mathsf{id})$ from both parties, compute $\mathsf{Val}[\mathsf{id}] := \sum_{i\in[0,\kappa)} \mathsf{Bit}[\mathsf{id}^{(i)}] \cdot X^i \in \mathbb{F}_{2^\kappa}$.

Unpack: Upon receiving $(\mathsf{unpack}, \mathsf{id}, \{\mathsf{id}^{(i)}\}_{i\in[0,\kappa)})$ from both parties, decompose $\mathsf{Val}[\mathsf{id}] := \sum_{i\in[0,\kappa)} x^{(i)} \cdot X^i \in \mathbb{F}_{2^\kappa}$ and define $\mathsf{Bit}[\mathsf{id}^{(i)}] := x^{(i)} \in \mathbb{F}_2$ for each $i \in [0, \kappa)$.

Open: Upon receiving $(\mathsf{open}, \mathsf{id})$ from both parties, send $\mathsf{Val}[\mathsf{id}] \in \mathbb{F}_{2^\kappa}$ to the adversary, wait for $x \in \mathbb{F}_{2^\kappa}$ from the adversary, and send x to both parties. If $x \neq \mathsf{Val}[\mathsf{id}]$, set a cheat flag.

Check: This command is allowed to be called only once. Upon receiving (check) from both parties, do the following:

1) Wait for a predicate $P : \mathbb{F}_2^{|\mathcal{I}|} \times \mathbb{F}_{2^\kappa}^{|\mathcal{J}|} \to \mathbb{F}_2$ from the adversary, where $\mathcal{I}$ (resp., $\mathcal{J}$) is the set of all available identifiers in list Bit (resp., Val).
2) If $P(\{\mathsf{Bit}[\mathsf{id}]\}_{\mathsf{id}\in\mathcal{I}}, \{\mathsf{Val}[\mathsf{id}]\}_{\mathsf{id}\in\mathcal{J}}) = 0$ or a cheat flag is set, abort.

Figure 3: Functionality for secure two-party computation with one-bit leakage.

malicious adversary, who corrupts either P_0 or P_1, can leak at most one-bit information on secret elements by inputting a predicate P only once.

Based on a garbling scheme and functionality $\mathcal{F}_{\mathsf{aBit}}$, we present an efficient 2PC protocol Π_{2PC} with active security in Figure 4. This protocol adopts the dual-execution framework [41], and securely realizes functionality $\mathcal{F}_{\mathsf{2PC}}$ (Figure 3). Note that the check procedure works as the batch check of SPDZ-style authenticated sharings, where BDOZ-style authenticated sharings are converted into SPDZ-style ones. The checking result allows a malicious adversary to make a selective-failure attack, i.e., an incorrect guess on the secret values will lead to the protocol aborts, and a correct guess will make the honest party accept. All the checks are done at the end of protocol execution, and thus the adversary can reveal at most one-bit information.

We use the Yao's 2PC protocol [54] based on garbling schemes to securely compute any Boolean circuit, and adopt dual execution to achieve active security with one-bit leakage. In the original dual execution [41], [30], each of two parties first acts as a garbler and then acts as an evaluator, and then both parties execute an equality check immediately after the circuit was computed. Different from the original dual execution, we defer the check to the open phase, and

Protocol Π_{2PC}

This protocol invokes $\Pi_{\mathsf{BatchCheck}}$ (Figure 2) as a sub-protocol, and adopts a garbling scheme $\mathcal{GS} = (\mathsf{Garble}, \mathsf{Eval})$.

Initialize: For each $b \in \mathbb{F}_2$, P_b samples $\Delta_b \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{lsb}(\Delta_b) = b$, and sends $(\mathsf{init}, b, \Delta_b)$ to $\mathcal{F}_{\mathsf{aBit}}$.

Input: For each $b \in \mathbb{F}_2$, for each input bit $x \in \mathbb{F}_2$ held by P_b, two parties P_0 and P_1 do the following:

1) P_b and P_{1-b} call $\mathcal{F}_{\mathsf{aBit}}$ on respective inputs $(\mathsf{auth}, b, x, 1)$ and $(\mathsf{auth}, b, 1)$ to obtain respective outputs $\mathsf{M}_b[x]$ and $\mathsf{K}_{1-b}[x]$. Then, P_{1-b} samples $r \leftarrow \mathbb{F}_{2^\kappa}$ and send r to P_b. Next, P_{1-b} updates $\mathsf{K}_{1-b}[x] := \mathsf{K}_{1-b}[x] \oplus r$, and P_b updates $\mathsf{M}_b[x] := \mathsf{M}_b[x] \oplus r$.
2) P_b defines $\mathsf{K}_b[0] = s$ and P_{1-b} sets $\mathsf{M}_{1-b}[0] = s$ by letting P_b sample $s \leftarrow \mathbb{F}_{2^\kappa}$ and send s to P_{1-b}.
3) Both parties define $\langle\!\langle x \rangle\!\rangle = (\langle\!\langle x \rangle\!\rangle_b, \langle\!\langle x \rangle\!\rangle_{1-b})$, where $\langle\!\langle x \rangle\!\rangle_b := (x, \mathsf{K}_b[0], \mathsf{M}_b[x])$ and $\langle\!\langle x \rangle\!\rangle_{1-b} := (0, \mathsf{K}_{1-b}[x], \mathsf{M}_{1-b}[0])$.

Eval: To compute $(y^{(0)}, \ldots, y^{(m-1)}) \leftarrow \mathcal{C}(x^{(0)}, \ldots, x^{(n-1)})$, two parties P_0 and P_1 use BDOZ-style authenticated sharings $\{\langle\!\langle x^{(i)} \rangle\!\rangle\}_{i \in [0,n)}$ to compute $\{\langle\!\langle y^{(i)} \rangle\!\rangle\}_{i \in [0,m)}$ as follows, where $\langle\!\langle x^{(i)} \rangle\!\rangle_b = (x_b^{(i)}, \mathsf{K}_b[x_{1-b}^{(i)}], \mathsf{M}_b[x_b^{(i)}])$ for each $b \in \mathbb{F}_2$ and $i \in [0, n)$.

1) For each $b \in \mathbb{F}_2$, P_b computes $\mathsf{K}_b[x^{(i)}] := \mathsf{K}_b[x_{1-b}^{(i)}] \oplus x_b^{(i)} \cdot \Delta_b$ and P_{1-b} computes $\mathsf{M}_{1-b}[x^{(i)}] := \mathsf{M}_{1-b}[x_{1-b}^{(i)}]$ such that $\mathsf{M}_{1-b}[x^{(i)}] = \mathsf{K}_b[x^{(i)}] \oplus x^{(i)} \cdot \Delta_b$ for each $i \in [0, n)$.
2) As a garbler, for each $b \in \mathbb{F}_2$, P_b runs $(GC_b, \{\mathsf{K}_b[y^{(i)}]\}_{i \in [0,m)}) \leftarrow \mathsf{Garble}(\{\mathsf{K}_b[x^{(i)}]\}_{i \in [0,n)}, \Delta_b, \mathcal{C})$, and sends GC_b to P_{1-b}.
3) As an evaluator, for each $b \in \mathbb{F}_2$, P_{1-b} runs $\{\mathsf{M}_{1-b}[y^{(i)}]\}_{i \in [0,m)} \leftarrow \mathsf{Eval}(\{\mathsf{M}_{1-b}[x^{(i)}]\}_{i \in [0,n)}, GC_b)$.
4) For each $b \in \mathbb{F}_2$, P_b computes $y_b^{(i)} := \mathsf{lsb}(\mathsf{K}_b[y^{(i)}] \oplus \mathsf{M}_b[y^{(i)}]) \in \mathbb{F}_2$ and $\langle\!\langle y^{(i)} \rangle\!\rangle_b := (y_b^{(i)}, \mathsf{K}_b[y^{(i)}] \oplus y_b^{(i)} \cdot \Delta_b, \mathsf{M}_b[y^{(i)}])$ for each $i \in [0, m)$. As a result, both parties hold BDOZ-style authenticated sharing $\langle\!\langle y^{(i)} \rangle\!\rangle$ for each $i \in [0, m)$.

Rand: To compute SPDZ-style authenticated sharing $[\![r]\!]$ for a random $r \leftarrow \mathbb{F}_{2^\kappa}$, two parties P_0 and P_1 do the following:

1) For each $b \in \mathbb{F}_2$, P_b samples $\boldsymbol{r}_b \leftarrow \mathbb{F}_2^\kappa$, and then P_b and P_{1-b} call functionality $\mathcal{F}_{\mathsf{aBit}}$ on respective inputs $(\mathsf{auth}, b, \boldsymbol{r}_b, \kappa)$ and $(\mathsf{auth}, b, \kappa)$ to obtain respective outputs $\mathsf{M}_b[\boldsymbol{r}_b]$ and $\mathsf{K}_{1-b}[\boldsymbol{r}_b]$.
2) Two parties define $\langle\!\langle \boldsymbol{r} \rangle\!\rangle = (\langle\!\langle \boldsymbol{r} \rangle\!\rangle_0, \langle\!\langle \boldsymbol{r} \rangle\!\rangle_1)$, where $\langle\!\langle \boldsymbol{r} \rangle\!\rangle_b := (\boldsymbol{r}_b, \mathsf{K}_b[\boldsymbol{r}_{1-b}], \mathsf{M}_b[\boldsymbol{r}_b])$ for each $b \in \mathbb{F}_2$, and run $\langle\!\langle r \rangle\!\rangle := \mathsf{B2F}(\langle\!\langle \boldsymbol{r} \rangle\!\rangle)$ and $[\![r]\!] := \mathsf{Convert}(\langle\!\langle r \rangle\!\rangle)$.

Pack: To pack BDOZ-style authenticated sharings $\{\langle\!\langle x^{(i)} \rangle\!\rangle\}_{i \in [0,\kappa)}$ into one SPDZ-style authenticated sharing $[\![x]\!]$ such that $x = \sum_{i \in [0,\kappa)} x^{(i)} \cdot \mathsf{X}^i \in \mathbb{F}_{2^\kappa}$, both parties run $\langle\!\langle x \rangle\!\rangle := \mathsf{B2F}(\langle\!\langle x^{(0)} \rangle\!\rangle, \ldots, \langle\!\langle x^{(\kappa-1)} \rangle\!\rangle)$ and $[\![x]\!] := \mathsf{Convert}(\langle\!\langle x \rangle\!\rangle)$.

