

Maliciously Secure Shuffled Distributed OPRF with Applications to Private Set Operations

Aron van Baarsen

Aarhus University
Aarhus, Denmark
aron.van.baarsen@cs.au.dk

Aarushi Goel

Rutgers University
New Brunswick, New Jersey, USA
aarushi.goel@rutgers.edu

Lisa Kohl

Cryptology Group, CWI Amsterdam
Amsterdam, Netherlands
lisa.kohl@cw.nl

Peihan Miao

Brown University
Providence, Rhode Island, USA
peihan_miao@brown.edu

Phuoc Van Long Pham

Brown University
Providence, Rhode Island, USA
phuoc_pham_van_long@brown.edu

Peter Scholl

Aarhus University
Aarhus, Denmark
peter.scholl@cs.au.dk

Satvinder Singh

Purdue University
West Lafayette, Indiana, USA
sing1745@purdue.edu

Abstract

Private Set Intersection (PSI) enables two parties to compute the intersection of their private sets without revealing any additional information. For plain PSI, there exist extremely efficient solutions based on symmetric-key techniques for both the semi-honest and malicious settings. However, for more enriched set operations such as PSI-Cardinality, PSI-Sum, Circuit-PSI, and Private Set Union, the efficiency gap between the semi-honest and malicious settings is significant, often spanning orders of magnitude. A key bottleneck underlying these protocols is the need for a *shuffled distributed oblivious pseudorandom function (SH-DOPRF)* with malicious security. Existing constructions for this primitive are asymptotically suboptimal and/or concretely inefficient.

In this work, we present a new protocol for SH-DOPRF with malicious security that achieves linear computation and communication complexity. We present two instantiations: a DDH-based construction using the Dodis-Yampolskiy PRF (PKC 2005), and a plausibly post-quantum secure construction based on a variant of the Dark-Matter PRF (Aranha et al., S&P 2026). Our approach is almost entirely based on symmetric-key techniques, with public-key operations used only for evaluating PRF outputs. This primitive yields maliciously secure protocols for *multi-query reverse private membership test* (Zhang et al., USENIX Security 2023), which in turn can be used to realize the aforementioned private set operations. These protocols provide two-sided outputs while only additionally leaking the cardinality of the intersection.

We implement and evaluate our construction to demonstrate concrete efficiency. Our DDH-based maliciously-secure SH-DOPRF achieves a 2–3 orders of magnitude improvement in total running time compared to the state-of-the-art (Miao et al., CRYPTO 2020

& Yang et al., PoPETs 2025), leading to a similar improvement for PSI-Cardinality. The resulting protocols for various other private set operations also achieve significant improvements over prior work; for example, our PSI-Sum and Circuit-PSI protocols are 1–2 orders of magnitude more efficient.

CCS Concepts

• Security and privacy → Cryptography.

Keywords

Private Set Operations, Shuffled Distributed OPRF, Malicious Security, Post-Quantum Security

ACM Reference Format:

Aron van Baarsen, Aarushi Goel, Lisa Kohl, Peihan Miao, Phuoc Van Long Pham, Peter Scholl, and Satvinder Singh. 2026. Maliciously Secure Shuffled Distributed OPRF with Applications to Private Set Operations. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 25 pages. <https://doi.org/10.1145/3830454.3846787>

1 Introduction

Private Set Intersection (PSI) allows two mutually distrusting parties, each holding a private input set S_x and S_y respectively, to compute an intersection $S_x \cap S_y$ of these sets without revealing any additional information. PSI has found applications in many real-world scenarios, including password breach detection [67], online advertising measurement [45], private contact discovery [32], and pattern matching [68]. A long line of work [25, 36, 49, 57, 59, 62, 65] has made significant progress towards realizing PSI efficiently. State-of-the-art PSI protocols only rely on lightweight cryptographic tools such as vector oblivious linear evaluation (VOLE) [16, 17] and symmetric-key operations, resulting in extremely fast implementations in both the semi-honest and malicious security settings.

These works primarily focus on computing the *plain* PSI functionality, where both parties learn the entire intersection $S_x \cap S_y$. However, many practical applications require computing more



This work is licensed under a Creative Commons Attribution 4.0 International License. CCS '26, The Hague, Netherlands

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2871-6/2026/11

<https://doi.org/10.1145/3830454.3846787>

advanced data analytics on the intersection [20, 45, 52], namely $f(S_x \cap S_y)$ for some function f . For instance, *PSI-Cardinality* computes the size of the intersection; *PSI-Sum* computes the sum of payload data associated with the elements in the intersection. More generally, *Circuit-PSI* enables parties to learn secret shares of all the elements in the intersection, enabling them to perform arbitrary two-party computation over the result. In addition, some applications require computing the union of the parties' sets, which can be achieved via *Private Set Union (PSU)*.

A substantial body of work has developed protocols for PSI-Cardinality [30, 36, 37, 48], PSI-Sum [26, 45, 46, 54], Circuit-PSI [24, 44, 60, 65] and PSU [23, 26, 48, 50, 61, 75]. However, unlike plain PSI – where state-of-the-art protocols achieve comparable efficiency in both the semi-honest and malicious settings – in these works on more advanced set operations, achieving malicious security comes at a significantly higher cost, resulting in a large performance gap relative to their semi-honest counterparts.

Semi-Honest vs Malicious Security. State-of-the-art semi-honest protocols for Circuit-PSI rely on techniques such as Cuckoo hashing [37] and Oblivious Key Value Stores (OKVS) [13, 62, 65]. While, in principle, one could attempt to upgrade these protocols to the malicious setting by incorporating zero-knowledge proofs, such approaches incur prohibitive overheads in practice. State-of-the-art constructions for maliciously secure Circuit-PSI [73] instead rely on a primitive known as a *Shuffled Distributed Oblivious Pseudorandom Function (SH-DOPRF)*.¹ Unfortunately, existing constructions of SH-DOPRF are asymptotically suboptimal [73] (i.e., quasi-linear in the length of the input) and/or concretely inefficient [54]. For example, prior work [73] on maliciously secure circuit-PSI reports a runtime of ≈ 110 hours for input sets of size one million.

Another contributing factor to this disparity between efficiency in the semi-honest and malicious settings is that many advanced set operations require *two-sided output*, i.e., both parties learn the result. While upgrading a semi-honest protocol with one-sided output to two-sided output is straightforward in the semi-honest setting (by having the output-holding party simply send the result to the other party), this approach is no longer secure against malicious adversaries [54, 64]. Therefore, achieving two-sided output in the malicious setting incurs significant additional overhead. For instance, the best known maliciously secure protocol [54] for PSI-Sum with two-sided output requires approximately 19 hours for sets of around one million elements.

This raises the following question:

Can we construct concretely efficient protocols for advanced private set operations with two-sided outputs in the malicious setting?

1.1 Our Contributions

We answer the above question in the affirmative. Specifically, our contributions are as follows:

- (1) **New Constructions of SH-DOPRF:** We present two new constructions of maliciously secure SH-DOPRF. Our first construction makes use of the Dodis-Yampolskiy PRF [34] which is based on the ℓ -decisional Diffie-Hellman inversion (ℓ -DDHI) assumption. Our second construction relies on the recent Dark-Matter

PRF [4, 15] and yields the *first plausibly post-quantum secure* instantiation of SH-DOPRF. Both our constructions achieve linear computational and communication complexity² and are concretely efficient, relying almost entirely on symmetric-key techniques, with public-key operations used only for PRF evaluation.

- (2) **PSO with Two-Sided Output:** We then use SH-DOPRF to construct maliciously secure protocols for various private set operations (PSO), including PSI-Cardinality, PSI-Sum, Circuit PSI and PSU. All our protocols achieve linear (in the set sizes) computation and communication complexity. We note that for circuit-PSI all prior maliciously secure protocols incur at least quasilinear computation and communication. For other functionalities, while existing protocols achieve similar asymptotic complexity as us, they rely heavily on expensive public-key operations, making them impractical.
- (3) **Implementation and Evaluation:** We implement our DDH-based instantiation of SH-DOPRF and the resulting PSO protocols to evaluate their concrete efficiency. Our experiments demonstrate significant improvements over state-of-the-art protocols for these functionalities. In particular, our protocols for SH-DOPRF and two-sided PSI-Cardinality achieve $100 - 613\times$ improvements over prior work. Similarly, compared to the state-of-the-art, our Circuit-PSI protocol is $13 - 451\times$ faster, our PSI-Sum protocol achieves a $19 - 80\times$ speedup, and our PSU protocol is $1.06 - 1.9\times$ faster in the online phase.

1.2 Technical Overview

We now discuss the main ideas underlying our results.

Private Set Intersection (with one-sided output). State-of-the-art PSI protocols [62, 65] implicitly rely on a primitive called oblivious pseudorandom function (OPRF). In an OPRF, one party $\mathcal{P}_1$ holds a PRF key k , while the other party $\mathcal{P}_0$ holds an input x and learns the PRF evaluations $F_k(x)$, without learning any additional information about k ; meanwhile, $\mathcal{P}_1$ learns nothing about the input x . Using this primitive, constructing a PSI protocol (with one-sided output) is straightforward: $\mathcal{P}_0$ obtains PRF evaluations $F_k(x_i)$ for each element x_i in her set S_x through OPRF. Since $\mathcal{P}_1$ knows the PRF key k , he can compute $F_k(y_i)$ for all elements in his set S_y and send these to $\mathcal{P}_0$, which enables $\mathcal{P}_0$ to compare the PRF values and recover the intersection $S_x \cap S_y$.

From Plain PSI to Advanced Private Set Operations. Achieving malicious security for other advanced set operations, even as simple as PSI-Cardinality (with one-sided output), is significantly harder than plain PSI. The main challenge is that $\mathcal{P}_0$ must not learn the correspondence between her elements and their PRF evaluations; otherwise she could identify which elements are in the intersection. That is, the PRF evaluations $\{F_k(x_i)\}$ must be randomly *shuffled* before being revealed to $\mathcal{P}_0$, rather than revealing them one-by-one as in the PSI protocol.

Realizing this shuffling step with malicious security is the main bottleneck in prior work. Miao et al. [54] achieve this using linearly homomorphic encryption and zero-knowledge proofs, both

¹We explain this primitive in more detail in Section 1.2.

²The round count of our (and related works') protocols is a small constant. As such, the network latency is insignificant compared to the overall cost, and explicit mention of the round complexity is omitted in the remainder of this work.

requiring expensive public-key operations. More recently, Yang et al. [73] realize this using SPDZ-based MPC techniques, which reduces computational cost but relies on permutation networks such as Beneš [12], which incurs either quadratic offline complexity or quasilinear overall complexity.

Furthermore, many applications require *two-sided output* where both parties learn the result. However, the OPRF-based approach does not naturally extend to this setting, particularly in the presence of malicious adversaries. Finally, none of the existing constructions achieve post-quantum security. We discuss other related work on maliciously secure private set operations in Appendix C.

Shuffled Distributed OPRF. This combination of OPRF and the shuffling step is abstracted as a primitive called *Shuffled Distributed OPRF (SH-DOPRF)* in prior works [54, 73]. Formally, in SH-DOPRF, the two parties jointly hold secret shares of a PRF key k (unlike the plain PSI setting above, where one party holds the key). One party $\mathcal{P}_0$ holds a vector $\vec{x}$ of size n and the other party $\mathcal{P}_1$ holds a permutation $\pi_1 : [n] \rightarrow [n]$. At the end of the protocol, $\mathcal{P}_1$ learns all the PRF evaluations $\{F_k(x_i)\}_{i \in [n]}$ while $\mathcal{P}_0$ learns the permuted evaluations $\{F_k(x_{\pi_1(i)})\}_{i \in [n]}$. Since neither party knows k in the clear, revealing the OPRF values to both parties does not compromise security, which also helps achieve two-sided output.

We present two new instantiations of maliciously secure SH-DOPRFs: one based on the Dodis-Yampolskiy PRF [34] and one based on a recent plausibly post-quantum secure Dark-Matter PRF [4, 15]. Both our constructions achieve $O(n)$ computation and communication complexity and are concretely efficient.