Unpack: To unpack $[\![x]\!]$ into $\{\langle\!\langle x^{(i)} \rangle\!\rangle\}_{i \in [0,\kappa)}$ such that $x = \sum_{i \in [0,\kappa)} x^{(i)} \cdot \mathsf{X}^i \in \mathbb{F}_{2^\kappa}$, two parties P_0 and P_1 do the following:

1) For each $b \in \mathbb{F}_2$, P_b decomposes $x_b \in \mathbb{F}_{2^\kappa}$ in $[\![x]\!]_b$ as $\boldsymbol{x}_b = (x_b^{(0)}, \ldots, x_b^{(\kappa-1)}) \in \mathbb{F}_2^\kappa$ such that $x_b = \sum_{i \in [0,\kappa)} x_b^{(i)} \cdot \mathsf{X}^i$, and then P_b and P_{1-b} call $\mathcal{F}_{\mathsf{aBit}}$ on respective inputs $(\mathsf{auth}, b, \boldsymbol{x}_b, \kappa)$ and $(\mathsf{auth}, b, \kappa)$ to obtain respective outputs $\mathsf{M}_b[\boldsymbol{x}_b]$ and $\mathsf{K}_{1-b}[\boldsymbol{x}_b]$.
2) Both parties define $(\langle\!\langle x^{(0)} \rangle\!\rangle, \ldots, \langle\!\langle x^{(\kappa-1)} \rangle\!\rangle) = \langle\!\langle \boldsymbol{x} \rangle\!\rangle := (\langle\!\langle \boldsymbol{x} \rangle\!\rangle_0, \langle\!\langle \boldsymbol{x} \rangle\!\rangle_1)$, where $\langle\!\langle \boldsymbol{x} \rangle\!\rangle_b := (\boldsymbol{x}_b, \mathsf{K}_b[\boldsymbol{x}_{1-b}], \mathsf{M}_b[\boldsymbol{x}_b])$ for each $b \in \mathbb{F}_2$.
3) Both parties run $\langle\!\langle \tilde{x} \rangle\!\rangle := \mathsf{B2F}(\langle\!\langle \boldsymbol{x} \rangle\!\rangle)$ and $[\![\tilde{x}]\!] := \mathsf{Convert}(\langle\!\langle \tilde{x} \rangle\!\rangle)$,
4) Both parties locally compute $[\![y]\!] := [\![x]\!] - [\![\tilde{x}]\!]$, and run sub-protocol $\Pi_{\mathsf{BatchCheck}}$ (Figure 2) on input $([\![y]\!], 0)$ to check $y = 0$.

Open: To open $x \in \mathbb{F}_{2^\kappa}$ in $[\![x]\!]$, where $[\![x]\!]_b = (x_b, \mathsf{M}_b[x])$ for each $b \in \mathbb{F}_2$, P_0 sends $x_0 \in \mathbb{F}_{2^\kappa}$ to P_1, and P_1 sends $x_1 \in \mathbb{F}_{2^\kappa}$ to P_0 in parallel. Two parties output $\tilde{x} := x_0 \oplus x_1$, and run sub-protocol $\Pi_{\mathsf{BatchCheck}}$ (Figure 2) on input $([\![x]\!], \tilde{x})$ to check $x = \tilde{x}$.

Check: The consistency of values, sent to $\mathcal{F}_{\mathsf{aBit}}$ or two parties, has been checked by running sub-protocol $\Pi_{\mathsf{BatchCheck}}$. All these checks are done in a batch at the end of protocol execution.

Figure 4: Actively secure constant-round 2PC protocol with one-bit leakage in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.

use sub-protocol $\Pi_{\mathsf{BatchCheck}}$ to perform the verification of dual execution, where garbled labels in the dual execution are transformed into BDOZ-style authenticated sharings which are in turn converted into SPDZ-style ones.

Our 2PC protocol requires a garbling scheme (e.g., half-gates [55]) to be compatible with free XOR [36]. In this case, we can set the global key in authenticated sharings as the global offset in free XOR. As a result, garbled labels can be converted to BDOZ-style authenticated sharings. To obtain garbled labels to evaluate a garbled circuit, the two parties maintain the invariant that, for each wire carrying bit x, they hold a BDOZ-style authenticated sharing $\langle\!\langle x \rangle\!\rangle$. Such a sharing can be obtained from (i) calling $\mathcal{F}_{\mathsf{aBit}}$ to authenticate an input bit, or (ii) computing it from garbled labels on the wire. Functionality $\mathcal{F}_{\mathsf{aBit}}$ allows the corrupted party to choose its output, and thus it fails to comply with the uniform distribution of 0-labels on input wires. Thus, we randomize each 0-label with a public randomness r. The correctness of garbling scheme gives $\mathsf{M}_{1-b}[y^{(i)}] = \mathsf{K}_b[y^{(i)}] \oplus y^{(i)} \cdot \Delta_b$ for each $b \in \{0, 1\}$ and output bit $y^{(i)}$. From $\mathsf{lsb}(\Delta_0 \oplus \Delta_1) = 1$, we have

$$y_b^{(i)} \oplus y_{1-b}^{(i)} = \mathsf{lsb}(\mathsf{K}_b[y^{(i)}] \oplus \mathsf{M}_b[y^{(i)}]) \oplus \mathsf{lsb}(\mathsf{K}_{1-b}[y^{(i)}] \oplus \mathsf{M}_{1-b}[y^{(i)}]) = y^{(i)} \cdot \mathsf{lsb}(\Delta_b \oplus \Delta_{1-b}) = y^{(i)}.$$

Reactive 2PC. For the sake of simplicity, we describe the protocol Π_{2PC} (Figure 4) to securely compute a single Boolean circuit. Nevertheless, protocol Π_{2PC} is natural to support reactive computation, as the state information can be transferred via BDOZ-style authenticated sharings, and this protocol realizes the efficient conversion between BDOZ-style authenticated sharings and garbled labels in the dual execution. Specifically, a reactive computation consists of a series of circuits $(\mathcal{C}_0, \ldots, \mathcal{C}_\ell)$, and each circuit $\mathcal{C}_j$ takes as input a state σ_{j-1} and a bit string $x_j \in \{0, 1\}^n$, and outputs an updated state σ_j and a bit string $y_j \in \{0, 1\}^m$. For each

Boolean circuit $\mathcal{C}_j$, our protocol Π_{2PC} is able to take as input $\langle\!\langle \sigma_{j-1} \rangle\!\rangle$ and $\langle\!\langle x_j \rangle\!\rangle$ and then output $\langle\!\langle \sigma_j \rangle\!\rangle$ and $\langle\!\langle y_j \rangle\!\rangle$. When computing circuit $\mathcal{C}_{j+1}$, Π_{2PC} can use $\langle\!\langle \sigma_j \rangle\!\rangle$ and $\langle\!\langle x_{j+1} \rangle\!\rangle$ to compute $\langle\!\langle \sigma_{j+1} \rangle\!\rangle$ and $\langle\!\langle y_{j+1} \rangle\!\rangle$. In this way, protocol Π_{2PC} is able to securely perform the whole reactive computation.

Security. The active security of protocol Π_{2PC} is stated in Theorem 1, and we give its proof in Appendix A.

Theorem 1. *Let $\mathcal{GS}$ be a garbling scheme with obliviousness. Then, protocol Π_{2PC} (Figure 4) securely realizes functionality $\mathcal{F}_{\mathsf{2PC}}$ (Figure 3) against malicious adversaries in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.*

5. Actively Secure Distributed Point Function

In Figure 6, we define an ideal functionality $\mathcal{F}_{\mathsf{DPF}}$ for distributed point function in the active setting. Similarly, it allows the adversary to make a single selective-failure query by inputting a predicate. Then, we present an actively secure two-party protocol Π_{DPF} (shown in Figure 5) to instantiate $\mathcal{F}_{\mathsf{DPF}}$. In this protocol, we suppose that the BDOZ-style and SPDZ-style authenticated sharings input by two parties have been generated by executing the **Input** and **Pack** of protocol Π_{2PC}. Our actively secure DPF protocol builds upon the semi-honest DPF protocol [27], which is based on circular correlation robust (CCR) hash functions [26], [25].

Definition 2. *Let $\mathcal{H} : \{0,1\}^\kappa \to \{0,1\}^\kappa$, χ be a distribution on $\{0,1\}^\kappa$, $\mathcal{F}_{\kappa+1,\kappa}$ be a family of functions with $(\kappa+1)$-bit input and κ-bit output, and $\mathcal{O}^{\mathsf{ccr}}_{\mathcal{H},\Delta}(x,b) := \mathcal{H}(x \oplus \Delta) \oplus b \cdot \Delta$ be an oracle for $x, \Delta \in \{0,1\}^\kappa$ and $b \in \{0,1\}$.*

We say that $\mathcal{H}$ is (t,q,ρ,ϵ)-CCR if for any distinguisher $\mathcal{D}$ running in time at most t and making at most q queries to $\mathcal{O}^{\mathsf{ccr}}_{\mathcal{H},\Delta}(\cdot,\cdot)$, and any χ with min-entropy at least ρ, it holds

$$\left| \Pr_{\Delta \leftarrow \chi}\left[\mathcal{D}^{\mathcal{O}^{\mathsf{ccr}}_{\mathcal{H},\Delta}(\cdot,\cdot)}(1^\kappa) = 1\right] - \Pr_{f \leftarrow \mathcal{F}_{\kappa+1,\kappa}}\left[\mathcal{D}^{f(\cdot,\cdot)}(1^\kappa) = 1\right] \right|$$

is at most ϵ, where $\mathcal{D}$ cannot query both $(x,0)$ and $(x,1)$ for any $x \in \{0,1\}^\kappa$.

Compared to the prior semi-honest DPF protocol [27], our actively secure protocol Π_{DPF} performs a consistency check on all leaf nodes. If a corrupted party sends an incorrect share of a correction word and makes a wrong guess on some prefix of α to remove this error, then the error will propagate in the tree expansion of Π_{DPF} and fail the check. Allowing the adversary to guess a prefix of α leads to one-bit leakage.

Through a simple induction, protocol Π_{DPF} ensures that, for $i \in [0,n]$ and $j \in [0, 2^i)$,

$$(s_b^{(i,j)} \,\|\, t_b^{(i,j)}) \oplus (s_{1-b}^{(i,j)} \,\|\, t_{1-b}^{(i,j)}) = \begin{cases} 0, & j \neq \alpha^{(0)}, \ldots, \alpha^{(i-1)} \\ \Delta_b \oplus \Delta_{1-b}, & \text{otherwise} \end{cases}$$

As $\mathsf{lsb}(\Delta_0 \oplus \Delta_1) = 1$, one can check that $[\![\boldsymbol{u}]\!]$ is a vector of SDPZ-style authenticated sharings on $\boldsymbol{u} = \mathbf{unit}(N, \alpha)$. Moreover, given the above equality, $v^{(j)} := v_b^{(j)} \oplus v_{1-b}^{(j)} = \beta \in \mathbb{F}_{2^\kappa}$ and $\mathsf{M}_b[v^{(j)}] \oplus \mathsf{M}_{1-b}[v^{(j)}] = \mathsf{M}_b[\beta] \oplus \mathsf{M}_{1-b}[\beta]$ if and only if $j = \alpha$. Thus, $[\![\boldsymbol{v}]\!]$ is a vector of SPDZ-style authenticated sharings on $\boldsymbol{v} = \mathbf{unit}(N,\alpha) \cdot \beta$.

Security. We state the security of our DPF protocol Π_{DPF} in Theorem 2, and provide its proof in Appendix B.