Before describing our SH-DOPRF constructions, we briefly outline how to obtain maliciously secure private set operations (PSO) protocols with two-sided output from SH-DOPRF. We first use our SH-DOPRF to achieve a two-sided variant of the *multi-query reverse private membership test* (mqRPMT) functionality [75], and then build various PSO protocols based on this functionality. mqRPMT was previously used to build PSO protocols [26], but their construction was only secure against semi-honest adversaries, and no maliciously secure construction has been proposed so far.

mqRPMT from SH-DOPRF. In mqRPMT, $\mathcal{P}_0$ holds a set S_x (with corresponding vector $\vec{x}$) and $\mathcal{P}_1$ holds a set S_y (with corresponding vector $\vec{y}$); at the end of the protocol, $\mathcal{P}_1$ receives as output an indicator vector $b_1, \dots, b_n$, where $b_i = 1$ iff $x_i \in \vec{y}$. In the two-sided variant, $\mathcal{P}_0$ additionally receives a vector $b'_1, \dots, b'_n$, where $b'_i = 1$ iff $y_i \in \vec{x}$.

Given an SH-DOPRF protocol, it is straightforward to obtain mqRPMT, by simply running SH-DOPRF in both directions with the same key. In one execution, $\mathcal{P}_1$ learns $\{F_k(x_i)\}_{i \in [n]}$ and $\mathcal{P}_0$ learns the permuted evaluations $\{F_k(x_{\pi_1(i)})\}_{i \in [n]}$, where π_1 is known to $\mathcal{P}_1$. In the other execution, $\mathcal{P}_0$ learns $\{F_k(y_i)\}_{i \in [n]}$ and $\mathcal{P}_1$ learns the permuted evaluations $\{F_k(y_{\pi_0(i)})\}_{i \in [n]}$, where π_0 is known to $\mathcal{P}_0$. Each party can then compare these PRF evaluations to recover their respective indicator vectors.

Private Set Operations From mqRPMT. The mqRPMT functionality immediately reveals the *cardinality* of the intersection to both parties, and all our PSO protocols therefore inherit this leakage. Beyond PSI-Cardinality, mqRPMT can be used to obtain PSI-Sum and (more generally) Circuit-PSI using multiplication triples [10]. Specifically, computing secret-shares of the product between each

indicator b_i and the corresponding payload results in secret shares of the payload values in the intersection; adding these shares up gives PSI-Sum. Similarly, multiplying b_i with the corresponding x_i leads to the parties obtaining secret shares of the intersection $S_x \cap S_y$, achieving Circuit-PSI. A protocol for Private Set Union (PSU) follows by multiplying $(1 - b_i)$ with x_i and revealing the results [26]. In all cases, a *two-sided* mqRPMT protocol along with n multiplication triples suffices to achieve protocols for these set operations with two-sided output. For ensuring malicious security, we additionally require the output of mqRPMT and the multiplication triples to be secret shared in an authenticated manner.

Our SH-DOPRF Constructions. We now explain our constructions for SH-DOPRF. Let $\mathcal{P}_0$ hold a PRF key share k_0 and a vector $\vec{x}$, and $\mathcal{P}_1$ hold another PRF key share k_1 and a permutation $\pi_1 : [n] \rightarrow [n]$. Our constructions leverages lightweight VOLE-based commitments and VOLE-based zero-knowledge proofs [8, 16, 71, 72].³

At a high level, our constructions proceed in two steps. Parties first commit to their respective inputs, $(k_0, \vec{x})$ and (k_1, π_1) , via VOLE-based commitments (which are linearly homomorphic). Through authenticated arithmetic operations on these committed values, $\mathcal{P}_1$ obtains VOLE-based-authenticated OPRF values $\{F_k(x_i)\}_{i \in [n]}$. Second, $\mathcal{P}_1$ applies the permutation π_1 on these values and proves correctness of the shuffling via a VOLE-based zero-knowledge proof. Since the OPRF values are already VOLE-authenticated, this proof is highly efficient. Looking ahead, we can further authenticate the output indicator vectors in the mqRPMT protocol, which are subsequently required in our protocols for other private set operations.

Instantiation 1: Based on Dodis-Yampolskiy PRF. The Dodis-Yampolskiy PRF [34] is defined as $F_k(x) = g^{\frac{1}{k+x}}$. We will assume that the key $k = k_0 + k_1$ is secret shared using an authenticated secret sharing scheme (i.e., $\mathcal{P}_i$ receives a commitment to $k_{i \oplus 1}$). $\mathcal{P}_0$ starts by committing to her set elements $\{x_i\}_{i \in [n]}$ and $\mathcal{P}_1$ commits to the permutation π_1 . We now describe how each party obtains the final permuted PRF evaluations.

• **How $\mathcal{P}_1$ gets the PRF evaluations:** $\mathcal{P}_0$ randomly samples $\{r_i\}_{i \in [n]}$ and sends commitments of $\{r_i\}_{i \in [n]}$ along with $\{G^{r_i}\}_{i \in [n]}$ to $\mathcal{P}_1$. For each $i \in [n]$, the parties now need to jointly compute $\tilde{a}_i := (k + x_i) \cdot r_i$ and reconstruct this value to $\mathcal{P}_1$, who can then locally compute $F_k(x_i) = (G^{r_i})^{\frac{1}{\tilde{a}_i}}$. A straightforward approach for enabling this joint computation of $\tilde{a}_i$ makes use of (authenticated) multiplication triples. However, known approaches for generating such multiplication triples, especially in the malicious setting are computationally expensive. We therefore want to avoid this approach.

To this end, we observe that $(k + x_i) \cdot r_i := (k_0 + x_i) \cdot r_i + k_1 \cdot r_i$, where the first term, $(k_0 + x_i) \cdot r_i$, is entirely known to $\mathcal{P}_0$. Thus, $\mathcal{P}_0$ can send a commitment to this term to $\mathcal{P}_1$, along with a proof of consistency with respect to the previously sent commitments to k_0, x_i , and r_i . Next, given a linearly homomorphic commitment to r_i , $\mathcal{P}_1$ can efficiently compute a commitment to the second

³These are commit-and-prove style proof systems that can be used to generate proofs over values committed using VOLE-based commitments.

term, $k_1 \cdot r_i$. The parties can then combine these commitments to obtain a commitment to $(k + x_i) \cdot r_i$.

In Section 3.1, we show that by leveraging VOLE-based commitments, VOLE-based authenticated secret sharing, and VOLE-based zero-knowledge proofs, the parties can also jointly compute a valid opening of this final commitment. Our techniques make use of an interesting observation that $k_1 \cdot r_i$ can be viewed as a VOLE correlation with k_1 as the VOLE global key and r_i as the payload. VOLE-ZK can be further leveraged to ensure that the r_i used in the first term is consistent with the one in the second terms. Overall, this entire computation only relies on VOLE-assisted operations and is therefore highly efficient.

- **How $\mathcal{P}_0$ gets the permuted PRF evaluations:** Next, $\mathcal{P}_1$ commits to $a'_i := \frac{1}{a_i}$ and sends this commitment to $\mathcal{P}_0$, along with a proof that a'_i is the multiplicative inverse of a_i . In addition, $\mathcal{P}_1$ sends the permuted PRF evaluations. Using the linear homomorphism of the underlying commitment scheme, the parties can combine G^{r_i} with the commitment to a_i to obtain a commitment (in the exponent) to the *unpermuted* PRF evaluations, i.e., in the original order. Finally, $\mathcal{P}_1$ must convince $\mathcal{P}_0$ that these values are consistent with the committed permutation π_1 and the permuted PRF evaluations it previously sent.

For this permutation proof, we build on the well-known approach of Neff [55], which enables a prover to show that two committed vectors are permutations of one another. However, this technique cannot be applied directly in our setting for two reasons: first, it only establishes that the two vectors are related by *some* permutation, whereas we require a proof with respect to a *specific*, committed permutation; second, our setting involves commitments in the exponent, which differs from the standard commitment model considered in [55].

A challenge similar to our second issue is addressed by Bayer and Groth [9], using Pedersen commitments [56], but their approach relies heavily on public-key operations. At a conceptual level, we replace Pedersen commitments and the Neff-style shuffling argument in [9] with VOLE commitments and VOLE-ZK respectively. The remaining step in [9] involves a multi-exponentiation argument, which we adapt by lifting VOLE-ZK to the exponent which we formalize as VOLE-ZK over groups in Section 2.6.1. More details about the adaptation are presented in Section 3.1. Overall, using a local proof of shuffling instead of oblivious shuffling reduces the asymptotic cost from quasilinear to linear, and VOLE-based commitments and proofs yield much better concrete efficiency.

Finally, note that the mqRPMT protocol additionally requires the output indicator vectors to be VOLE-authenticated for subsequent set operations. We achieve this by applying a similar VOLE-based shuffling proof with respect to the same committed permutation.

Instantiation 2: Based on Dark-Matter PRF. Our second instantiation is based on a recent maliciously-secure OPRF protocol by Aranha et al. [4]. The PRF underlying their construction can be seen as a variant of the Dark-Matter PRF paradigm [15], and is defined as $F_k(x) := B \cdot_3 \text{vec}(k \cdot y)$, where $B \in \mathbb{F}_3^{t \times r}$ is a public matrix, the key $k \in \mathbb{F}_{2^r}$, and $y \in \mathbb{F}_{2^r}$ is an encoded version $y = \text{Enc}(x)$ of the input $x \in \mathbb{F}_2^d$ according to some public encoding function. Here we furthermore use $\cdot_3$ to indicate multiplication modulo 3

and $\text{vec} : \mathbb{F}_{2^r} \rightarrow \mathbb{F}_2^r$ to indicate a canonical mapping from a field element to its vector representation.

Following the general approach described above, to obtain a SH-DOPRF, we build upon their OPRF protocol to instantiate the same blue-print as underlying the Dodis-Yampolskiy PRF instantiation:

- (1) For each input $x_i \in \mathbb{F}_2^d$, commit to x_i by using a VOLE commitment, i.e., one party $\mathcal{P}_0$ holds $x_i, w_i \in \mathbb{F}_2^d$ and party $\mathcal{P}_1$ holds Δ_1 and $v_i \in \mathbb{F}_2^d$ such that $x_i \cdot \Delta_1 = w_i + v_i$ (in the following denoted as $\llbracket x_i \rrbracket_0$);
- (2) Authenticated secret-shared key $\langle k \rangle \leftarrow (\llbracket k_0 \rrbracket_0, \llbracket k_1 \rrbracket_1)$, i.e., key shares k_0 and k_1 are authenticated under Δ_1 and Δ_0 , resp.;
- (3) Compute authenticated outputs $\langle F_k(x_i) \rangle$ and output $\llbracket F_k(x_i) \rrbracket_1$ to $\mathcal{P}_1$ (i.e., $\mathcal{P}_1$ obtains $F_k(x_i)$ in the clear which is authenticated under Δ_0);
- (4) $\mathcal{P}_1$ outputs $\{(F_k(x_{\pi_1(i)}))\}_{i \in n}$ to $\mathcal{P}_0$ and proves that this corresponds to a correct shuffling of $\llbracket F_k(x_i) \rrbracket_1$ under some permutation π_1 (chosen by $\mathcal{P}_1$) using VOLE-based zero-knowledge.

Since the protocol from [4] already makes extensive use of VOLE-based techniques, we can leverage their building blocks to realize the above VOLE-based commitments. To allow (1): $\mathcal{P}_0$ proves using a VOLE-based zero-knowledge proof that they correctly evaluated the input encoding on their committed input as $\llbracket y_i \rrbracket_0 = \text{Enc}(\llbracket X[i] \rrbracket_0)$.