Theorem 2. *Let $\mathcal{H}_0$ be a CCR hash function. Then, protocol Π_{DPF} (Figure 5) securely realizes functionality $\mathcal{F}_{\mathsf{DPF}}$ (Figure 6) against malicious adversaries in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.*

6. RAM-based 2PC with Active Security

We present our RAM-based two-party computation functionality $\mathcal{F}_{\mathsf{RAM2PC}}$ in Figure 7, along with its instantiation Π_{RAM2PC} in Figure 8. As discussed in Section 2, we follow the blueprint in Floram [16], which was designed for the semi-honest setting. We use a Read-Only Memory (ROM), a Write-Only Memory (WOM), a refresh procedure synchronizing these two types of memory, and a linear-scan stash to store updates between two refresh procedures. For readers who are familiar with Floram, we note that the main difference is in the structure of ROM and WOM, which now needs to store authenticated shares to prevent active attacks.

ROM and WOM structure. Consider an N-element memory $\boldsymbol{D}$. In our protocol, two parties construct a WOM $\boldsymbol{W}$ from $\boldsymbol{D}$ with each holding $[\![W^{(i)}]\!] := [\![D^{(i)}]\!]$ for every $i \in [0, N)$. They also build a ROM $\boldsymbol{R}$ where each has the same value $R^{(i)} := (D^{(i)} \,\|\, D^{(i)} \cdot \Delta) \oplus F(k_0, i) \oplus F(k_1, i)$ for every $i \in [0,N)$, with $k_0, k_1 \in \mathbb{F}_{2^\kappa}$ held by P_0 and P_1 respectively. Here, $F : \mathbb{F}_{2^\kappa} \times [0,N) \to \mathbb{F}^2_{2^\kappa}$ is a PRF.

To realize a read operation on position α from ROM, two parties input $\langle\!\langle \alpha \rangle\!\rangle$ and a dummy $[\![\beta]\!]$ (e.g., $[\![\beta]\!] = [\![0]\!]$) to Π_{DPF}. Then, $[\![D^{(\alpha)}]\!]$ can be computed from using inner product $\langle \boldsymbol{R}, \{[u^{(i)}]\}_{i \in [0,N)} \rangle$ to select its masked entry $[R^{(\alpha)}]$ and removing mask $[F(k_0,\alpha) \oplus F(k_1,\alpha)]$, which is computed using 2PC.

To implement a write operation from WOM such that $[\![D^{(\alpha)}]\!]$ is updated to $[\![D^{(\alpha)} \oplus \epsilon]\!]$, two parties input $\langle\!\langle \alpha \rangle\!\rangle$ and a random $[\![\beta]\!]$ to Π_{DPF} to obtain $[\![\boldsymbol{u}]\!]$ and $[\![\boldsymbol{v}]\!]$. Then, they open $[\![\epsilon \oplus \beta]\!]$ to obtain $\epsilon \oplus \beta$ and compute the difference $[\![\boldsymbol{\delta}]\!] = [\![\boldsymbol{u}]\!] \cdot (\epsilon \oplus \beta) \oplus [\![\boldsymbol{v}]\!]$ with $\boldsymbol{\delta} = \mathbf{unit}(N, \alpha) \cdot \epsilon \in \mathbb{F}^N_{2^\kappa}$. Two parties update WOM $[\![\boldsymbol{W}]\!] := [\![\boldsymbol{W}]\!] \oplus [\![\boldsymbol{\delta}]\!]$.

Stash-based lookup. After a write operation, the data in ROM will no longer be current. we implement a linear-scan stash, $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$, in secure computation with maximum size σ. It is a temporary storage for all WOM updates that have not yet been applied to ROM. Each element in $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$ includes BDOZ-style authenticated sharings of an index and the updated value at this index. The two parties use $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$ with our ROM and WOM structures as follows:

- For a read operation, the two parties also search for a valid value in $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$ with the index they intend to read. If found, this value, rather than the value from ROM, will returned as the output.
- For a write operation, the two parties clear all values in $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$ with the same index in the current operation. Then, they append this new updated value to $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$.

Refresh procedure. Every write operation updates the authenticated shares in WOM to reflect the most recent content. However, as the stash grows, the cost to access

Protocol Π_{DPF}

This protocol invokes $\Pi_{\mathsf{BatchCheck}}$ (Figure 2) as a sub-protocol.

Initialize: For each $b \in \mathbb{F}_2$, P_b samples $\Delta_b \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{lsb}(\Delta_b) = b$, and sends $(\mathsf{init}, b, \Delta_b)$ to $\mathcal{F}_{\mathsf{aBit}}$.

Protocol inputs: Two parties P_0 and P_1 hold n BDOZ-style authenticated sharings $\langle\!\langle \alpha^{(i)} \rangle\!\rangle = (\langle\!\langle \alpha^{(i)} \rangle\!\rangle_0, \langle\!\langle \alpha^{(i)} \rangle\!\rangle_1)$ for all $i \in [0, n)$ as well as a SPDZ-style authenticated sharing $[\![\beta]\!] = ([\![\beta]\!]_0, [\![\beta]\!]_1)$. Let $N = 2^n$ for some $n \in \mathbb{N}$. Let $\mathcal{H}_0 : \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a CCR hash function and $\mathcal{H}_1 : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ such that $\mathcal{H}_1(x) := \mathcal{H}_0(x) \,\|\, \mathcal{H}_0(x \oplus 1)$.

Generate SPDZ-style authenticated sharings of DPF outputs: Let $\langle\!\langle \alpha^{(i)} \rangle\!\rangle_b = (\alpha_b^{(i)}, \mathsf{K}_b[\alpha_{1-b}^{(i)}], \mathsf{M}_b[\alpha_b^{(i)}])$ and $[\![\beta]\!]_b = (\beta_b, \mathsf{M}_b[\beta])$ for each $b \in \{0,1\}$. The parties P_0 and P_1 do the following.

1) Both parties call $\mathcal{F}_{\mathsf{coin}}$ to sample a public randomness $W \in \mathbb{F}_{2^\kappa}$. Each party P_b sets $(s_b^{(0,0)} \,\|\, t_b^{(0,0)}) := \Delta_b \oplus W \in \{0,1\}^\kappa$.
2) For each $b \in \{0,1\}$, for each $i \in [0, n)$, P_b computes the following:
$$\mathsf{CW}_b^{(i)} := \left(\bigoplus\nolimits_{j \in [0,2^i)} \mathcal{H}_0(s_b^{(i,j)} \,\|\, t_b^{(i,j)})\right) \oplus \Delta_b \oplus \left(\alpha_b^{(i)} \cdot \Delta_b \oplus \mathsf{K}_b[\alpha_{1-b}^{(i)}] \oplus \mathsf{M}_b[\alpha_b^{(i)}]\right) \in \{0,1\}^\kappa,$$
and sends $\mathsf{CW}_b^{(i)}$ to P_{1-b}. For each $i \in [0, n)$, both parties compute $\mathsf{CW}^{(i)} := \mathsf{CW}_0^{(i)} \oplus \mathsf{CW}_1^{(i)}$, and each party P_b computes:
$$\left(s_b^{(i+1,2j)} \,\|\, t_b^{(i+1,2j)}\right) := \mathcal{H}_0\left(s_b^{(i,j)} \,\|\, t_b^{(i,j)}\right) \oplus t_b^{(i,j)} \cdot \mathsf{CW}^{(i)} \text{ for each } j \in [0, 2^i),$$
$$\left(s_b^{(i+1,2j+1)} \,\|\, t_b^{(i+1,2j+1)}\right) := \mathcal{H}_0\left(s_b^{(i,j)} \,\|\, t_b^{(i,j)}\right) \oplus \left(s_b^{(i,j)} \,\|\, t_b^{(i,j)}\right) \oplus t_b^{(i,j)} \cdot \mathsf{CW}^{(i)} \text{ for each } j \in [0, 2^i).$$
3) For each $b \in \{0,1\}$, P_b computes
$$\mathsf{CW}_b^{(n)} := \left(\bigoplus\nolimits_{j \in [0,N)} \mathcal{H}_1(s_b^{(n,j)} \,\|\, t_b^{(n,j)})\right) \oplus (\beta_b \,\|\, \mathsf{M}_b[\beta]) \in \{0,1\}^{2\kappa},$$
and sends $\mathsf{CW}_b^{(n)}$ to P_{1-b}. Then, both parties compute $\mathsf{CW}^{(n)} := \mathsf{CW}_0^{(n)} \oplus \mathsf{CW}_1^{(n)}$. For each $b \in \{0,1\}$, P_b computes
$$[\![u^{(j)}]\!]_b := \left(u_b^{(j)} = t_b^{(n,j)}, \mathsf{M}_b[u^{(j)}] = (s_b^{(n,j)} \,\|\, t_b^{(n,j)})\right) \text{ for each } j \in [0, N),$$
$$[\![v^{(j)}]\!]_b = \left(v_b^{(j)} \,\|\, \mathsf{M}_b[v^{(j)}]\right) := \mathcal{H}_1\left(s_b^{(n,j)} \,\|\, t_b^{(n,j)}\right) \oplus t_b^{(n,j)} \cdot \mathsf{CW}^{(n)} \text{ for each } j \in [0, N).$$
4) As in the **Rand** process of protocol Π_{2PC} (Figure 4), both parties call functionality $\mathcal{F}_{\mathsf{aBit}}$ to generate $[\![r]\!]$ with a random $r \in \mathbb{F}_{2^\kappa}$. Then, both parties call functionality $\mathcal{F}_{\mathsf{coin}}$ to sample a random challenge $\chi \in \mathbb{F}_{2^\kappa}$, and locally compute
$$[\![a]\!] := \sum\nolimits_{j \in [0,N)} \chi^j \cdot [\![u^{(j)}]\!] + \sum\nolimits_{j \in [0,N)} \chi^{N+j} \cdot [\![v^{(j)}]\!] + [\![r]\!].$$
5) As in the **Open** process of protocol Π_{2PC}, both parties open $[\![a]\!]$ to obtain $\tilde{a} = a_0 + a_1 \in \mathbb{F}_{2^\kappa}$ by letting P_0 send a_0 to P_1 and P_1 send a_1 to P_0 in parallel. Then, both parties run sub-protocol $\Pi_{\mathsf{BatchCheck}}$ (Figure 2) on input $([\![a]\!], \tilde{a})$ to check $a = \tilde{a}$.
6) For each $j \in [0, N)$, both parties obtain $[\![u^{(j)}]\!] = ([\![u^{(j)}]\!]_0, [\![u^{(j)}]\!]_1)$ and $[\![v^{(j)}]\!] = ([\![v^{(j)}]\!]_0, [\![v^{(j)}]\!]_1)$.

Figure 5: Actively secure two-party protocol for DPF with one-bit leakage in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.

it will grow; thus we need a refresh procedure to update the content of ROM so that $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$ can be emptied. In this procedure, each party P_b samples an secret key k_b to mask their authenticated shares for every $i \in [0, N)$ to obtain $R_b^{(i)} := (W_b^{(i)} \,\|\, \mathsf{M}_b[W^{(i)}]) \oplus F(k_b, i)$. Then, it sends this masked value to P_{1-b} and computes $\boldsymbol{R} := \boldsymbol{R}_b \oplus \boldsymbol{R}_{1-b}$. Finally, each party inputs its secret key k_b to the secure computation to allow read operations. Note that a refresh procedure requires no secure computation due to the ROM and WOM structure.

Similar to Floram, a refresh procedure is invoked after σ write operations, and the stash-based lookup incurs an $O(\sigma)$ overhead for both read and write operations. So, setting σ to $O(\sqrt{N})$ can achieve the best in the overall complexity.

Full private access. Similar to Floram, protocol Π_{RAM2PC} considers full private access to RAM. A full private access refers to the functionality that, on input an oblivious function F, an element $D^{(\alpha)}$ in the memory, and auxiliary input aux, update $(D^{(\alpha)}, \mathsf{aux}) := F(D^{(\alpha)}, \mathsf{aux})$.

We follow the blueprint of Floram to implement full private accesses from our ROM and WOM structure and stash. More specifically, the two parties do:

1) Perform a read operation to retrieve $D^{(\alpha)}$.
2) Run 2PC protocol to compute $F(D^{(\alpha)}, \mathsf{aux})$.
3) Perform a write operation to update $D^{(\alpha)}$.

Optimization on read operations. In read operations of our protocol, we only utilize $\{[u^{(i)}]\}_{i \in [0,N)}$ from Π_{DPF}, which is independent of β. We use a technique called *tree-trimming optimization* [8], to avoid the expansion of last $\log \kappa$ levels of the tree in our DPF protocol and set $\beta = 2^{(\alpha \bmod \kappa)}$. We note that $[\![\beta]\!]$ can be computed from secure computation, and the bit decomposition of the only non-zero position in $\{[v^{(i)}]\}_{i \in [0,N/2^\kappa)}$ corresponds to that in the above utilized $\{[u^{(i)}]\}_{i \in [0,N)}$. This optimization significantly improves the efficiency of read operations for a large RAM size.

Achieving overall one-bit leakage. Note that our RAM-based 2PC protocol Π_{RAM2PC} calls the interfaces of Π_{2PC} and Π_{DPF}, each of which invokes a consistency check that leads to 1-bit leakage therein. Since the two checks follow

Functionality $\mathcal{F}_{\mathsf{DPF}}$

This functionality initializes two identifier-value lists Bit and Val, where each value in Bit (resp., Val) is an element in $\mathbb{F}_2$ (resp., $\mathbb{F}_{2^\kappa}$). It interacts with two parties P_0 and P_1.

Input: For $b \in \{0,1\}$, upon receiving $(\mathsf{input}, \mathsf{id}, b, x)$ from P_b and $(\mathsf{input}, \mathsf{id}, b)$ from P_{1-b}, where either $x \in \mathbb{F}_2$ or $x \in \mathbb{F}_{2^\kappa}$, set either $\mathsf{Bit}[\mathsf{id}] := x$ or $\mathsf{Val}[\mathsf{id}] := x$ depending on whether x is a bit or not.

Gen: Upon receiving $(\mathsf{gen}, \{\mathsf{id}^{(\alpha_i)}\}_{i\in[0,n)}, \mathsf{id}, \{\mathsf{id}^{(i)}\}_{i\in[0,2N)})$ from P_0 and P_1 where $N = 2^n$, do the following:

1) Compute $\alpha := \sum_{i\in[0,n)} \mathsf{Bit}[\mathsf{id}^{(\alpha_i)}] \cdot 2^i \in [0, N)$ and set $\beta := \mathsf{Val}[\mathsf{id}] \in \mathbb{F}_{2^\kappa}$.
2) Perform the following:
$$(\mathsf{Val}[\mathsf{id}^{(0)}], \ldots, \mathsf{Val}[\mathsf{id}^{(N-1)}]) := \mathbf{unit}(N, \alpha) \in \mathbb{F}_{2^\kappa}^N,$$
$$(\mathsf{Val}[\mathsf{id}^{(N)}], \ldots, \mathsf{Val}[\mathsf{id}^{(2N-1)}]) := \mathbf{unit}(N, \alpha) \cdot \beta \in \mathbb{F}_{2^\kappa}^N.$$

Check: This command is allowed to be called only once. Upon receiving (check) from both parties, do the following:

1) Wait for a predicate $P : \mathbb{F}_2^{|\mathcal{I}|} \times \mathbb{F}_{2^\kappa}^{|\mathcal{J}|} \to \mathbb{F}_2$ from the adversary, where $\mathcal{I}$ (resp., $\mathcal{J}$) is the set of all available identifiers in list Bit (resp., Val).
2) If $P(\{\mathsf{Bit}[\mathsf{id}]\}_{\mathsf{id}\in\mathcal{I}}, \{\mathsf{Val}[\mathsf{id}]\}_{\mathsf{id}\in\mathcal{J}}) = 0$, abort.

Figure 6: Functionality for DPF with one-bit leakage.

Functionality $\mathcal{F}_{\mathsf{RAM2PC}}$

This functionality initialize an identifier-value array Bit, where each entry in Bit is an element in $\mathbb{F}_2$. It interacts with two parties P_0 and P_1.

Initialize: Upon receiving (init, N) from both parties where $N = 2^n$, initialize a memory list $\boldsymbol{D} := (D^{(0)}, ..., D^{(N-1)}) = (0, \ldots, 0) \in \mathbb{F}_{2^\kappa}^N$.

Input: For $b \in \{0,1\}$, upon receiving $(\mathsf{input}, \mathsf{id}, b, x)$ from P_b and $(\mathsf{input}, \mathsf{id}, b)$ from P_{1-b} with $x \in \mathbb{F}_2$, set $\mathsf{Bit}[\mathsf{id}] := x$.

Private RAM read/write access: Upon receiving $(\mathsf{access}, F, \{\mathsf{id}^{(\alpha,i)}\}_{i\in[0,n)}, \{\mathsf{id}^{(\mathsf{aux},i)}\}_{i\in[0,\ell)}, \{\mathsf{id}^{(\mathsf{aux}',i)}\}_{i\in[0,\ell)})$ from P_0 and P_1, where $\ell \in \mathbb{N}$ and $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\kappa \times \{0,1\}^\ell$ is a Boolean circuit, do the following:

1) Compute $\alpha := \sum_{i\in[0,n)} \mathsf{Bit}[\mathsf{id}^{(\alpha,i)}] \cdot 2^i \in [0, N)$ and $\mathsf{aux} := \left(\mathsf{Bit}[\mathsf{id}^{(\mathsf{aux},0)}], \ldots, \mathsf{Bit}[\mathsf{id}^{(\mathsf{aux},\ell-1)}]\right) \in \{0,1\}^\ell$.
2) Compute $(D'^{(\alpha)}, \mathsf{aux}') := F(D^{(\alpha)}, \mathsf{aux})$.
3) Update $D^{(\alpha)} := D'^{(\alpha)}$.
4) Set $\left(\mathsf{Bit}[\mathsf{id}^{(\mathsf{aux}',0)}], \ldots, \mathsf{Bit}[\mathsf{id}^{(\mathsf{aux}',\ell-1)}]\right) := \mathsf{aux}'$.

Check: This command is allowed to be called only once. Upon receiving (check) from both parties, do the following:

1) Wait for a predicate $P : \mathbb{F}_2^{|\mathcal{I}|} \times \mathbb{F}_{2^\kappa}^N \to \mathbb{F}_2$ from the adversary, where $\mathcal{I}$ is the set of all identifiers in Bit.
2) If $P(\{\mathsf{Bit}[\mathsf{id}]\}_{\mathsf{id}\in\mathcal{I}}, \boldsymbol{D}) = 0$, abort.

Figure 7: Functionality for RAM-based 2PC with one-bit leakage.

the same form (i.e., calling sub-protocol $\Pi_{\mathsf{BatchCheck}}$), they can be merged at the end of protocol Π_{RAM2PC}. Intuitively, this merged consistency check is performed only once so that the adversary can only learn a one-bit predicate of all inputs of the honest party and intermediate results from whether the check passes or not. Meanwhile, all intermediate transcripts exchanged by the two parties are indistinguishable from truly random values.

Security. We present our main theorem in Theorem 3. A critical aspect of our protocol Π_{RAM2PC} is its *non-black-box* utilization of authenticated secret sharings generated in our DPF protocol Π_{DPF}. Thus, it will invoke Π_{DPF} directly instead of $\mathcal{F}_{\mathsf{DPF}}$ in a hybrid model. We provide a sketched proof of this theorem in Appendix C.

Theorem 3. *Let $\mathcal{H}_0$ be a CCR hash function and $\mathcal{GS}$ be a garbling scheme whose obliviousness can be based on CCR $\mathcal{H}_0$. Then, protocol Π_{RAM2PC} (Figure 8) securely realizes functionality $\mathcal{F}_{\mathsf{RAM2PC}}$ (Figure 7) against malicious adversaries in the ($\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{F}_{\mathsf{coin}}$, $\mathcal{F}_{\mathsf{com}}$)-hybrid model.*

7. Evaluation

We would like to study the performance of our protocol in the following four aspects.

Q1 What is the cost of our actively secure protocol compared to the state-of-the-art semi-honest ones?

Q2 How many improvements in efficiency are there when comparing our protocol to state-of-the-art maliciously secure ones?

Q3 What is the bottleneck of our protocol in different scenarios and array sizes?

Q4 What is the practical performance when putting our protocol in end-to-end applications?

To answer these questions, we implement our protocol and made code available in EMP [47]. Below, we provide implementation details and setup, with the answers to all questions.

7.1. Experimental Setup

We implement all of our protocol in C++ based on EMP toolkit [47]. We instantiate $\mathcal{F}_{\mathsf{COT}}$ using Ferret OT [53] and instantiate PRFs using AES-128. All code is compiled using `gcc` version 11.4.0, with `-O3` optimization flag enabled.

Our benchmark is performed on a pair of AWS `R5.8xlarge` instances, each with 32 vCPUs and 256 GB memory. To simulate a LAN network, we use two instances in the same availability zone, and manually limit the network bandwidth to 2 Gbps; the round-trip time (RTT) between two instances is roughly 0.1 ms. To simulate a WAN network, we limit the bandwidth to 100 Mbps and set RTT to 60 ms using `tc` command. These settings are similar to related prior works [45].

If not specified otherwise, we vary the number of elements in the array from 2^{12} to 2^{29}, and use an element size of 8 bytes, corresponding to up to 4 GiB of data. Read-only operations refer to reading an element in the array without any modification, and full-access operations support both reading an element from the array and writing a new value back. Our implementation of the dual-execution-based 2PC is always single-threaded; other parts of our protocol are multi-threaded when possible. We use 16 threads by default. We set the stash size σ of our protocol to $\sqrt{N/T}/20$ in the LAN setting, and $\sigma := \sqrt{N}/16$ in the WAN setting, where N is the size of RAM and T is the number of threads.