Now to realize (2): $\mathcal{P}_0$ can prove in VOLE-ZK that they correctly multiplied the encoded input with their own key share by showing that $\llbracket k_0 \cdot y_i \rrbracket_0 = \llbracket k_0 \rrbracket_0 \cdot \llbracket y_i \rrbracket_0$, and the parties can use an authenticated VOLE protocol to compute authenticated secret shares $\llbracket v_i \rrbracket_0$ and $\llbracket w_i \rrbracket_1$ such that $v_i + w_i = k_1 \cdot y_i$. Hence, the parties obtain authenticated secret shares $\langle k \cdot y_i \rangle \leftarrow (\llbracket v_i \rrbracket_0 + \llbracket k_0 \cdot y_i \rrbracket_0, \llbracket w_i \rrbracket_1)$.

Finally, to get (3): the parties can convert the authenticated secret shares $\langle k \cdot y_i \rangle \bmod 2$ to authenticated secret shares $\langle \text{vec}(k \cdot y_i) \rangle \bmod 3$ using doubly-authenticated bits and perform the final multiplication with matrix B locally on their shares, similarly to [4], obtaining $\langle F_k(x_i) \rangle = \langle B \cdot_3 \text{vec}(k \cdot y_i) \rangle$. Now, $\mathcal{P}_0$ opens their share to party $\mathcal{P}_1$, for $\mathcal{P}_1$ to obtain $\llbracket F_k(x_i) \rrbracket_1$.

Towards (4): party $\mathcal{P}_1$ commits to a permutation π_1 of their choice, opens $\{F_k(x_{\pi_1(i)})\}_{i \in [n]}$ to $\mathcal{P}_0$ and proves using VOLE-ZK that this indeed corresponds to a valid permutation of $\{\llbracket F_k(x_i) \rrbracket_1\}_{i \in [n]}$, as in [55]. Committing to the permutation will further be useful to obtain the mqRPMT protocol, where party $\mathcal{P}_1$ has to permute the output vector $\vec{b}$ (indicating the set intersection) according to the same permutation π_1 .

For a batch of n inputs, the resulting SH-DOPRF protocol only incurs $\approx 5\lambda n$ bits of additional (amortized) communication in the online phase, and minimal computation overhead compared to the OPRF protocol from [4].

2 Preliminaries

In this section, we recall some preliminary definitions.

2.1 Notation

We start by establishing some notations.

Security Parameters. λ is used to denote the computational security parameter and σ the statistical security parameter. We assume that all algorithms/functions implicitly take the security parameters as input, but generally omit it for brevity.

Sets and Vectors. $x \xleftarrow{\$} S$ denotes that x is sampled uniformly at random from the set S . We use arrow notation like $\vec{a}$ to denote vectors. We represent the i^{th} element of a vector $\vec{v}$ as v_i . We denote $\vec{v} \odot \vec{w} := (v_1 \cdot w_1, \dots, v_n \cdot w_n)$ for the pointwise product, and $\vec{v} + \vec{w}$ for the pointwise sum of two vectors $\vec{v}, \vec{w} \in \mathbb{F}^n$. We also let $\odot$ (or $+$) denote the scalar-vector product (or sum). When $\vec{v}$ is a vector of vectors, We use $\vec{v}[i : j]$ to denote *pointwise slicing* $(v_{1,i:j}, \dots, v_{n,i:j})$, when $\vec{v}$ is a vector that contains n small vectors. For $n \in \mathbb{N}$, let $[n]$ represent the set $\{1, 2, \dots, n\}$.

Group Notation. We use $\mathbb{G}$ to denote the group we work over. For a cyclic group $\mathbb{G}$ of prime order q , we write $G^{\vec{x}}$ for the vector of component-wise exponentiations $G^{\vec{x}} := (G^{x_1}, \dots, G^{x_n})$ for $G \in \mathbb{G}$ and $\vec{x} \in \mathbb{F}_q^n$. More general, for a vector of base elements $\vec{G} \in \mathbb{G}^n$, we write $\vec{G}^{\vec{x}} := (G_1^{x_1}, \dots, G_n^{x_n})$. Similarly, we use $\odot$ for the componentwise product of vectors of group elements $\vec{G} \odot \vec{H} := (G_1 \cdot H_1, \dots, G_n \cdot H_n)$, and extend group inversion componentwise as $\vec{G}^{-1} := (G_1^{-1}, \dots, G_n^{-1})$.

Field Notation. We use $\mathbb{F}$ to denote the field we work over. For a finite field $\mathbb{F}_q$ of prime order q and extension field $\mathbb{F}_{q^r}$, we denote $\text{vec} : \mathbb{F}_{q^r} \rightarrow \mathbb{F}_q^r$ for the map converting a field element $z \in \mathbb{F}_{q^r}$ to its vector representation $\text{vec}(z) \in \mathbb{F}_q^r$ under a (fixed) choice of basis, and denote $\text{elt} : \mathbb{F}_q^r \rightarrow \mathbb{F}_{q^r}$ for the converse map. For a prime q and extension field parameter r , we define the *degree* of an arithmetic circuit $C : \mathbb{F}_q^\ell \rightarrow \mathbb{F}_q$ to be the maximal degree of the univariate polynomial $C(a_1X + b_1, \dots, a_nX + b_n)$, where $a_i, b_i \in \mathbb{F}_{q^r}$.

2.2 Secure Multi-Party Computation

Looking ahead, we see that it is easy to define our protocols such as Circuit-PSI and PSI-Cardinality as ideal functionalities and then argue security. Secure multi-party computation (MPC) [39, 74] allows mutually distrusting parties to compute a function on their joint private inputs without revealing anything but the output. We follow standard MPC literature and consider the real-ideal paradigm and provide the full definition of security in Appendix D.

2.3 Vector Oblivious Linear Evaluation

Vector Oblivious Linear Evaluation (VOLE) [16, 17] is an input-less secure two-party functionality that allows one party to obtain random values $\Delta \in \mathbb{F}_{q^r}$ and $\vec{v} \in \mathbb{F}_{q^r}^\ell$ and the other party to obtain values $\vec{u}, \vec{w} \in \mathbb{F}_{q^r}^\ell$ such that $\vec{v} = \vec{w} + \Delta \cdot \vec{u}$. The global key Δ is sampled once and multiple such VOLE correlations can be obtained with respect to the same Δ . We formally define the ideal functionality for VOLE in Figure 1.⁴

2.4 VOLE-based Commitments

In this work, we make use of VOLE-based commitments [8, 16, 71, 72], a common method of committing to vectors of values extremely efficiently. The ideal VOLE functionality can already be seen as a commitment to the random vector $\vec{u}$ held by $P_{i \oplus 1}$, where $P_{i \oplus 1}$ holds $(\vec{u}, \vec{w})$ and P_i holds $(\Delta, \vec{v})$. To open the commitment, $P_{i \oplus 1}$ sends $(\vec{u}, \vec{w})$ to P_i who can verify whether $\vec{v} = \vec{w} + \Delta \cdot \vec{u}$. The VOLE correlation can then be de-randomized to allow commitments to arbitrary values. We first briefly discuss this construction.

⁴In this subfield variant, each entry of $\vec{u}$ is constrained to lie in the subfield $\mathbb{F}_q \subset \mathbb{F}_{q^r}$.

Ideal subfield VOLE functionality $\mathcal{F}_{\text{subfield VOLE}}^{q,r}$

Parameters: Two parties, $\mathcal{P}_0$ and $\mathcal{P}_1$. Base field characteristic q , extension field parameter r (defining $\mathbb{F}_q$ and $\mathbb{F}_{q^r}$).

Initialize: Upon receiving $(\text{sid}, \text{init}, i)$ from $\mathcal{P}_0$ and $\mathcal{P}_1$ with $i \in \{0, 1\}$, the functionality initializes an empty dictionary $D = \emptyset$, and samples $\Delta \xleftarrow{\$} \mathbb{F}_{q^r}$ if $\mathcal{P}_i$ is honest. Otherwise, receive $\Delta \in \mathbb{F}_{q^r}$ from $\mathcal{A}$. The functionality returns Δ to $\mathcal{P}_i$ and ignores all future init calls for the same (sid, i) .

Evaluate: Upon receiving $(\text{sid}, \text{eval}, i, \ell, \text{id}_1, \dots, \text{id}_\ell)$ from $\mathcal{P}_0$ and $\mathcal{P}_1$, with $\ell \in \mathbb{N}$ and $\{\text{id}_j\}_j$ distinct identifiers:

- If $\mathcal{P}_{i \oplus 1}$ is honest, sample $\vec{u} \xleftarrow{\$} \mathbb{F}_q^\ell$. Otherwise, receive $\vec{u} \in \mathbb{F}_q^\ell$ from $\mathcal{A}$.
- If $\mathcal{P}_i$ is honest, sample $\vec{v} \xleftarrow{\$} \mathbb{F}_{q^r}^\ell$. Otherwise, receive $\vec{v} \in \mathbb{F}_{q^r}^\ell$ from $\mathcal{A}$.
- If $\mathcal{P}_{i \oplus 1}$ is honest, compute $\vec{w} := \vec{v} - \Delta \cdot \vec{u}$. Otherwise, receive $\vec{w} \in \mathbb{F}_{q^r}^\ell$ from $\mathcal{A}$ and replace $\vec{v} \leftarrow \vec{w} + \Delta \cdot \vec{u}$.
- Return $[\vec{u}]_{i \oplus 1}^{(q)}$ to the parties, that is, output $(\vec{u}, \vec{w})$ to $\mathcal{P}_{i \oplus 1}$ and $(\Delta, \vec{v})$ to $\mathcal{P}_i$, and store $D[\text{id}_j] = \vec{u}[j]$, for $j = 1, \dots, \ell$

Figure 1: Ideal functionality for subfield VOLE.

Construction. Suppose $\mathcal{P}_0$ wants to commit a vector $\vec{x}$ to $\mathcal{P}_1$.

- **Setup:** Both parties invoke $\mathcal{F}_{\text{subfield VOLE}}^{q,r}$ by sending $(\text{sid}, \text{init}, 1)$.
- **Commit Protocol:** Both parties generate a VOLE correlation by invoking evaluate of $\mathcal{F}_{\text{subfield VOLE}}^{q,r}$ so that $\mathcal{P}_0$ obtains $\vec{u}, \vec{w}$ and $\mathcal{P}_1$ obtains $\vec{v}$ satisfying $\vec{v} = \vec{w} + \Delta_1 \cdot \vec{u}$. $\mathcal{P}_0$ then sends $\vec{z} = \vec{x} - \vec{u}$ to $\mathcal{P}_1$. $\mathcal{P}_0$ then outputs $(\vec{x}, \vec{w})$ and $\mathcal{P}_1$ outputs $(\Delta_1, \vec{v}')$ where $\vec{v}' = \vec{v} + \Delta_1 \cdot \vec{z}$.
- **Open Protocol:** $\mathcal{P}_0$ can send $(\vec{x}, \vec{w})$ to $\mathcal{P}_1$ who can verify whether $\vec{v}' = \vec{w} + \Delta_1 \cdot \vec{x}$ holds.

Notation. We will use the notation $[\vec{x}]_i^{(q)}$ to represent a commitment made by party $\mathcal{P}_i$ to the value $\vec{x}$, where q is a parameter to the VOLE functionality indicating the field $\mathbb{F}_q$ over which $\vec{x}$ is defined. We keep the parameter r , defining the extension field $\mathbb{F}_{q^r}$ from which the global key $\Delta_{i \oplus 1}$ is sampled for soundness reasons, implicit in this notation. Note that this is a joint commitment where both parties hold “shares” of the commitment. As such, we abuse notation when we say parties send/receive $[\vec{x}]_i^{(q)}$ to/from the ideal functionality to represent parties sending/receiving their corresponding “shares”. For example, in the commitment scheme discussed above, the shares of the committer are $(\vec{x}, \vec{w})$ and the shares of the receiver are $(\Delta_1, \vec{v}')$.