Protocol Π_{RAM2PC}

This protocol invokes Π_{2PC} (Figure 4) and Π_{DPF} (Figure 5) as two sub-protocols, and maintains three memories: the ROM $\boldsymbol{R}$, WOM $[\![\boldsymbol{W}]\!]$ and stash $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$. Let PRF : $\{0,1\}^\kappa \times [0,N) \to \{0,1\}^{2\kappa}$ be a pseudo-random function (PRF) and $\sigma \in \mathbb{N}$ denote the maximum number of entries in a stash.

Initialize: Two parties P_0 and P_1 execute **Initialize** of sub-protocol Π_{2PC} to initialize two global keys $\Delta_0 \in \mathbb{F}_{2^\kappa}$ and $\Delta_1 \in \mathbb{F}_{2^\kappa}$. Then, both parties execute as follows:

1) Both parties set $[\![W^{(i)}]\!] := [\![0]\!]$ for each $i \in [0,N)$ to initialize WOM $[\![\mathbf{W}]\!]$, and initialize stash $\langle\!\langle \boldsymbol{S} \rangle\!\rangle := \emptyset$.
2) Both parties run the following **Refresh** procedure to initialize ROM $\boldsymbol{R} \in \mathbb{F}_{2^\kappa}^N$.
3) For an initial auxiliary input aux, both parties execute **Input** of sub-protocol Π_{2PC} to generate $\langle\!\langle \mathsf{aux} \rangle\!\rangle$.

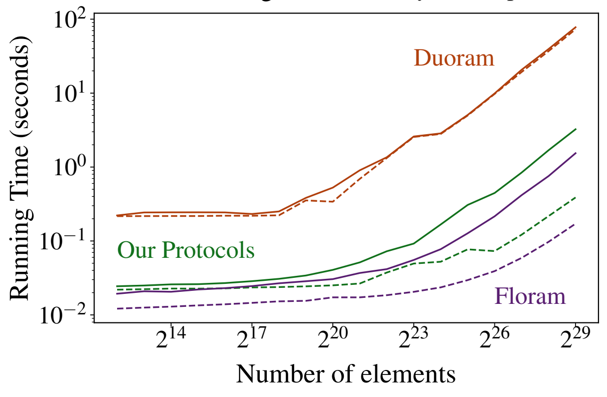
Full private access: To obliviously read or write an entry in the α-th position with $\alpha \in [0,N)$ and $N = 2^n$, P_0 and P_1 hold $\langle\!\langle \alpha \rangle\!\rangle = (\langle\!\langle \alpha^{(0)} \rangle\!\rangle, \ldots, \langle\!\langle \alpha^{(n-1)} \rangle\!\rangle)$ such that $\alpha^{(i)} \in \{0,1\}$ for $i \in [0,n)$ and $\sum_{i\in[0,n)} \alpha^{(i)} \cdot 2^i = \alpha$, and then do the following:

1) Both parties execute **Rand** of sub-protocol Π_{2PC} to generate $[\![\beta]\!]$ with a random element $\beta \in \mathbb{F}_{2^\kappa}$.
2) Both parties execute sub-protocol Π_{DPF} on the input $\{\langle\!\langle \alpha^{(i)} \rangle\!\rangle\}_{i\in[0,n)}$ and $[\![\beta]\!]$ to obtain $[\![\boldsymbol{u}]\!]$ and $[\![\boldsymbol{v}]\!]$ such that $\boldsymbol{u}$ (resp., $\boldsymbol{v}$) is an unit vector with exactly one nonzero entry $u^{(\alpha)} = 1$ (resp., $v^{(\alpha)} = \beta$).
3) Both parties locally compute a pair of unauthenticated additive sharings $([c],[d]) := \sum_{i\in[0,N)} (R^{(i)}[0], R^{(i)}[1]) \cdot [u^{(i)}]$, where $R^{(i)} = (R^{(i)}[0], R^{(i)}[1]) \in \{0,1\}^{2\kappa}$ for $i \in [0,N)$, and $[u^{(i)}]$ for all $i \in [0,N)$ are the additive secret sharings defined in $[\![\boldsymbol{u}]\!]$.
4) P_0 and P_1 execute **Eval** of sub-protocol Π_{2PC} on the input $(\langle\!\langle k_0 \rangle\!\rangle, \langle\!\langle k_1 \rangle\!\rangle, \langle\!\langle \alpha \rangle\!\rangle, \langle\!\langle \mathsf{aux} \rangle\!\rangle)$ to perform the following computation:
   a) If there exists an entry $(\langle\!\langle \alpha \rangle\!\rangle, \langle\!\langle x \rangle\!\rangle)$ in $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$, set $\langle\!\langle y \rangle\!\rangle := \langle\!\langle x \rangle\!\rangle$. Otherwise, compute $([y],[y\cdot\Delta]) := \mathsf{PRF}(\langle\!\langle k_0 \rangle\!\rangle, \langle\!\langle \alpha \rangle\!\rangle) \oplus \mathsf{PRF}(\langle\!\langle k_1 \rangle\!\rangle, \langle\!\langle \alpha \rangle\!\rangle) \oplus ([c]_0,[d]_0) \oplus ([c]_1,[d]_1)$, set $[\![y]\!] = ([y],[y\cdot\Delta])$ and unpack $[\![y]\!]$ as $\langle\!\langle y \rangle\!\rangle$. (Note that both parties can run **Unpack** of sub-protocol Π_{2PC} on the input $[\![y]\!]$ to generate $\langle\!\langle y \rangle\!\rangle$.)
   b) Compute $(\langle\!\langle y' \rangle\!\rangle, \langle\!\langle \mathsf{aux}' \rangle\!\rangle) := F(\langle\!\langle y \rangle\!\rangle, \langle\!\langle \mathsf{aux} \rangle\!\rangle)$.
   c) If there exists an entry $(\langle\!\langle \alpha \rangle\!\rangle, \langle\!\langle x \rangle\!\rangle)$ in $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$, set the entry as $(\perp,\perp)$ and add $(\langle\!\langle \alpha \rangle\!\rangle, \langle\!\langle y' \rangle\!\rangle)$ to $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$.
5) Both parties update $\langle\!\langle \mathsf{aux} \rangle\!\rangle$ as $\langle\!\langle \mathsf{aux}' \rangle\!\rangle$, and run **Pack** of sub-protocol Π_{2PC} on the input $\langle\!\langle y \rangle\!\rangle$ to generate $[\![y]\!]$.
6) P_0 and P_1 run $[\![y']\!] := \mathsf{Convert}(\langle\!\langle y' \rangle\!\rangle)$, and then locally compute $[\![\delta]\!] := [\![\beta]\!] \oplus [\![y]\!] \oplus [\![y']\!]$. Then, both parties execute **Open** of sub-protocol Π_{2PC} open $[\![\delta]\!]$ to obtain $\delta \in \{0,1\}^\kappa$.
7) For $i \in [0,N)$, both parties locally compute and update $[\![W^{(i)}]\!] := [\![W^{(i)}]\!] \oplus \delta \cdot [\![u^{(i)}]\!] \oplus [\![v^{(i)}]\!]$, meaning that $[\![\boldsymbol{W}]\!]$ is updated.
8) If the number of entries in stash $\langle\!\langle \boldsymbol{S} \rangle\!\rangle$ is identical to σ, both parties run the following **Refresh** procedure.

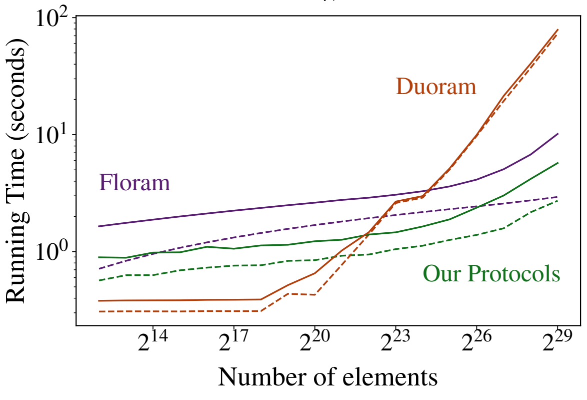
Refresh: Both parties clear the stash, i.e., set $\langle\!\langle \boldsymbol{S} \rangle\!\rangle := \emptyset$, and then do the following:

1) For each $b \in \{0,1\}$, P_b samples $k_b = (k_b^{(0)}, \ldots, k_b^{(\kappa-1)}) \leftarrow \{0,1\}^\kappa$, which is used as a PRF key, and both parties execute **Input** of sub-protocol Π_{2PC} on the input bits $k_b^{(i)}$ for $i \in [0,\kappa)$ to generate $\langle\!\langle k_b \rangle\!\rangle$.
2) For each $b \in \{0,1\}$, for each $i \in [0,N)$, P_b computes $R_b^{(i)} = (R_b^{(i)}[0], R_b^{(i)}[1]) := [\![W^{(i)}]\!]_b \oplus \mathsf{PRF}(k_b, i) \in \{0,1\}^{2\kappa}$ and sends $R_b^{(i)}$ to P_{1-b}.
3) Both parties compute $\boldsymbol{R}$ by setting $R^{(i)} := R_0^{(i)} \oplus R_1^{(i)} \in \{0,1\}^{2\kappa}$ for each $i \in [0,N)$, and also obtain $\langle\!\langle k_0 \rangle\!\rangle$ and $\langle\!\langle k_1 \rangle\!\rangle$.

Figure 8: Actively secure protocol for RAM-based 2PC with one-bit leakage.



(a) LAN setting.



(b) WAN setting.

Figure 9: **Wall-clock time of an access operation using our protocol, Floram and Duoram in LAN and WAN settings.** Solid lines are for full-access operations, which support both read and write operations; dashed lines are for read-only operations. Each entry in the array has 8 bytes. All timings are average of a sufficiently large number of accesses.

7.2. Overhead Compared to Semi-Honest Protocols

Floram [16] and Duoram [45] are the state-of-the-art RAM-based 2PC protocols secure against semi-honest adversaries in the LAN and WAN settings, respectively. To understand the overhead of our protocol, we compare our cost with the cost of Floram and Duoram in both network settings and on both read-only and full-access operations.

Since some protocols need cost amortization, we run a sufficient number of accesses and report the average wall-clock time across all operations.

We show the result in Figure 9. In the LAN setting, Floram is approximately $2\times$ faster than our protocol on both read-only and full-access operations. On the other hand, our protocol is about $1-2$ orders of magnitude faster than Duoram on both operations. In the WAN setting, our protocol, interestingly, is roughly $2\times$ faster than Floram on full-access operations and has an advantage on read-only operations. Compared with Duoram, it is slower when the array size is less than 2^{21} elements. Regarding the impact of access operation types, full-access operations require about a 50% extra cost compared to read-only operations in the WAN setting and incur a five-fold overhead in the LAN setting in our protocol and Floram. For Duoram, their read-only and full access have similar performance.

Our protocol only imposes a constant overhead on top of Floram; thus its performance is similar to Floram's. In particular, we need roughly $2\times$ overhead in both secure computation and local expansion of trees needed in DPF. At the same time, since our underlying malicious secure DPF protocol integrates optimizations shown in Half-Tree [27], our protocol shows nearly no overhead for active security compared to Floram. When compared to Duoram, our protocol needs $O(\log N)$ rounds for an access operation to an array of size N, but Duoram has an amortized constant roundtrips by performing an offline phase for $\log N$ operations together. As a result, our protocol performs worse than Duoram when the array size is small in the WAN setting. However, when the array size is sufficiently large or in the LAN setting, our protocol still outperforms Duoram because computation is the bottleneck, not the roundtrip. Regardless, one can conclude that our active protocol is competitive with state-of-the-art semi-honest protocols.

7.3. Comparison to SOTA Active Protocols

The state-of-the-art maliciously secure protocol is KY18 [35]. As mentioned before, this protocol uses a generic compilation from malicious MPC [37], [38], [32] and ORAM protocol [46] to RAM-based MPC. Their actively secure MPC is instantiated by a SPDZ protocol with an offline and an online phase. We contacted the authors and utilized their script to obtain an accurate estimation of the cost of KY18. We first calculate the number of field multiplication triples for an access operation from the number of AND gates required; then we estimate the wall-clock time based on the state-of-the-art triple generation protocol [33] instead of MASCOT used in KY18 for the most up-to-date estimation. We also compare against a possible combination by using the authenticated garbling [48] (dubbed KY18-AGC) instead of SPDZ-BMR. KY18 noted that such a combination might not be compatible with their memory optimizations and put it an open problem; nevertheless, it represents a hypothetical best possible solution. Since KY18 has the same complexity on read and write operations, we compare one full-access operation of our protocol (which supports both), and one write operation of KY18 and KY18-AGC. Note that the results for KY18 and KY18-AGC are depicted for array sizes ranging from 2^9 to 2^{24} 8-byte elements; this is because KY18 only provides the number of AND gates required for these array sizes.

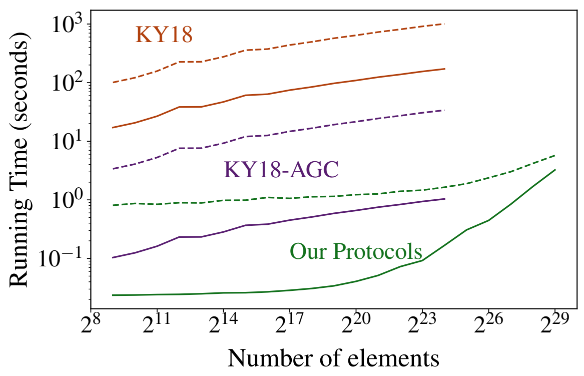


Figure 10: **Wall-clock time of a full-access operation using our protocol, KY18 and KY18-AGC in both network settings.** Solid lines are in LAN settings; dashed lines are in WAN settings.

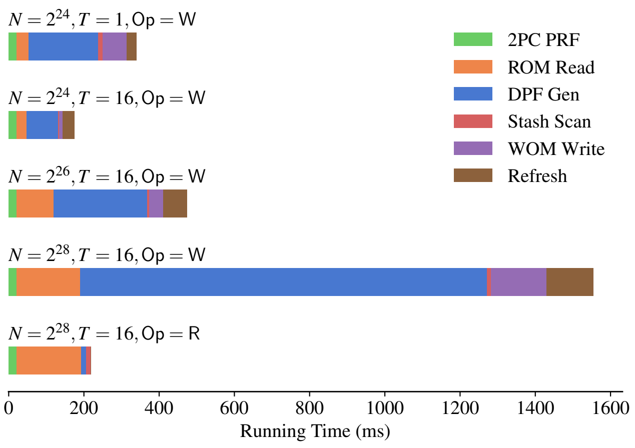
We report the comparison result in Figure 10 and can observe that our protocol consistently outperforms KY18 by about two orders of magnitude in both network settings. It is also nearly one order of magnitude faster than the hypothetical result KY18-AGC. However, it's important to mention that our protocol has a one-bit leakage to the overall protocol (not each access operation); thus it presents a trade-off between security and efficiency. Additionally, we observe a trend that the performance gap of our protocol between LAN and WAN settings narrows as the array size increases due to the increasing cost of computation. This does not happen for KY18, as their computation complexity is also sublinear. So, for the commonly used array size in MPC applications, our protocol provides a much better trade-off and enables efficient access with just one-bit leakage.

7.4. Microbenchmarks

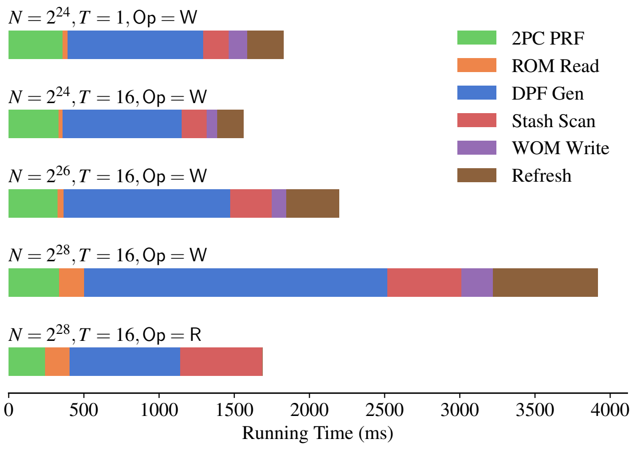
We delve into our protocol and analyze the cost of each part of an access operation. We divide the cost of an access operation into six steps: 1) DPF generation; 2) memory access to read secret shares from public encrypted ROM; 3) secure evaluation of PRFs; 4) secure linear scan of stash; 5) memory access to write authenticated secret shares to WOM; 6) refresh cost amortized over each full-access operation. Note that a full-access operation has all these 6 steps while a read-only operation only has the first 4 steps.

We record the average wall-clock time of each part for a full-access operation as well as a read-only operation for array of size 2^{24}, 2^{26}, and 2^{28} in different network settings.

Figure 11a presents the cost breakdown of an access operation in different scenarios with LAN settings. In a full-access operation, the cost of securely computing PRF remains constant, but other costs increase as the array size increases. Notably, the cost of computing PRF becomes the bottleneck with small array sizes, but it is minor when the size is sufficiently large. Refresh and stash scanning costs take an insignificant portion of the total costs across all scenarios. Conversely, the costs of DPF generation and



(a) LAN setting.



(b) WAN setting.

Figure 11: **Cost breakdown of an access operation in different scenarios.** N is the size of an array and T is the maximum number of threads used. `Op=R` denotes a read-only operation, and `Op=W` denotes a full-access operation. Elements in the array have a size of 8 bytes. The wall-clock time of each part is averaged from a number of access operations, which is multiple of the refresh period.

memory access increase approximately linearly with the array size, becoming the primary expense when the array size is sufficiently large. Applying parallelization can efficiently mitigate it, and our protocol with 16 threads performs approximately twice as fast as using only one thread by utilizing multithreading in local computation. We also notice the bottleneck in a read-only operation differs from that in a full-access operation, where scanning ROM memory rather than DPF generation becomes the primary cost in read-only operations since our protocol requires a relatively small DPF tree with tree-trimming optimization in Section 6.

In WAN settings, as illustrated in Figure 11b, the cost distribution significantly differs. Local memory access takes a minor fraction of the total cost of an access operation, thus multithreading has a minor effect on mitigating the overall costs. Meanwhile, 2PC (including PRF and stash scanning) along with the refresh procedure incur a considerably higher cost due to bandwidth limitations, and become the bottlenecks of an access operation for large array sizes. While DPF generation remains the principal cost factor, unlike in LAN settings, it does not increase linearly with the array size. This is because the latency from round complexity contributes significantly to the cost since there is an $O(\log N)$ round complexity on DPF generation. Compared to a full-access operation, DPF generation in a read-only operation is faster since roughly $\log \kappa$ rounds are eliminated from tree-trimming optimization.

7.5. RAM Applications

To benchmark the performance of our protocol in real-world scenarios, we extend our protocol to several RAM applications: oblivious binary search, stable matching and the scrypt function. We report all the results in Table 1 and present our experiments in detail below.

Binary search. RAM-based MPC protocols can obliviously execute binary searches on an array with a few access operations. The performance of such searches in an array has been benchmarked in various protocols [23], [56], [16].

Benchmark	Parameters	LAN (sec)	WAN (sec)
Binary Search	1 search	81.68	185.2
	2^5 searches	120.42	1118.7
	2^{10} searches	1894.23	30989.6
Gale-Shapley	2^3 pairs	9.2	268.7
	2^6 pairs	670.3	19975.0
	2^9 pairs	44476.1	*about 19 days*
Scrypt	$N = 2^5, r = 8$	42.1	1159.1
	$N = 2^{10}, r = 1$	167.6	4721.4
	$N = 2^{10}, r = 8$	1396.4	41022.7

TABLE 1: **Summary of benchmark results.** All results are wall-clock time in seconds if not specified. The array contains 2^{25} 8-byte elements for all binary search benchmarks.

We extend our protocol and implement an oblivious binary search. It needs $O(\log N)$ read-only operations for each search on an array of N elements. To evaluate it, we set the array size of 2^{25} 8-byte elements, and record the wall-clock time of executing 1, 2^5, and 2^{10} searches on both network settings. This time is composed of initialization of RAM structure and performing read-only operations.

Stable matching. Gale-Shapley algorithm [18] is a typical solution to the stable matching problem. We extend our RAM-based 2PC protocol to implement an oblivious version of the Gale-Shapley algorithm, aimed at benchmarking performance in a complex, end-to-end application.

Our implementation closely follows the origin Gale-Shapley algorithm, and it requires $O(n^2)$ access operations of arrays size of up to n^2 elements for matching n pairs.

We evaluate the wall-clock time for full protocol execution, including tests with 8, 64 and 512 pairs in LAN settings and 8 and 64 pairs in WAN settings. We also estimate the wall-clock time for 512 pairs in WAN settings based on the microbenchmark results mentioned above. We notice since the array size is not sufficiently large, securely computing PRF is the primary overhead for access operations. Consequently, our protocol shows a relatively poor performance when the number of pairs is small, but becomes efficient as the pair count increases. Doerner et al. [15] proposed serveral optimzied algorithms for stable matching, all of which can be implemented using the building blocks proposed in this paper as well.

Scrypt. Scrypt is a key derivation function intended to provide resistance against parallelized brute-force attacks by using a large amount of memory. We implement an oblivious scrypt function to enable securely executing some cryptographic functions using RAM-based 2PC.

We denote the cost factor of a scrypt function by N, the parallelization factor by p, and the block size factor by r. Our implementation requires $O(Nr)$ read-only operations of an array size of Nr 1Kbit elements for computing a function. We select three representative parameters [42] and benchmark each of them in both network settings.

8. Future Work

Castro and Polychroniadou [14] proposed a malicious DPF protocol in a weaker model with two servers and one client where at most one party can be corrupted. It is a future work to apply our optimizations to their model. It would also be interesting to apply this protocol to improve the concrete efficiency of slient correlation generation.

9. Acknowledgements

Work of Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037 and 61932019). Work of Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 62125204 and 61872236). Yu Yu's work has also been supported by the New Cornerstone Science Foundation through the XPLORER PRIZE. Work of Xiao Wang is supported by NSF awards #2236819 and #2310927.