Furthermore, we abbreviate the Commit protocol with a macro $\Pi_{\text{Commit}}((\vec{x}, q, r), (\Delta_{i \oplus 1}, q, r))$ where the values in the first parenthesis represent $\mathcal{P}_i$'s (i.e., the committer's) input and the values in the second parenthesis represent $\mathcal{P}_{i \oplus 1}$'s input (public parameters occur in both parenthesis). Combined, a party $\mathcal{P}_i$ committing to a vector $\vec{x}$ is represented as

$$[\vec{x}]_i^{(q)} \leftarrow \Pi_{\text{Commit}}((\vec{x}, q, r), (\Delta_{i \oplus 1}, q, r)).$$

The session identifier (sid) is generally left implicit for readability.

Properties. It is easy to see that the commitment scheme satisfies standard definitions of hiding and binding.⁵ Furthermore, VOLE-based commitments are linearly homomorphic. Observe that this VOLE-based commitment can be seen as the committer holding a degree 1 polynomial of the form $P(X) := u \cdot X + w$ and the receiver obtaining an evaluation of this polynomial $P(X)$ at Δ . This idea can naturally be generalized to degree d polynomials (and vectors) giving rise to degree- d VOLE-commitments. These further support bounded multiplicative homomorphism [1, 7, 72].

For completeness, we give the definition of degree- d VOLE-commitments and their homomorphic properties in Appendix F. Note that the commitment scheme discussed in this section is a special case (degree $d = 1$). Furthermore, all newly generated commitments are degree $d = 1$ commitments.

2.5 Authenticated VOLE Correlation

In this work, we further need the extended functionality $\mathcal{F}_{\text{aVOLE}}^{q,r}$ of authenticated VOLE as defined in Figure 2, where the VOLE correlation vectors themselves are committed. More precisely, $\mathcal{P}_0$ gets $(\vec{y}, \vec{w})$ and $\mathcal{P}_1$ gets $(k, \vec{v})$ satisfying $\vec{v} + \vec{w} = k \cdot \vec{y}$ as the VOLE correlation. Note that this functionality differs slightly from the one defined in [4] in the sense that we chose the functionality to additionally take authenticated inputs $[\vec{y}]_0^{(q')}$, whereas [4] takes an unauthenticated $\vec{y}$. We provide a protocol realizing this functionality in Figure 10 (Appendix H.1), which notably has lower communication and computation cost than the protocol in [4].

2.6 VOLE-based Zero-Knowledge Proofs

VOLE-based zero-knowledge proofs [8, 71, 72] leverage homomorphic properties of VOLE-based commitments to enable a prover to prove in zero-knowledge that a set of committed values satisfy some given relation (modeled as a circuit). Such proof systems securely realize the ideal functionality $\mathcal{F}_{\text{VOLE-ZK}}^{q,r}$ described in Figure 3.

This ideal functionality can be viewed as an extension of the previously defined ideal functionalities for VOLE and authenticated VOLE. In our description in Figure 3, party $\mathcal{P}_{i \oplus 1}$ is the prover and $\mathcal{P}_i$ is the verifier. This ideal functionality receives values $(Z_1, \dots, Z_l)$ from the parties. Each value Z_j can be parsed as $\text{id}_j \| z_j$. Recall that a VOLE commitment consisted of 2 steps: generating a VOLE correlation and sending a masked value. id_j represents the id associated with the VOLE correlation stored by the dictionary (refer Figure 1) and z_j is the masked value known to both parties. Computing $D[\text{id}_j] + z_j$ gives the committed value. These are then fed as input into a collection of circuits (which model the relation that the prover wants to prove). The functionality checks whether this circuit is satisfied over the committed values and returns success/abort accordingly. The security of the VOLE-ZK functionality holds as long as the global key of the other party is unknown, in which case an adversary has negligible probability of guessing it. Since the adversary might obtain more information about the key from the environment, we need to model this guess explicitly via the Global-Key Query command in the ideal functionality.

⁵We do not formally discuss why this construction satisfies hiding and binding. This is because, looking ahead, we do not use these properties in a black-box manner in our security proofs and instead reprove them depending upon the use-case.

Ideal authenticated VOLE functionality $\mathcal{F}_{\text{aVOLE}}^{q,r}$

The ideal functionality builds over $\mathcal{F}_{\text{sVOLE}}^{q,r}$ (Fig. 1) with the same parameters and adopts the following new commands:

Initialize: Upon receiving $(\text{sid}_{\text{VOLE}}, \text{sid}_{\text{MAC}}, \text{init})$ from $\mathcal{P}_0$ and $\mathcal{P}_1$, the functionality returns $\Delta_0, \Delta_1 \xleftarrow{\$} \mathbb{F}_{q^r}$ associated with sid_{MAC} to $\mathcal{P}_0$ and $\mathcal{P}_1$, respectively, returns $k \xleftarrow{\$} \mathbb{F}_{q^r}$ associated with sid_{VOLE} to $\mathcal{P}_1$, and outputs $[[k]]_1^{(q')}$.

Evaluate: Upon receiving $(\text{sid}_{\text{VOLE}}, \text{sid}_{\text{MAC}}, [\vec{y}]_0^{(q')})$ from $\mathcal{P}_0$ and $\mathcal{P}_1$, with $\vec{y} \in \mathbb{F}_{q^r}^\ell$:

- Sample a VOLE correlation $\vec{v} + \vec{w} = k \cdot \vec{y}$ corresponding to sid_{VOLE} , similarly to Fig. 1. That is, if $\mathcal{P}_1$ is honest, sample $\vec{v} \xleftarrow{\$} \mathbb{F}_{q^r}^\ell$, otherwise receive $\vec{v} \in \mathbb{F}_{q^r}^\ell$ from $\mathcal{A}$. If $\mathcal{P}_0$ is honest, compute $\vec{w} := k \cdot \vec{y} - \vec{v}$, otherwise receive $\vec{w} \in \mathbb{F}_{q^r}^\ell$ from $\mathcal{P}_0$. Return $(\vec{y}, \vec{w})$ to $\mathcal{P}_0$ and $(k, \vec{v})$ to $\mathcal{P}_1$.
- Sample authentications $[[\vec{v}]]_1^{(q')}$ and $[[\vec{w}]]_0^{(q')}$ corresponding to sid_{MAC} , as in Fig. 1, where a corrupt party may replace their component.
- Return $\langle k \cdot \vec{y} \rangle^{(q')} \leftarrow ([[\vec{w}]]_0^{(q')}, [[\vec{v}]]_1^{(q')})$.

Figure 2: Ideal functionality for authenticated VOLE.

For completeness, we recall the protocol from [4] that realizes this functionality along with batch opening in Appendix H.2 (Fig. 11). In essence, the parties can evaluate a circuit $C : \mathbb{F}_q^\ell \rightarrow \mathbb{F}$ of degree d on committed values $[[x_1]]^{(q)}, \dots, [[x_\ell]]^{(q)}$ using the homomorphic properties introduced in Appendix F. This results in a degree- d VOLE commitment to $F(x_1, \dots, x_\ell)$, for which the parties can verify whether it opens to zero. This readily generalizes to multiple circuits $C_1, \dots, C_m$, where a batch of commitments can be opened at the cost of one by opening a random linear combination. The protocol in Figure 11 is used implicitly in the protocols in Figures 6, 7, 10, 14 and 16, whenever arithmetic circuits are being evaluated on commitments or commitments are being opened.

2.6.1 Extending VOLE-Commitment and VOLE-ZK to Cyclic Groups.

As we later see, we will further require VOLE-ZK proofs to work over group relations where the exponent (in the field) is committed. We can extend the VOLE-ZK functionality $\mathcal{F}_{\text{VOLE-ZK}}^{q,1}$ from Figure 3 to prove statements regarding cyclic groups $\mathbb{G}$ of prime order q matching the VOLE-ZK field $\mathbb{F}_q$. That is, for any public $G \in \mathbb{G}$, the parties can take a commitment $[[x]]^{(q)} = ((x, w), (\Delta, v))$ on a value $x \in \mathbb{Z}_q$, where $v = w + x \cdot \Delta$, and lift it to a commitment on $H := G^x$ by simply exponentiating the tag and key as $W := G^w$ and $V := G^v$. Opening a value now simply proceeds by the prover sending (H, W) , and the verifier checking whether $V = H^\Delta \cdot W$. The prover can only open this commitment to a different value $h' \neq G^x$ if they successfully manage to guess $G, H \in \mathbb{G}$ with $H = G^\Delta$ and $G \neq 1_{\mathbb{G}}$, since the verifier will only accept an opening H', W' if $W' \cdot W^{-1} = (G^x \cdot H'^{-1})^\Delta$ (i.e., $H = W' \cdot W^{-1}$ and $G = G^x \cdot H'^{-1}$).

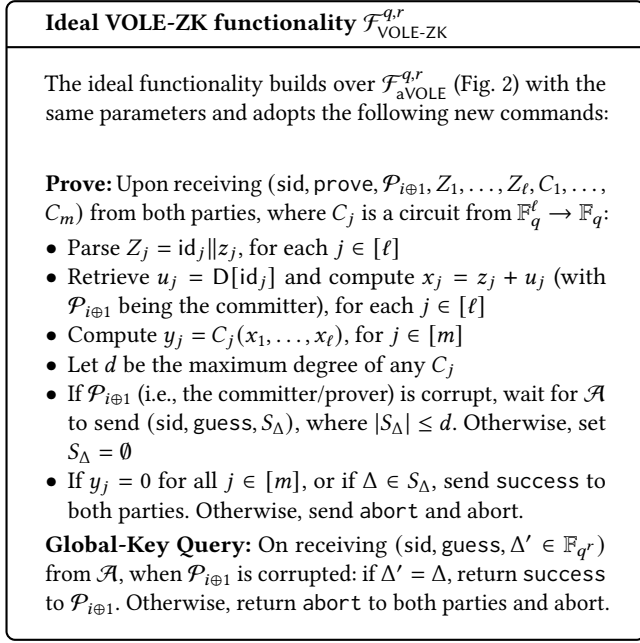


Figure 3: Ideal functionality for VOLE-based ZK.

Since this lifting preserves linearity of commitments, we can use this to extend the ideal functionality $\mathcal{F}_{\text{VOLE-ZK}}^{q,1}$ to prove statements of the form (1), where the prover knows $x_1, \dots, x_\ell \in \mathbb{Z}_q^\ell$ such that:

$$G_1^{x_1} \dots G_\ell^{x_\ell} = H, \quad (1)$$

where $G_1, \dots, G_\ell, H \in \mathbb{G}$ are arbitrary public elements and $1_\mathbb{G}$ is the identity. Using batching techniques as in Figure 11, we can extend this to proving multiple statements of the form (1), for $\vec{x}_1, \dots, \vec{x}_m \in \mathbb{Z}_q^\ell$, $\vec{G}_1, \dots, \vec{G}_m \in \mathbb{G}^\ell$ and $H_1, \dots, H_m \in \mathbb{G}$, at the cost of a single opening. We formalize the ideal functionality in Figure 4.

The protocol for realizing this is shown in Figure 12 (Appendix H.3). We prove in Theorem H.1 that this protocol realizes the functionality $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,r}$ in Figure 4, which works similarly to Figure 3 except that for proving statements about groups, a corrupt prover is allowed to try to cheat by guessing G^Δ for any $G \neq 1_\mathbb{G}$.

Looking ahead, in our SH-DOPRF protocols presented in Section 3, all openings and arithmetic operations on commitments are instantiated via invocations to the $\mathcal{F}_{\text{VOLE-ZK}}$ and $\mathcal{F}_{\text{VOLE-ZK-G}}$ functionalities.