References

[1] I. Abraham, C. W. Fletcher, K. Nayak, B. Pinkas, and L. Ren, "Asymptotically tight bounds for composing ORAM with PIR," in *PKC 2017, Part I*, ser. LNCS, S. Fehr, Ed., vol. 10174. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Mar. 28–31, 2017, pp. 91–120.

[2] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers, "Smcql: Secure querying for federated databases," *Proc. VLDB Endow.*, p. 673–684, feb 2017.

[3] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *ACM CCS 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. Raleigh, NC, USA: ACM Press, Oct. 16–18, 2012, pp. 784–796.

[4] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *EUROCRYPT 2011*, ser. LNCS, K. G. Paterson, Ed., vol. 6632. Tallinn, Estonia: Springer, Heidelberg, Germany, May 15–19, 2011, pp. 169–188.

[5] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl, "Correlated pseudorandomness from expand-accumulate codes," in *CRYPTO 2022, Part II*, ser. LNCS, Y. Dodis and T. Shrimpton, Eds., vol. 13508. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 15–18, 2022, pp. 603–633.

[6] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round OT extension and silent non-interactive secure computation," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. London, UK: ACM Press, Nov. 11–15, 2019, pp. 291–308.

[7] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators from ring-LPN," in *CRYPTO 2020, Part II*, ser. LNCS, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 17–21, 2020, pp. 387–416.

[8] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. Vienna, Austria: ACM Press, Oct. 24–28, 2016, pp. 1292–1303.

[9] S. S. Burra, E. Larraia, J. B. Nielsen, P. S. Nordholt, C. Orlandi, E. Orsini, P. Scholl, and N. P. Smart, "High-performance multi-party computation for binary circuits based on oblivious transfer," *Journal of Cryptology*, vol. 34, no. 3, p. 34, Jul. 2021.

[10] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, Jan. 2000.

[11] H. Cui, X. Wang, K. Yang, and Y. Yu, "Actively secure half-gates with minimum overhead under duplex networks," in *EUROCRYPT 2023, Part II*, ser. LNCS, C. Hazay and M. Stam, Eds., vol. 14005. Lyon, France: Springer, Heidelberg, Germany, Apr. 23–27, 2023, pp. 35–67.

[12] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits," in *ESORICS 2013*, ser. LNCS, J. Crampton, S. Jajodia, and K. Mayes, Eds., vol. 8134. Egham, UK: Springer, Heidelberg, Germany, Sep. 9–13, 2013, pp. 1–18.

[13] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2012, pp. 643–662.

[14] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *EUROCRYPT 2022, Part I*, ser. LNCS, O. Dunkelman and S. Dziembowski, Eds., vol. 13275. Trondheim, Norway: Springer, Heidelberg, Germany, May 30 – Jun. 3, 2022, pp. 150–179.

[15] J. Doerner, D. Evans, and a. shelat, "Secure stable matching at scale," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. Vienna, Austria: ACM Press, Oct. 24–28, 2016, pp. 1602–1613.

[16] J. Doerner and a. shelat, "Scaling ORAM for secure computation," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. Dallas, TX, USA: ACM Press, Oct. 31 – Nov. 2, 2017, pp. 523–535.

[17] S. Faber, S. Jarecki, S. Kentros, and B. Wei, "Three-party ORAM for secure computation," in *ASIACRYPT 2015, Part I*, ser. LNCS, T. Iwata and J. H. Cheon, Eds., vol. 9452. Auckland, New Zealand: Springer, Heidelberg, Germany, Nov. 30 – Dec. 3, 2015, pp. 360–385.

[18] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *American Mathematical Monthly*, 1962.

[19] C. Gentry, K. A. Goldman, S. Halevi, C. S. Jutla, M. Raykova, and D. Wichs, "Optimizing ORAM and using it efficiently for secure computation," in *PETS 2013*, ser. LNCS, E. De Cristofaro and M. K. Wright, Eds., vol. 7981. Bloomington, IN, USA: Springer, Heidelberg, Germany, Jul. 10–12, 2013, pp. 1–18.

[20] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *EUROCRYPT 2014*, ser. LNCS, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Copenhagen, Denmark: Springer, Heidelberg, Germany, May 11–15, 2014, pp. 640–658.

[21] O. Goldreich, *Foundations of Cryptography: Basic Applications*. Cambridge, UK: Cambridge University Press, 2004, vol. 2.

[22] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[23] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis, "Secure two-party computation in sublinear (amortized) time," in *ACM CCS 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. Raleigh, NC, USA: ACM Press, Oct. 16–18, 2012, pp. 513–524.

[24] S. D. Gordon, J. Katz, and X. Wang, "Simple and efficient two-server ORAM," in *ASIACRYPT 2018, Part III*, ser. LNCS, T. Peyrin and S. Galbraith, Eds., vol. 11274. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, Dec. 2–6, 2018, pp. 141–157.

[25] C. Guo, J. Katz, X. Wang, C. Weng, and Y. Yu, "Better concrete security for half-gates garbling (in the multi-instance setting)," in *CRYPTO 2020, Part II*, ser. LNCS, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 17–21, 2020, pp. 793–822.

[26] C. Guo, J. Katz, X. Wang, and Y. Yu, "Efficient and secure multiparty computation from fixed-key block ciphers," in *2020 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 18–21, 2020, pp. 825–841.

[27] X. Guo, K. Yang, X. Wang, W. Zhang, X. Xie, J. Zhang, and Z. Liu, "Half-tree: Halving the cost of tree expansion in COT and DPF," in *EUROCRYPT 2023, Part I*, ser. LNCS, C. Hazay and M. Stam, Eds., vol. 14004. Lyon, France: Springer, Heidelberg, Germany, Apr. 23–27, 2023, pp. 330–362.

[28] A. Hamlin and M. Varia, "Two-server distributed ORAM with sublinear computation and constant rounds," in *PKC 2021, Part II*, ser. LNCS, J. Garay, Ed., vol. 12711. Virtual Event: Springer, Heidelberg, Germany, May 10–13, 2021, pp. 499–527.

[29] C. Hazay, P. Scholl, and E. Soria-Vazquez, "Low cost constant round MPC combining BMR and oblivious transfer," in *ASIACRYPT 2017, Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Hong Kong, China: Springer, Heidelberg, Germany, Dec. 3–7, 2017, pp. 598–628.

[30] Y. Huang, J. Katz, and D. Evans, "Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution," in *2012 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 21–23, 2012, pp. 272–284.

[31] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *CRYPTO 2015, Part I*, ser. LNCS, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9215. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 16–20, 2015, pp. 724–741.

[32] ——, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. Vienna, Austria: ACM Press, Oct. 24–28, 2016, pp. 830–842.

[33] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," in *EUROCRYPT 2018, Part III*, ser. LNCS, J. B. Nielsen and V. Rijmen, Eds., vol. 10822. Tel Aviv, Israel: Springer, Heidelberg, Germany, Apr. 29 – May 3, 2018, pp. 158–189.

[34] M. Keller and P. Scholl, "Efficient, oblivious data structures for MPC," in *ASIACRYPT 2014, Part II*, ser. LNCS, P. Sarkar and T. Iwata, Eds., vol. 8874. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 7–11, 2014, pp. 506–525.

[35] M. Keller and A. Yanai, "Efficient maliciously secure multiparty computation for RAM," in *EUROCRYPT 2018, Part III*, ser. LNCS, J. B. Nielsen and V. Rijmen, Eds., vol. 10822. Tel Aviv, Israel: Springer, Heidelberg, Germany, Apr. 29 – May 3, 2018, pp. 91–124.

[36] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP 2008, Part II*, ser. LNCS, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., vol. 5126. Reykjavik, Iceland: Springer, Heidelberg, Germany, Jul. 7–11, 2008, pp. 486–498.

[37] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant round multi-party computation combining BMR and SPDZ," in *CRYPTO 2015, Part II*, ser. LNCS, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9216. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 16–20, 2015, pp. 319–338.

[38] Y. Lindell, N. P. Smart, and E. Soria-Vazquez, "More efficient constant-round multi-party computation from BMR and SHE," in *TCC 2016-B, Part I*, ser. LNCS, M. Hirt and A. D. Smith, Eds., vol. 9985. Beijing, China: Springer, Heidelberg, Germany, Oct. 31 – Nov. 3, 2016, pp. 554–581.

[39] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 17–21, 2015, pp. 359–376.

[40] S. Lu and R. Ostrovsky, "Distributed oblivious RAM for secure two-party computation," in *TCC 2013*, ser. LNCS, A. Sahai, Ed., vol. 7785. Tokyo, Japan: Springer, Heidelberg, Germany, Mar. 3–6, 2013, pp. 377–396.

[41] P. Mohassel and M. Franklin, "Efficiency tradeoffs for malicious two-party computation," in *PKC 2006*, ser. LNCS, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958. New York, NY, USA: Springer, Heidelberg, Germany, Apr. 24–26, 2006, pp. 458–473.

[42] C. Percival, "Stronger key derivation via sequential memory-hard functions," 2009.

[43] M. Rosulek and L. Roy, "Three halves make a whole? Beating the half-gates lower bound for garbled circuits," in *CRYPTO 2021, Part I*, ser. LNCS, T. Malkin and C. Peikert, Eds., vol. 12825. Virtual Event: Springer, Heidelberg, Germany, Aug. 16–20, 2021, pp. 94–124.

[44] L. Roy, "SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model," in *CRYPTO 2022, Part I*, ser. LNCS, Y. Dodis and T. Shrimpton, Eds., vol. 13507. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 15–18, 2022, pp. 657–687.

[45] A. Vadapalli, R. Henry, and I. Goldberg, "Duoram: A bandwidth-efficient distributed ORAM for 2- and 3-party computation," in *USENIX Security 2023*. Anaheim, CA, USA: USENIX Association, Aug. 9–11, 2023, pp. 3907–3924.

[46] X. Wang, T.-H. H. Chan, and E. Shi, "Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound," in *ACM CCS 2015*, I. Ray, N. Li, and C. Kruegel, Eds. Denver, CO, USA: ACM Press, Oct. 12–16, 2015, pp. 850–861.

[47] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[48] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. Dallas, TX, USA: ACM Press, Oct. 31 – Nov. 2, 2017, pp. 21–37.

[49] ——, "Global-scale secure multiparty computation," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. Dallas, TX, USA: ACM Press, Oct. 31 – Nov. 2, 2017, pp. 39–56.

[50] X. S. Wang, Y. Huang, T.-H. H. Chan, a. shelat, and E. Shi, "SCORAM: Oblivious RAM for secure computation," in *ACM CCS 2014*, G.-J. Ahn, M. Yung, and N. Li, Eds. Scottsdale, AZ, USA: ACM Press, Nov. 3–7, 2014, pp. 191–202.

[51] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *2021 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 24–27, 2021, pp. 1074–1091.

[52] K. Yang, X. Wang, and J. Zhang, "More efficient MPC from improved triple generation and authenticated garbling," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. Virtual Event, USA: ACM Press, Nov. 9–13, 2020, pp. 1627–1646.

[53] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. Virtual Event, USA: ACM Press, Nov. 9–13, 2020, pp. 1607–1626.

[54] A. C.-C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th FOCS*. Toronto, Ontario, Canada: IEEE Computer Society Press, Oct. 27–29, 1986, pp. 162–167.

[55] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *EUROCRYPT 2015, Part II*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9057. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 26–30, 2015, pp. 220–250.

[56] S. Zahur, X. S. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz, "Revisiting square-root ORAM: Efficient random access in multi-party computation," in *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2016, pp. 218–234.

Appendix A. Proof of Theorem 1

Theorem 1. *Let $\mathcal{GS}$ be a garbling scheme with obliviousness. Then, protocol Π_{2PC} (Figure 4) securely realizes functionality $\mathcal{F}_{\mathsf{2PC}}$ (Figure 3) against malicious adversaries in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.*

Proof (Sketch). As the two parties are symmetric in Π_{2PC}, we w.l.o.g. assume that P_b is corrupted and P_{1-b} is honest for some fixed $b \in \{0,1\}$. Simulator $\mathcal{S}_{\mathsf{2PC}}$ extracts $\Delta_b \in \mathbb{F}_{2^\kappa}$ from emulated $\mathcal{F}_{\mathsf{aBit}}$ (and aborts if $\mathsf{lsb}(\Delta_b) \neq b$) and maintains the secret share of corrupted P_b for each BDOZ- or SPDZ-style authenticated sharing.

In particular, maintaining such shares is straightforward for all input bits (in **Input**), random values (in **Rand**) and unpacked bits (in **Unpack**) since $\mathcal{S}_{\mathsf{2PC}}$ emulates $\mathcal{F}_{\mathsf{aBit}}$. In **Eval**, $\mathcal{S}_{\mathsf{2PC}}$ sends a uniformly random GC_{1-b} (which is indistinguishable from a real one due to obliviousness) to corrupted P_b, and uses Δ_b and the extracted secret shares of P_b to evaluate garbled circuit GC_{1-b} to compute P_b's secret shares of the BDOZ-style authenticated sharings of circuit outputs. As **Pack** only includes local computation, $\mathcal{S}_{\mathsf{2PC}}$ follows the same computation to maintain shares.

Given the extracted secret shares of corrupted P_b, $\mathcal{S}_{\mathsf{2PC}}$ works as follows. In **Input**, $\mathcal{S}_{\mathsf{2PC}}$ extracts all input bits of corrupted P_b from emulated $\mathcal{F}_{\mathsf{aBit}}$ and sends them to $\mathcal{F}_{\mathsf{2PC}}$. In **Open**, $\mathcal{S}_{\mathsf{2PC}}$ receives $x \in \mathbb{F}_{2^\kappa}$ from $\mathcal{F}_{\mathsf{2PC}}$ and sends $x_{1-b} := x \oplus x_b$ to P_b, where x_b is given by the extracted P_b's secret share of SPDZ-style authenticated sharing $[\![x]\!]$. In **Check**, $\mathcal{S}_{\mathsf{2PC}}$ emulates $\mathcal{F}_{\mathsf{coin}}$ to get $\chi \in \mathbb{F}_{2^\kappa}$ and extracts $V_b \in \mathbb{F}_{2^\kappa}$ from emulated $\mathcal{F}_{\mathsf{com}}$. Moreover, $\mathcal{S}_{\mathsf{2PC}}$ constructs a predicate (i.e., a mixed circuit) P such that, on input all values authenticated in BDOZ or SPDZ style in Π_{2PC} (or equivalently, all values stored in $\mathcal{F}_{\mathsf{2PC}}$), it uses these inputs, the extracted inputs and secret shares of P_b, and the public random coins to define $V_{1-b} \in \mathbb{F}_{2^\kappa}$ in an equivalent way and outputs 1 if and only if $V_{1-b} = V_b$.

The two worlds are indistinguishable unless the adversary breaks the obliviousness of garbling schemes or leads to a non-negligible difference in abort probability. Note that the latter difference is negligible given uniform χ. □

Appendix B. Proof of Theorem 2

Theorem 2. *Let $\mathcal{H}_0$ be a CCR hash function. Then, protocol Π_{DPF} (Figure 5) securely realizes functionality $\mathcal{F}_{\mathsf{DPF}}$ (Figure 6) against malicious adversaries in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{com}})$-hybrid model.*

Proof (Sketch). As the two parties are symmetric in Π_{DPF}, we w.l.o.g. assume that P_b is corrupted and P_{1-b} is honest for some fixed $b \in \{0,1\}$. Simulator $\mathcal{S}_{\mathsf{DPF}}$ extracts $\Delta_b \in \mathbb{F}_{2^\kappa}$ from emulated $\mathcal{F}_{\mathsf{aBit}}$ (and aborts if $\mathsf{lsb}(\Delta_b) \neq b$) and maintains all secret shares held by corrupted P_b. These shares can be computed from public randomness W and the extracted Δ_b, each $\langle\!\langle \alpha^{(i)} \rangle\!\rangle_b$, and $[\![\beta]\!]_b$.

$\mathcal{S}_{\mathsf{DPF}}$ sends a uniformly sampled $\mathsf{CW}_{1-b}^{(i)}$ to corrupted P_b for each $i \in [0,n]$. This is indistinguishable from the real-world execution, where $\mathsf{CW}_{1-b}^{(i)} = \mathsf{CW}^{(i)} \oplus \mathsf{CW}_b^{(i)}$, since $\mathsf{CW}^{(i)}$ is pseudorandom given a CCR hash function $\mathcal{H}_0$ and the entropy of Δ_{1-b}. $\mathcal{S}_{\mathsf{DPF}}$ also receives $\mathsf{CW}_b^{(i)\prime}$ from corrupted P_b to extracts additive noise $\delta^{(i)} := \mathsf{CW}_b^{(i)\prime} \oplus \mathsf{CW}_b^{(i)}$ for each $i \in [0,n]$.

To simulate batch check, $\mathcal{S}_{\mathsf{DPF}}$ sends a uniformly random a_{1-b} to corrupted P_b. Then, it constructs a predicate P such that, on input all values authenticated in BDOZ or SPDZ style in Π_{DPF} (or equivalently, all values stored in $\mathcal{F}_{\mathsf{DPF}}$), it uses a prefix $\alpha^{(0)}, \ldots, \alpha^{(\ell-1)}$ for some maximal $\ell \in [0,n]$ such that $\delta^{(\ell)} \neq 0$, the extracted secret shares of corrupted P_b, and the public random coins to (i) remove additive noises on the prefix path per level, and (ii) follow the protocol specification of P_b to compute $V_b' \in \mathbb{F}_{2^\kappa}$. This predicate outputs 1 if and only if $V_b' = V_b$, which is extracted from emulated $\mathcal{F}_{\mathsf{com}}$.

The two worlds are indistinguishable unless the adversary breaks the CCR security of $\mathcal{H}_0$ or leads to a non-negligible difference in abort probability. Note that the latter difference is negligible given uniform χ. □

Appendix C. Proof of Theorem 3

Theorem 3. *Let $\mathcal{H}_0$ be a CCR hash function and $\mathcal{GS}$ be a garbling scheme whose obliviousness can be based on CCR $\mathcal{H}_0$. Then, protocol Π_{RAM2PC} (Figure 8) securely realizes functionality $\mathcal{F}_{\mathsf{RAM2PC}}$ (Figure 7) against malicious adversaries in the ($\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{F}_{\mathsf{coin}}$, $\mathcal{F}_{\mathsf{com}}$)-hybrid model.*

Proof (Sketch). Since the two parties are symmetric in Π_{RAM2PC}, we w.l.o.g. assume that P_b is corrupted and P_{1-b} is honest for some fixed $b \in \{0,1\}$. Note that protocol Π_{RAM2PC} invokes two sub-protocols Π_{2PC} and Π_{DPF} as subroutines. Simulator $\mathcal{S}_{\mathsf{RAM2PC}}$ can invoke simulator $\mathcal{S}_{\mathsf{2PC}}$ and $\mathcal{S}_{\mathsf{DPF}}$ to simulate the transcripts of the two sub-protocols.

A subtle issue is that, since Π_{2PC} and Π_{DPF} share the same global authentication keys Δ_0 and Δ_1, the indistinguishability between these transcripts and truly random values sampled in $\mathcal{S}_{\mathsf{RAM2PC}}$ (as per invoked $\mathcal{S}_{\mathsf{2PC}}$ and $\mathcal{S}_{\mathsf{DPF}}$) cannot be reduced to the obliviousness of garbling schemes or the CCR security of $\mathcal{H}_0$ in a black-box way. However, we note that the obliviousness of garbling schemes can also be based on CCR [55], [43]. So, we can use a CCR-based garbling scheme in a *non-black-box* way and prove the above indistinguishability using CCR.

Except the transcripts sent in the two sub-protocols, the only additional transcript exchanged between the two parties is in **Refresh** procedure. This transcript is indistinguishable from a truly random value, which is sampled by $\mathcal{S}_{\mathsf{RAM2PC}}$, due to the PRF security.

To incur abort in the ideal world with nearly the same probability in the real world, $\mathcal{S}_{\mathsf{RAM2PC}}$ constructs a predicate P in **Check** such that it "stacks" the predicates in $\mathcal{S}_{\mathsf{2PC}}$ and $\mathcal{S}_{\mathsf{DPF}}$ according to how the real execution of Π_{RAM2PC} invokes the commands in Π_{2PC} and Π_{DPF}. The abort probability can have negligible difference in the two worlds due the uniformness of χ. $\square$

Appendix D. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary of Paper

The paper introduces cryptographic constructions for enhancing the efficiency of actively secure two-party computation (2PC) RAM programs, providing security against static malicious adversaries. The authors propose an efficient protocol for distributed point function (DPF) as a key component, employing a dual execution strategy to enhance the security from semi-honest to malicious corruption. Despite incurring a one-bit leakage as a trade-off, empirical evaluations showcase the protocol's effectiveness over existing semi-honest protocols in achieving malicious security while minimizing overhead.

D.2. Scientific Contributions

Provides a Valuable Step Forward in an Established Field

D.3. Reasons for Acceptance

- Proposes an efficient actively secure Distributed Point Function (DPF) construction.
- Introduces a novel approach for malicious security in RAM-based two-party computation (2PC), incorporating a single bit of leakage.
- Achieves up to two-order-of-magnitude improvement in performance over existing malicious RAM-based 2PC.
- Open sourced benchmarking code for the proposed protocol.

D.4. Noteworthy Concerns

- The current version exclusively supports Boolean computations, lacking discussion on extending support for arithmetic computations.
- The paper's current writing style is dense, potentially hindering accessibility for non-experts.
- There's a need for improved clarity in distinguishing existing ideas from the new techniques proposed in the paper.
- The paper lacks a comparison with the related work "Lightweight, Maliciously Secure Verifiable Function Secret Sharing. EUROCRYPT'22," which could provide valuable insights and context.
- The authors implemented a non-optimized version of Gale-Shapley, despite asserting that all required building blocks are present in the paper.