2.7 Authenticated Secret Sharing

We say that a value $x \in \mathbb{F}$ is authenticated secret shared if parties $\mathcal{P}_0$ and $\mathcal{P}_1$ hold x_0 and x_1 such that $x = x_0 + x_1$, and the values x_0 and x_1 are authenticated under MAC keys Δ_1 and Δ_0 , respectively. We represent this more succinctly as

$$\langle x \rangle \leftarrow ([x_0]_0, [x_1]_1)$$

and also refer to $[x_i]_i$ as the share of party $\mathcal{P}_i$. Such sharings correspond to BDOZ-style authenticated secret sharing [11] and support

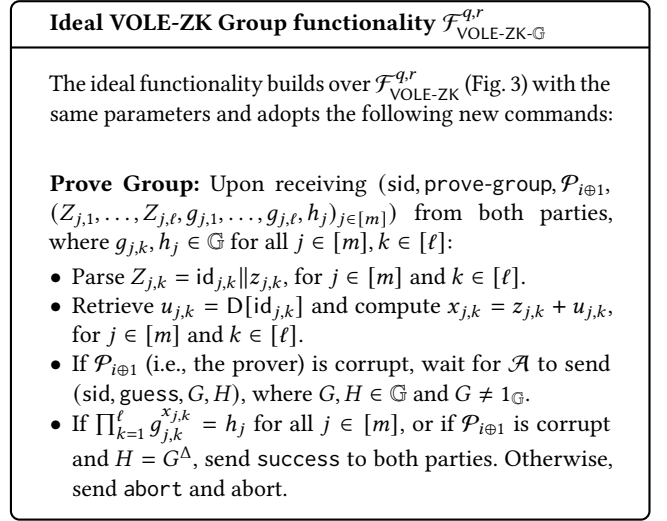


Figure 4: Ideal functionality for VOLE-based ZK with group arithmetic.

a wide range of operations such as linear combinations and multiplications, as detailed in [11]. Note that we can interpret a value $[x]_0$ as $\langle x \rangle$ via $\langle x \rangle \leftarrow ([x]_0, 0)$ (and similarly for $[x]_1$).

2.8 Pseudorandom Function

We work with two particular instantiations of PRFs that we now discuss (the definition can be found in Appendix E):

2.8.1 Dodis-Yampolskiy PRF. In particular, we work with the Dodis-Yampolskiy PRF [34] which is defined as

$$F_k(x) = G^{\frac{1}{k+x}}$$

where $G \in \mathbb{G}$ is a generator of the group $\mathbb{G}$ with prime order q , $x \in \mathbb{Z}_q$ and $k \xleftarrow{\$} \mathbb{Z}_q$.

As shown in [34], if the input domain X is restricted to $X \subset \mathbb{Z}_q$ for any polynomial-sized set X , the pseudorandomness follows from the decisional ℓ -Diffie-Hellman inversion assumption (ℓ -DDHI), where $\ell = |X|$. For the full input domain $\mathbb{Z}_q$, pseudorandomness can be shown in the generic group model [22]. (Note that both references consider the bilinear setting, since the focus is to obtain a verifiable random function, where correct evaluation is publicly verifiable, whereas for the PRF version the non-bilinear version is sufficient. We expect that a GGM proof in the non-bilinear setting follows by a straightforward adaptation, though to our knowledge this has not been written down explicitly.)

2.8.2 Dark-Matter PRF. Our second construction is based on the recent dark-matter PRF variant introduced by Aranha et al. [4]. Their construction in turn builds upon the line of work initiated by Boneh et al. [15] of constructing post-quantum PRF candidates by mixing linear operations over small moduli. Subsequent works [2, 3, 33] introduced different variants of this original PRF construction together with protocols for oblivious evaluation, but many of these have seen attacks weakening their security [6, 27, 28, 43, 66]. The

variant introduced by [4] can be seen as a more conservative option, and so we base our construction on theirs.

Recent cryptanalysis [5] showed that using a code $C \in \mathbb{F}_3^{r' \times d}$ in systematic form (i.e. of the form $C = [I_d | C']^T$) for the input-encoding required to transform the weak PRF to a strong PRF, as proposed in [3, 4, 15], together with the typical choice of parameters where $d = \lambda$ and $r = 2\lambda$ (i.e., the length of the encoded input), as proposed in [3, 4], enables an efficient key-recovery attack on the strong PRF using only ≈ 300 queries. Of the suggested countermeasures in [5], the one with the least impact on the structure of the SH-DOPRF protocol presented in this work would be to use a code C in non-systematic form instead. Since the binary decomposition operation now has to be applied to all r' coordinates of $C \cdot_3 \vec{x}$, this simultaneously has the effect that the length of the encoded input $r = 2r' + 1$ increases to $3\lambda + 1$ even when we put $d = \lambda$ and $r' = 3\lambda/2$ (i.e., virtually unchanged from [4]). Together, these changes have the effect that the attack discussed in [5] becomes inadmissible.

In terms of our SH-DOPRF protocol in Figure 7, the main change is that the binary decomposition is now applied to all entries of $\vec{z}^j$, which implies we need $\approx \lambda$ extra local daBits from preprocessing. All the steps after the input encoding are unchanged but will become $\approx 1.5\times$ more expensive due to the increased length of the encoded input.

For an input $y \in \mathbb{F}_{2^r}$, key $k \in \mathbb{F}_{2^r}$ and public (random) matrix $B \in \mathbb{F}_3^{t \times r}$, the weak PRF (i.e., pseudorandom on randomly chosen inputs) is defined as

$$F'_k(y) := B \cdot_3 \text{vec}(k \cdot y), \quad (2)$$

where we use the shorthand $\cdot_q$ to denote multiplication modulo q . To obtain a strong PRF (i.e., pseudorandom on any choice of input), we restrict the input space to $\mathbb{F}_2^d$ and encode⁶ an input $x \in \mathbb{F}_2^d$ as

$$y = \text{elt}(\text{bin}(G \cdot_3 x) \| 1), \quad (3)$$

where $G \in \mathbb{F}_3^{r' \times d}$ is a linear code and bin denotes binary decomposition. The strong PRF is defined as

$$F_k(\vec{x}) := F'_k(y), \quad (4)$$

with y as in (3) and F' as in (2), and $2r' + 1 = r$. For λ -bit security, proposed parameters [5] are $d = \lambda$, $r = 3\lambda + 1$ and $t = \lambda/\log_2 3$.

3 Shuffled Distributed OPRF (SH-DOPRF)

We define and construct Shuffled Distributed Oblivious PseudoRandom Functions (SH-DOPRF), our core building block for different PSO protocols.

Definition. The functionality $\mathcal{F}_{\text{SH-DOPRF}}$ first samples a random PRF key k and initializes the VOLE functionality. Note that this functionality works in the VOLE-hybrid model and can be seen as inheriting all commands and identifiers defined before. $\mathcal{P}_0$ holds a vector $\vec{x}$ and $\mathcal{P}_1$ holds a permutation π as input. The ideal functionality sends the PRF values $F_k(\vec{x})$ to $\mathcal{P}_1$ and the permuted PRF values $\pi(F_k(\vec{x}))$ to $\mathcal{P}_0$. This is accompanied by corresponding commitments to $\vec{x}$ and π (Note that the permutation π is converted to vector form).

⁶Note that this encoding is different than the one originally used in [4] in order to avoid the vulnerability discovered by [5]

Shuffled DOPRF $\mathcal{F}_{\text{SH-DOPRF}}^{q,r}$

The ideal functionality builds over $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,r}$ (Fig. 4) with the same parameters. The functionality is parametrized by a PRF family $F : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda$, indexed by the security parameter $\lambda \in \mathbb{N}$, which we omit for brevity.

Initialize: Upon receiving $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{init})$ from both parties, sample a PRF key $k \xleftarrow{\$} \mathcal{K}$ and execute $(\text{sid}_{\text{MAC}}, \text{init}, i)$ as in $\mathcal{F}_{\text{SVOLE}}^{q,r}$ for $i \in \{0, 1\}$ to obtain global MAC keys $\Delta_0, \Delta_1 \in \mathbb{F}_{q^r}$. The functionality stores $(\text{sid}_{\text{PRF}}, k)$ and $(\text{sid}_{\text{MAC}}, \Delta_0, \Delta_1)$, outputs Δ_0 to $\mathcal{P}_0$ and Δ_1 to $\mathcal{P}_1$, and ignores all future setup calls with (one of) the same respective sid 's.

Evaluate:

- Receive $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{eval}, \vec{x})$ from $\mathcal{P}_0$ where $\vec{x}$ is a vector of size n .
- Compute $F_k(\vec{x}) = (F_k(x_1), F_k(x_2), \dots, F_k(x_n))$ and generate commitment $[\vec{x}]_0$.
- Output commitment $[\vec{x}]_0$ to both parties. Additionally, send $F_k(\vec{x})$ to $\mathcal{P}_1$.
- Receive $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \pi)$ from $\mathcal{P}_1$ where $\pi : [n] \rightarrow [n]$ is a permutation.
- Compute $\pi(F_k(\vec{x})) = (F_k(x_{\pi(1)}), \dots, F_k(x_{\pi(n)}))$ and generate commitment $[\vec{\pi}]_1$, where $\vec{\pi} = (\pi(1), \pi(2), \dots, \pi(n))$.
- Output commitment $[\vec{\pi}]_1$ to both parties. Additionally, send $\pi(F_k(\vec{x}))$ to $\mathcal{P}_0$.

Key-Guess: If $\mathcal{P}_i$ is corrupted, receive $(\text{sid}_{\text{PRF}}, \text{guess}, k')$ from $\mathcal{A}$ with $k' \in \mathcal{K}$. If $k' = k$, return success to $\mathcal{P}_i$ and ignore all future key-guess queries with the same sid_{PRF} . Otherwise, return abort to both parties and abort.

Figure 5: Ideal functionality for Shuffled DOPRF

3.1 Our Construction based on Dodis-Yampolskiy PRF

Our first protocol for achieving our desired SH-DOPRF functionality is based on the Dodis-Yampolskiy PRF as introduced in Section 2.8.1 and is formally presented in Figure 6. We first discuss the specific non-trivial VOLE-ZK relations required and then discuss the high-level construction.

3.1.1 Specific VOLE-ZK over groups relations. In this section, we detail two specific VOLE-ZK over groups relations, which we rely on for our DY-OPRF based construction to prove certain forms of consistency between group elements and VOLE-committed values. Note that for clarity, we work with the simplified notation $[\vec{x}]$ instead of $[\vec{x}]_b^{(q)}$ in this section.

Batch proof of committed exponents Given a batch of m pairs of public group elements (G_i, H_i) , for $i \in [m]$, suppose the prover wants to show that $H_i = G_i^{x_i}$ for all i , where the exponents are VOLE-committed values $[\vec{x}_i]$. This can be done easily using the

prove-group command of Figure 4, with $\ell = 1$, by giving the inputs $(G_i, H_i)_i$ and corresponding identifiers for the commitments $\llbracket x_i \rrbracket$.

When instantiating this with the protocol in Figure 12 (Appendix H.3), the computational cost for each of the prover and verifier is dominated by a single multi-exponentiation of length m for the prover and $m\ell + m = 2m$ for the verifier. We note that the verifier does not need to send all the χ_j challenges, and can instead send a seed that is expanded with a PRG; this makes the communication cost independent of the batch size.

Proof of shuffled committed exponents To obtain a shuffled set of OPRF outputs, another important building block is a proof of consistency between a set of exponentiations $G_1, G_2, \dots, G_n, H_1, H_2, \dots, H_n$ with a set of committed exponents $\llbracket x_1 \rrbracket, \llbracket x_2 \rrbracket, \dots, \llbracket x_n \rrbracket$ along with a committed permutation $\llbracket \pi(1) \rrbracket, \llbracket \pi(2) \rrbracket, \dots, \llbracket \pi(n) \rrbracket$. Specifically, two parties wish to verify that $H_i = G_{\pi(i)}^{x_{\pi(i)}}$, which means the set of values $H_1, \dots, H_n$ is a *shuffled* set of exponentiations with bases $G_1, \dots, G_n$ and committed exponents $x_1, \dots, x_n$.

The proving protocol is based on the proof-of-shuffle from Bayer and Groth [9], with full details deferred to Figure 13 (Appendix H.3). In the first stage of the protocol, the verifier sends a random challenge $c \in \mathbb{F}_q$, and the prover then commits to $c^{\pi(i)}$ for each i . Next, the prover wants to show that both $\{\pi(i)\}_i$ is a permutation of $\{1, \dots, n\}$ and $\{c^{\pi(i)}\}_i$ is a permutation of $\{c^i\}_i$ under the *same* permutation. This is done using a Neff-style shuffle argument [55]. The verifier sends two additional challenges $c_0, c_1 \in \mathbb{F}_q$. Using c_1 , the two pairs of permuted sets are randomly folded into one by defining the commitments $\llbracket d_i \rrbracket = \llbracket \pi(i) \rrbracket + c_1 \cdot \llbracket c^{\pi(i)} \rrbracket$. The goal is then to prove that $\{d_i\}_i$ is a permutation of $\{i + c_1 \cdot c^i\}_i$, which is done with a polynomial identity test with the challenge c_0 , by checking that $\prod_{i=1}^n (c_0 + i + c_1 \cdot c^i) = \prod_{i=1}^n (c_0 + \llbracket d_i \rrbracket)$. This product check can easily be done using the $\mathcal{F}_{\text{VOLE-ZK}}$ functionality.

Once we have established commitments to $c^{\pi(i)}$, proving that $H_i = G_{\pi(i)}^{x_{\pi(i)}}$ for all i can be done by checking that $\prod_i H_i^{\llbracket c^{\pi(i)} \rrbracket} = \prod_i G_i^{c^i \cdot x_i}$, using $\mathcal{F}_{\text{VOLE-ZK-G}}$. Since c was sampled before committing to $\pi(i)$, this can be seen as a polynomial identity test in the exponent, so only holds with probability at most n/q so if the statement is false.

We remark that the statement in [9] differs from ours: they prove that one set of ciphertexts is a permutation of another set of ciphertexts with respect to *some* randomness, whereas we prove that one set of group elements is a permutation of another set of group elements raised to a set of *VOLE-committed* exponents. Concretely, in the protocol of [9], the prover first commits to $\{\pi(i)\}_i$ and $\{c^{\pi(i)}\}_i$, and then prove that $\{c^{\pi(i)}\}_i$ is a permutation of $\{c^i\}_i$ via a Neff-style shuffle argument [55]. We replace the Pedersen commitments with VOLE commitments and the shuffling argument with VOLE-ZK. The remaining step in [9] involves a multi-exponentiation argument based on ElGamal ciphertexts, which we adapt by lifting VOLE-ZK to the exponent (Figure 12) and incorporating the pre-existing VOLE-committed exponents (Figure 13). Using lightweight VOLE commitments and VOLE-ZK significantly reduces the computational cost of both the product check and the final multi-exponentiation argument.

3.1.2 High-level description of construction. We provide a high-level description of the protocol herewith. Recall that all openings and arithmetic operations on commitments are modeled via invocations to the ideal VOLE-ZK functionality as discussed in Section 2.6 and are generally left implicit for readability. The protocol proceeds in four primary stages:

- **Initialize:** Here, parties initialize the VOLE functionality and commit to their shares of the PRF key k_0 and k_1 respectively.
- **Evaluate Step 1: $\mathcal{P}_0$ sends $\vec{a} = \vec{r} \odot (k_0 + k_1 + \vec{x})$ to $\mathcal{P}_1$.**
Here, the goal is for $\mathcal{P}_1$ to obtain $\vec{a}$. $\mathcal{P}_0$ can locally compute their “part” of $\vec{a}$ as $\vec{a}_0 = \vec{r} \odot (k_0 + \vec{x})$. Parties can jointly invoke the authenticated vole functionality to generate additive shares for the other “part” of $\vec{a}$ as $\vec{a}_1 = \vec{r} \cdot k_1$. $\mathcal{P}_0$ uses their additive share as a mask to send $\vec{a}_0$ (accompanied with the corresponding VOLE-ZK proof over fields) which allows $\mathcal{P}_1$ to reconstruct $\vec{a}$ by simple addition.
- **Evaluate Step 2: $\mathcal{P}_1$ computes PRF values $\vec{X} = F_{k_0+k_1}(\vec{x})$.**
Here, the goal is for $\mathcal{P}_1$ to obtain the PRF values on $\vec{x}$. To do this, it is sufficient to observe that, for the DY-PRF and any values x and r ,

$$F_{k_0+k_1}(x) = G^{\frac{1}{k_0+k_1+x}} = (G^r)^{\frac{1}{r(k_0+k_1+x)}}$$

holds. Since $\mathcal{P}_1$ already holds the value $r(k_0 + k_1 + x)$ in the clear by the previous step, it is sufficient for $\mathcal{P}_0$ to send G^r . Note that, this is done for the entire vectors $\vec{x}$ and $\vec{r}$ and is accompanied by the batch proof of committed exponents. We also require VOLE-ZK proofs (over fields) for inverting the exponent.

- **Evaluate Step 3: $\mathcal{P}_1$ sends the shuffled PRF values to $\mathcal{P}_0$.**
Here, the goal is for $\mathcal{P}_1$ to send back the PRF values obtained in the previous step after shuffling via the permutation π . Observe that $\mathcal{P}_1$ already holds implicit commitments to the PRF exponent via $\vec{a}$. It is thus sufficient for $\mathcal{P}_1$ to now commit to the permutation π and give two proofs: a zk proof that π is a permutation, and a zk proof of shuffled committed exponents. This step is combined into one zk proof and is presented formally in Figure 13.

THEOREM 3.1. *The protocol $\Pi_{\text{SH-DOPRF}}^{\text{DY}}$ in Figure 6 securely realizes the Shuffled DOPRF functionality $\mathcal{F}_{\text{SH-DOPRF}}^{q,1}$ defined in Figure 5 against malicious adversaries as per Definition D.1 in the $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,1}$ -hybrid model.⁷*

PROOF. The proof can be found in Appendix I. $\square$

3.2 Our Construction based on Dark-Matter PRF

Our second protocol realizing the SH-DOPRF functionality is based on the recently introduced dark-matter PRF variant and OPRF protocol from Aranha et al. [4] as introduced in Section 2.8.2. Recall that for a key $k \in \mathbb{F}_{2^r}$ and input $x \in \mathbb{F}_2^d$, the PRF is given by

$$F_k(x) = B \cdot \text{vec}(k \cdot \text{elt}(\text{bin}(G \cdot x) \| 1)),$$

where $B \in \mathbb{F}_3^{t \times r}$ is a public random matrix, $G \in \mathbb{F}_3^{r' \times d}$ is a linear code⁸, $\text{vec} : \mathbb{F}_{2^r} \rightarrow \mathbb{F}_2^r$ and $\text{elt} : \mathbb{F}_2^r \rightarrow \mathbb{F}_{2^r}$ are maps between vector

⁷Recall that $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,1}$ adopts $\mathcal{F}_{\text{VOLE-ZK}}^{q,1}$, $\mathcal{F}_{\text{aVOLE}}^{q,1}$ and $\mathcal{F}_{\text{sVOLE}}^{q,1}$ as sub-functionalities to enforce that consistent global keys are used throughout.

⁸Note that this slightly differs from the construction originally proposed by [4] to avoid the vulnerability coming from using a systematic code G discovered by [5].

Dodis-Yampolskiy-based Shuffled and Distributed OPRF $\Pi_{\text{SH-DOPRF}}^{\text{DY}}$

Parameters: Two parties, $\mathcal{P}_0$ and $\mathcal{P}_1$. $\mathcal{P}_0$ holds a batch of n inputs $\vec{x}$ and $\mathcal{P}_1$ holds a permutation $\pi : [n] \rightarrow [n]$. A group $\mathbb{G}$ of large prime order q with generator G . The prime-ordered field of exponentiations $\mathbb{F}_q$ for $\mathbb{G}$.

Initialize:

- $\mathcal{P}_0$ and $\mathcal{P}_1$ send $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{init})$ to $\mathcal{F}_{\text{aVOLE}}^{q,1}$ in both directions and respectively receive $\Delta_0, \Delta_1 \xleftarrow{\$} \mathbb{F}_q$ associated with sid_{MAC} as well as $k_0, k_1 \xleftarrow{\$} \mathbb{F}_q$ associated with sid_{PRF} . The parties also receive commitments $[\![k_0]\!]_0^{(q)}$ and $[\![k_1]\!]_1^{(q)}$.

Evaluate:

- $\mathcal{P}_0$ sends $\vec{a} = \vec{r} \odot (k_0 + k_1 + \vec{x})$ to $\mathcal{P}_1$.
 - $\mathcal{P}_0$ commits to its input as $[\![\vec{x}]\!]_0^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle \vec{x}, q, 1 \rangle, (\Delta_1, q, 1))$.
 - $\mathcal{P}_0$ samples $\vec{r} \xleftarrow{\$} \mathbb{F}_q^n$ and commits to the random masks as $[\![\vec{r}]\!]_0^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle \vec{r}, q, 1 \rangle, (\Delta_1, q, 1))$.
 - $\mathcal{P}_0$ locally computes $\vec{a}_0 = \vec{r} \odot (\vec{x} + k_0)$ and commits to it as $[\![\vec{a}_0]\!]_0^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle \vec{a}_0, q, 1 \rangle, (\Delta_1, q, 1))$.
 - $\mathcal{P}_0$ opens $[\![\vec{a}_0]\!]_0^{(q)} - [\![\vec{r}]\!]_0^{(q)} \odot ([\![\vec{x}]\!]_0^{(q)} + [\![k_0]\!]_0^{(q)})$ to $\mathcal{P}_1$ via $\mathcal{F}_{\text{VOLE-ZK}}$, who aborts if the opening is non-zero.
 - Parties send $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, [\![\vec{r}]\!]_0^{(q)})$ to $\mathcal{F}_{\text{aVOLE}}^{q,1}$. $\mathcal{P}_0$ receives $[\![\vec{r}_0]\!]_0^{(q)}$, and $\mathcal{P}_1$ receives $[\![\vec{r}_1]\!]_1^{(q)}$, satisfying $\vec{r}_0 + \vec{r}_1 = k_1 \cdot \vec{r}$.
 - $\mathcal{P}_0$ opens $[\![\vec{a}_0]\!]_0^{(q)} + [\![\vec{r}_0]\!]_0^{(q)}$ to $\mathcal{P}_1$ via $\mathcal{F}_{\text{VOLE-ZK}}$, and the parties compute $[\![\vec{a}]\!]_1^{(q)} = \vec{a}_0 + \vec{r}_0 + [\![\vec{r}_1]\!]_1^{(q)}$.
- $\mathcal{P}_1$ computes PRF values $\vec{X} = F_{k_0+k_1}(\vec{x})$.
 - $\mathcal{P}_0$ locally computes $\vec{R} = G^{\vec{r}}$ and sends it to $\mathcal{P}_1$. The parties verify if $G[\![\vec{r}]\!]_0^{(q)} \odot \vec{R}^{-1}$ opens to $1_{\mathbb{G}}$ via $\mathcal{F}_{\text{VOLE-ZK-G}}$.
 - $\mathcal{P}_1$ locally computes $\vec{a}' = \vec{a}^{-1}$ and commits to it as $[\![\vec{a}']\!]_1^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle \vec{a}', q, 1 \rangle, (\Delta_0, q, 1))$.
 - $\mathcal{P}_1$ opens $[\![\vec{a}']\!]_1^{(q)} \odot [\![\vec{a}]\!]_1^{(q)} - \vec{1}$ to $\mathcal{P}_0$ via $\mathcal{F}_{\text{VOLE-ZK}}$, who aborts if the opening is non-zero.
 - $\mathcal{P}_1$ locally computes $\vec{X} = \vec{R}^{\vec{a}'}$. // Note that $\vec{a}' = \frac{1}{\vec{r}(k+\vec{x})}$.
- $\mathcal{P}_1$ sends the shuffled PRF values to $\mathcal{P}_0$.
 - $\mathcal{P}_1$ sends $\pi(\vec{X})$ to $\mathcal{P}_0$.
 - Parties invoke $\Pi_{\text{ShuffleExp}}$ (Figure 13) with the public vectors $\vec{R}$ and $\vec{X}_{\pi} := \pi(\vec{X})$, where $\mathcal{P}_1$ acts as the prover with input $(\pi, [\![\vec{a}']\!]_1^{(q)})$ and $\mathcal{P}_0$ acts as the verifier with input $([\![\vec{a}']\!]_1^{(q)})$. Parties abort if the subprotocol aborts.

Figure 6: Protocol for Dodis-Yampolskiy-based Shuffled and Distributed OPRF

and extension field representations, and $2r' + 1 = r$. Proposed parameters [5] are $d = \lambda$, $r = 3\lambda + 1$ and $t = \lambda/\log_2 3$.

Aranha et al. [4] give a maliciously secure protocol for obliviously evaluating this PRF on a batch of inputs $(x_i)_{i \in [n]}$ held by $\mathcal{P}_0$ and a key $k \in \mathbb{F}_{2^n}$ held by $\mathcal{P}_1$, such that $\mathcal{P}_0$ receives $(F_k(x_i))_{i \in [n]}$. To obtain a SH-DOPRF, we first extend their protocol to support:

- Authenticated inputs $[\![x_i]\!]_0^{(2)}$ for $\mathcal{P}_0$;
- Secret-shared key $\langle k \rangle^{(2^n)} \leftarrow ([\![k_0]\!]_0^{(2^r)}, [\![k_1]\!]_1^{(2^r)})$;
- Output authenticated PRF evaluations $[F_k(x_i)]_1^{(3)}$ to $\mathcal{P}_1$.

After obtaining these, $\mathcal{P}_1$ shuffles the PRF evaluations according to their input permutation $\pi : [n] \rightarrow [n]$, sends $(F_k(x_{\pi(i)}))_{i \in [n]}$ to $\mathcal{P}_0$ and proves that these are consistent with $[F_k(x_i)]_1^{(3)}$ and the commitment to π , using the protocol from Figure 14. The secret-sharing of the key makes sure that the PRF key is random and unknown to both parties, and the authentication on the secret-shares allows them to make sure that the same key is being used in different executions of the protocol. The authentication on the PRF evaluations allows $\mathcal{P}_1$ to prove that they send correctly shuffled PRF evaluations to $\mathcal{P}_0$ using a VOLE-based shuffling proof. Finally, to make sure that the commitments to the inputs, permutation and PRF outputs can all be embedded into the same field, we reduce the input space for the SH-DOPRF protocol in Figure 7 to $\mathcal{X} = \mathbb{F}_2^t$, but

inside of the protocol evaluate the PRF with the original parameters, that is, input space $\mathbb{F}_2^d$ as presented in Section 2.8.2. Concretely, this means that $\mathcal{P}_0$ appends $d - t = \lambda - \lambda/\log_2 3$ zeros to their input before committing to it.

The main changes we make with respect to the protocol from [4] are: (1) The parties jointly compute the encoding as in equation (3) on the committed inputs $[\![x_i]\!]_0^{(2)}$ to obtain $[\![y_i]\!]_0^{(2^r)}$, which additionally requires d VOLE entries over $\mathbb{F}_2$, $2r' + d$ local daBits and $2d + 2r' = 5\lambda$ bits of (amortized) communication per evaluation; (2) We use a different authenticated VOLE protocol (Fig. 10), which actually needs $r = 3\lambda + 1$ less bits of (amortized) communication per evaluation and r less VOLE entries over $\mathbb{F}_2$, since this protocol is optimized to take authenticated $[\![y_i]\!]_0^{(2^r)}$. (3) Sending the shuffled PRF evaluations to $\mathcal{P}_0$ and proving these are consistent with $[\![\pi]\!]_1^{(3^t)}$, which requires an additional t VOLE entries over $\mathbb{F}_3$ and $3t \cdot \log_2 3 = 3\lambda$ bits of (amortized) communication per evaluation. So in total for a batch of n inputs, the protocol in Figure 7 requires an additional $\approx 5\lambda n$ bits of (amortized) communication in the online phase, and minimal computational overhead, with respect to the protocol from [4].⁹

⁹To obtain a fair comparison, we assume [4] incorporates the same countermeasures against the attack in [5] as we do. That is, removing the systematic form of G and increasing r from 2λ to $3\lambda + 1$.

The detailed protocol is presented in Figure 7 and we recall the ideal daBits functionality $\mathcal{F}_{\text{daBits}}^{r^{(2)}, r^{(3)}}$ from [4] in Appendix G (Fig. 8). Note that the daBits functionality $\mathcal{F}_{\text{daBits}}^{r^{(2)}, r^{(3)}}$ adopts the same parameters as well as all commands from the earlier VOLE functionalities (Figs. 1, 2, 3) for $(q, r) \in \{(2, r^{(2)}), (3, r^{(3)})\}$.

In Appendix J, we prove that the protocol $\Pi_{\text{SH-DOPRF}}^{\text{DM}}$ in Figure 7 securely realizes the ideal functionality $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$.

4 Maliciously Secure Private Set Operations

In this section, we show how parties can achieve maliciously secure private set operations such as PSI-Cardinality, PSI-Sum, Circuit-PSI, and PSU, using the Shuffled Distributed OPRF obtained in the prior sections. These can be achieved via maliciously secure multi-query Reverse Private Membership Test (mqRPMT) with minor modifications. As such, we first define and construct maliciously secure mqRPMT as a major building block.

4.1 Multi-Query Reverse Private Membership Test

We first recall the definition of mqRPMT in the semi-honest setting. Here, two parties $\mathcal{P}_0$ and $\mathcal{P}_1$ hold vectors $\vec{x}$ and $\vec{y}$. The output consists of an indicator bit vector $\vec{b}$ where $\vec{b}[i] = 1$ iff $x_i \in \vec{y}$. This vector $\vec{b}$ is given as output to $\mathcal{P}_1$ while $\mathcal{P}_0$ outputs $\perp$. Since we are in the malicious setting, we require a stronger functionality where all vectors are committed (for further computation). More precisely, $\mathcal{P}_0$ additionally commits to the vector $\vec{x}$ and $\mathcal{P}_1$ commits to the vector $\vec{b}$ where these commitments are consistent with respect to the final output. The functionality is formally presented in Figure 15 in Appendix K. We note that the cardinality of the intersection is leaked to both parties.

We can achieve mqRPMT as follows: We invoke Shuffled Distributed OPRF twice (once with $\mathcal{P}_0$ holding vector $\vec{x}$ and $\mathcal{P}_1$ holding permutation π ; and the other time with $\mathcal{P}_0$ holding permutation ρ and $\mathcal{P}_1$ holding vector $\vec{y}$). This allows both parties to get permuted PRF values $\pi(F_k(\vec{x}))$ and $\rho(F_k(\vec{y}))$ in the clear. Using these, they can compute the indicator bit vector $\vec{b}'$ where $\vec{b}'[i] = 1$ iff $F_k(x_{\pi(i)}) \in \rho(F_k(\vec{y}))$. It is easy to see that this condition is equivalent to stating that $b'_i = 1$ iff $x_{\pi(i)} \in \vec{y}$. Therefore, this vector $\vec{b}'$ is just a permutation of the vector $\vec{b}$ that we want to achieve with respect to the permutation π . Since $\mathcal{P}_1$ knows the permutation π , it can compute $\vec{b}$ and commit to it. This is accompanied by a zero-knowledge proof of permutation to ensure consistency. We give a formal description of our protocol for achieving maliciously secure mqRPMT in Figure 16 in Appendix K.1.

4.2 Other Advanced Private Set Operations

In this section, we present maliciously security protocols for important variants of PSI (PSI-Cardinality, PSI-Sum, Circuit-PSI) and PSU. Our constructions for PSI-Cardinality, PSI-Sum, and PSU all achieve *two-sided output*. We discuss two such variants here and provide constructions for the other variants in Appendix L.

PSI-Cardinality:

- *Functionality.* $\mathcal{P}_0$ holds a set S_x and $\mathcal{P}_1$ holds a set S_y . Both parties receive the intersection's cardinality $|S_x \cap S_y|$ as output.

- *Protocol.* It is easy to see that this is already obtained as a byproduct in our mqRPMT protocol. In fact, the last 3 steps can be omitted and parties can simply output $\sum_i b'_i$ to get the cardinality.

Circuit-PSI:

- *Functionality.* $\mathcal{P}_0$ holds a set S_x (corresponding to a vector $\vec{x}$) and $\mathcal{P}_1$ holds a set S_y (along with vector $\vec{y}$). Both parties receive authenticated secret shares of the intersection as output. More formally, both parties receive $\langle \vec{z} \rangle$ where $z_i = x_i$ if $x_i \in \vec{y}$ and $z_i = 0$ otherwise. Additionally, both parties get the cardinality of the intersection $|S_x \cap S_y|$ as output.
- *Protocol.* After running our mqRPMT protocol, parties hold commitments to the vector $\vec{x}$ and indicator bit vector $\vec{b}$. Viewing these commitments as authenticated secret shares, parties run a maliciously secure element-wise product protocol between $\vec{x}$ and $\vec{b}$ to get the output. The size of the intersection is already obtained via the mqRPMT protocol as discussed earlier.

5 Implementation

We implement the SH-DOPRF protocol based on DY-PRF and four different Private Set Operations that takes this primitive as a building block: PSI-Cardinality, Circuit-PSI, PSI-Sum, and PSU. For PSI-Cardinality and PSI-Sum, we compare with two prior works, with Miao et al. [54] using heavy public-key operations and Yang et al. [73] being more recent with an offline-online approach. Yang et al. [73] also achieve Circuit-PSI, hence we include the comparison for this problem as well. The SH-DOPRF primitive can also be applied to Private Set Union, and we put comparisons with the most efficient PSU work by Pu et al. [61], along with previously mentioned results, in Table 1. The table reports the communication cost¹⁰ and running time for multiple set sizes: 2^{16} , 2^{18} and 2^{20} . We measure the running time in three network settings: LAN, WAN with bandwidth 200 Mbps, and WAN with bandwidth 50 Mbps. The WAN setting is simulated with the Linux tc command, with round-trip time in both cases set to 80ms. We remark that the impact of network latency in both our work and related works is insignificant (concretely, less than 1 second in the WAN settings) compared to the overall cost, since the round count is a small constant. We therefore omit the comparison of round complexity in this section.

We do not compare the performance for *plain* PSI because state-of-the-art maliciously secure plain PSI is almost as efficient as its semi-honest counterpart [62, 65], whereas for the other variants there remains a significant gap in concrete efficiency between the malicious and semi-honest settings, as discussed in Section 1. Therefore, we focus on these other variants.

Experiments. All our experiments require two parties, which we emulate on two AWS c5a.8xlarge machines. For fair comparison, we also run the code from [73] and [61] on the same machines, while the code for [54] is not open-sourced, so we estimate the numbers based on what is reported in the paper. We provide more details on running the code and estimating the cost of mentioned works in Appendix M.

Implementation details. We implement the protocol in Rust and the code is available at: <https://github.com/phuocchubeo123/circuit->

¹⁰We report the total amount of bytes sent between the two parties.

Dark-Matter-based Shuffled and Distributed OPRF $\Pi_{\text{SH-DOPRF}}^{\text{DM}}$

Parameters: Two parties, $\mathcal{P}_0$ and $\mathcal{P}_1$. $\mathcal{P}_0$ holds a batch of n inputs $\vec{x} \in (\mathbb{F}_2^t)^n$ and $\mathcal{P}_1$ holds a permutation $\pi : [n] \rightarrow [n]$. The PRF is parametrized by a public matrix $B \in \mathbb{F}_3^{t \times r}$ and a linear code $G \in \mathbb{F}_3^{r' \times d}$. Integers $r, t, d, r' \in \mathbb{N}$ depending on the security parameter λ , with $r \geq \lambda$, $t \geq \lambda / \log_2 3$, $2r' + 1 = r$ and $3^t \geq n$.

Initialize:

- For each $i \in \{0, 1\}$, the parties send $(\text{sid}_{\text{MAC}}, \text{init}, i, 3)$ to $\mathcal{F}_{\text{daBits}}^{r,t}$ and respectively receive $\Delta_i^{(3)} \xleftarrow{\$} \mathbb{F}_{3^t}$.
- $\mathcal{P}_0$ and $\mathcal{P}_1$ send $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{init}, 2)$ to $\mathcal{F}_{\text{daBits}}^{r,t}$ in both directions and respectively receive $\Delta_0^{(2)}, \Delta_1^{(2)} \xleftarrow{\$} \mathbb{F}_{2^r}$ associated with sid_{MAC} as well as $k_0, k_1 \xleftarrow{\$} \mathbb{F}_{2^r}$ associated with sid_{PRF} . The parties also receive commitments $\llbracket k_0 \rrbracket_0^{(2r)}$ and $\llbracket k_1 \rrbracket_1^{(2r)}$.
- The parties send $(\text{sid}_{\text{MAC}}, \text{laBits}, (2r' + d) \cdot n, 0)$ and $(\text{sid}_{\text{MAC}}, \text{daBits}, r \cdot n)$ to $\mathcal{F}_{\text{daBits}}^{r,t}$, and receive $(\llbracket \vec{b} \rrbracket_0^{(2)}, \llbracket \vec{b} \rrbracket_0^{(3)})$ and $(\langle \vec{c} \rangle^{(2)}, \langle \vec{c} \rangle^{(3)})$, respectively, with $\vec{b} \in \{0, 1\}^{2r' + d \cdot n}$ and $\vec{c} \in \{0, 1\}^{r \cdot n}$.

Evaluate PRF:**• Compute Input Encoding:**

- $\mathcal{P}_0$ embeds their inputs $\vec{x}$, where $x_i \in \mathbb{F}_2^t$, into $\mathbb{F}_2^d$ as $x_i \leftarrow x_i \parallel \vec{0}^{d-t} \in \mathbb{F}_2^d$ and commits to the embedded inputs as $\llbracket \vec{x} \rrbracket_0^{(2)} \leftarrow \Pi_{\text{Commit}}(\langle \vec{x}, 2, r \rangle, (\Delta_1^{(2)}, 2, r))$. Then proves to $\mathcal{P}_1$ that $\llbracket \vec{x}[t + 1, d] \rrbracket_0^{(2)}$ opens to zero via $\mathcal{F}_{\text{VOLE-ZK}}$.
- Compute and open $\llbracket \vec{z} \rrbracket_0^{(2)} := \llbracket \vec{x} \rrbracket_0^{(2)} + \llbracket \vec{b}[1, d] \rrbracket_0^{(2)}$ via $\mathcal{F}_{\text{VOLE-ZK}}$, and put $\llbracket \vec{x} \rrbracket_0^{(3)} := \vec{e} + \llbracket \vec{b}[1, d] \rrbracket_0^{(3)} + \vec{e} \odot \llbracket \vec{b}[1, d] \rrbracket_0^{(3)}$.
- Compute $\llbracket \vec{x} \rrbracket_0^{(3t)} \leftarrow \text{elt}(\llbracket \vec{x}[1, t] \rrbracket_0^{(3)})$ (these will be used in the final output of the protocol).
- Compute pointwise $\llbracket \vec{z} \rrbracket_0^{(3)} := G \odot \llbracket \vec{x} \rrbracket_0^{(3)}$ and compute the pointwise binary decomposition $\llbracket \text{bin}(\vec{z}) \rrbracket_0^{(3)}$ via $\mathcal{F}_{\text{VOLE-ZK}}$ as
$$\llbracket \vec{y} \rrbracket_0^{(3)} := (2 \cdot \llbracket \vec{z} \rrbracket_0^{(3)} + 2 \cdot \llbracket \vec{z} \rrbracket_0^{(3)} \odot \llbracket \vec{z} \rrbracket_0^{(3)}, \llbracket \vec{z} \rrbracket_0^{(3)} + 2 \cdot \llbracket \vec{z} \rrbracket_0^{(3)} \odot \llbracket \vec{z} \rrbracket_0^{(3)})$$
- Compute and open $\llbracket \vec{f} \rrbracket_0^{(3)} := \llbracket \vec{y} \rrbracket_0^{(3)} + \llbracket \vec{b}[d + 1, 2r' + d] \rrbracket_0^{(3)} + \llbracket \vec{y} \rrbracket_0^{(3)} \odot \llbracket \vec{b}[d + 1, 2r' + d] \rrbracket_0^{(3)}$ via $\mathcal{F}_{\text{VOLE-ZK}}$, and put $\llbracket \vec{y} \rrbracket_0^{(2)} := \vec{f} + \llbracket \vec{b}[d + 1, 2r' + d] \rrbracket_0^{(2)}$.
- Let $\llbracket y_i \rrbracket_0^{(2r)} \leftarrow \text{elt}(\llbracket y_i \rrbracket_1^{(2)})$ be the encoded inputs $y_i \in \mathbb{F}_{2^r}$ of the receiver $\mathcal{P}_0$ and denote $\vec{y} := (y^i)_{i \in [n]}$.

• Key-Multiplication:

- $\mathcal{P}_0$ commits to $\llbracket k_0 \odot \vec{y} \rrbracket_0^{(2r)} \leftarrow \Pi_{\text{Commit}}(\langle k_0 \odot \vec{y}, 2, r \rangle, (\Delta_1^{(2)}, 2, r))$ and opens $\llbracket k_0 \odot \vec{y} \rrbracket_0^{(2r)} - \llbracket k_0 \rrbracket_0^{(2r)} \odot \llbracket \vec{y} \rrbracket_0^{(2r)}$ to $\mathcal{P}_1$ via $\mathcal{F}_{\text{VOLE-ZK}}$, who aborts if the opening is non-zero.
- The parties send $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \llbracket \vec{y} \rrbracket_0^{(2r)})$ to $\mathcal{F}_{\text{daBits}}^{r,t}$ and receive $\langle k_1 \odot \vec{y} \rangle^{(2r)} \leftarrow (\llbracket \vec{y} \rrbracket_0^{(2r)}, \llbracket \vec{w} \rrbracket_1^{(2r)})$.
- Let $\langle k \odot \vec{y} \rangle^{(2r)} \leftarrow (\llbracket \vec{y} \rrbracket_0^{(2r)} + \llbracket k_0 \odot \vec{y} \rrbracket_0^{(2r)}, \llbracket \vec{w} \rrbracket_1^{(2r)})$

• Modulus Conversion and Matrix Multiplication

- Compute $\langle \vec{c} \rangle^{(2r)} \leftarrow \text{elt}(\langle \vec{c} \rangle^{(2)})$ and open $\langle \vec{g} \rangle^{(2r)} \leftarrow \langle k \odot \vec{y} \rangle^{(2r)} + \langle \vec{c} \rangle^{(2r)}$ to both parties via $\mathcal{F}_{\text{VOLE-ZK}}$.
- Compute the modulo 3 sharing as $\langle \text{vec}(k \odot \vec{y}) \rangle^{(3)} \leftarrow \text{vec}(\vec{g}) + \langle \vec{c} \rangle^{(3)} + \text{vec}(\vec{g}) \odot \langle \vec{c} \rangle^{(3)}$.
- Compute the secret-shared PRF evaluation as $\langle F_k(\vec{x}) \rangle^{(3)} := B \cdot \langle \text{vec}(k \odot \vec{y}) \rangle^{(3)}$ and open $F_k(\vec{x})$ to $\mathcal{P}_1$ via $\mathcal{F}_{\text{VOLE-ZK}}$.
- Let $(\llbracket \vec{F}_0 \rrbracket_0^{(3)}, \llbracket \vec{F}_1 \rrbracket_1^{(3)})$ denote the parties' shares of $\langle F_k(\vec{x}) \rangle^{(3)}$, then $\llbracket F_k(\vec{x}) \rrbracket_1^{(3)} := \llbracket \vec{F}_1 \rrbracket_1^{(3)} + \vec{F}_0$ gives an authentication of $F_k(\vec{x})$ after the opening to $\mathcal{P}_1$.

Shuffle and Prove:

- $\mathcal{P}_1$ sends $(F_k(x_{\pi(i)}))_{i \in [n]}$ to $\mathcal{P}_0$. The parties compute $\tilde{F}_k(x_{\pi(i)}) := \text{elt}(F_k(x_{\pi(i)}))$ and $\llbracket \tilde{F}_k(x_i) \rrbracket_1^{(3t)} \leftarrow \text{elt}(\llbracket F_k(x_i) \rrbracket_1^{(3)})$.
- The parties invoke Π_{Shuffle} (Figure 14) on the committed input vector $(\llbracket \tilde{F}_k(x_i) \rrbracket_1^{(3t)})_{i \in [n]}$ and shuffled vector $(\tilde{F}_k(x_{\pi(i)}))_{i \in [n]}$, where $\mathcal{P}_1$ proves knowledge of the permutation π . The parties abort if the subprotocol aborts, and otherwise obtain $\llbracket \vec{\pi} \rrbracket^{(3t)}$.
- $\mathcal{P}_0$ outputs $\pi(F_k(\vec{x}))$ and $\mathcal{P}_1$ outputs $F_k(\vec{x})$. The parties also output commitments $\llbracket \vec{x} \rrbracket_0^{(3t)}$ and $\llbracket \vec{\pi} \rrbracket_1^{(3t)}$.

Figure 7: Protocol for Dark Matter-based Shuffled and Distributed OPRF

psi.git. We reimplement in Rust the VOLE protocol from Wolverine [71], which is based on the learning parity with noise (LPN) assumption [14]. The VOLE-based commitment scheme is implemented based on QuickSilver [72], with reference to the emp-toolkit [70] cryptographic suite. To generate authenticated multiplication triples, we modify the implementation of [73] and get

BDOZ-style [11] authenticated secret sharing. For DY-PRF, we use the FourQ curve¹¹, which has a Rust bind¹² from the FourQlib library. The curve targets a conjectured 128-bit security level, with a

¹¹We use the FourQ curve for a fair comparison with [73].

¹²<https://crates.io/crates/fourq>

Set size	Protocol	Offline		Online				Total Comp. (s)
		Comm. (MB)	Comp. (s)	Comm. (MB)	LAN (s)	200 Mbps (s)	50 Mbps (s)	
Two-sided-PSI-Cardinality								
2^{16}	[54]	0	0	56.6	4,430	4,432	4,439	4,430
	[73]	3,784	1,462	29.8	14.2	25.9	28.1	1,476
	Ours	0	0	114	14.8	41.0	50.7	14.8 ^{100×}
2^{18}	[54]	0	0	222	17,720	17,728	17,756	17,720
	[73]	15,184	25,536	119	57.2	75.8	87.4	25,593
	Ours	0	0	267	36.5 ^{1.5×}	73.4 ^{1.03×}	98.4	36.5 ^{485×}
2^{20}	[54]	0	0	873	71,166	71,201	71,306	71,166
	[73]	60,919	398,582	477	231	269	326	398,813
	Ours	0	0	877	116 ^{2×}	198 ^{1.35×}	278 ^{1.17×}	116 ^{613×}
Circuit-PSI								
2^{16}	[73]	3,784	1,462	29.8	14.2	25.9	28.1	1,476
	Ours	4,074	98.1	129	15.6	45.1	55.7	114 ^{13×}
2^{18}	[73]	15,184	25,536	119	57.2	75.8	87.4	25,593
	Ours	16,291	230	326	38.7 ^{1.48×}	81.2	112	269 ^{95×}
2^{20}	[73]	60,919	398,582	477	231	269	326	398,813
	Ours	65,164	754	1,114	129 ^{1.8×}	228 ^{1.18×}	332	883 ^{451×}
Two-sided-PSI-Sum								
2^{16}	[54]	0	0	56.6	4,430	4,432	4,439	4,430
	[73]	3,784	2,193	34.0	14.2	25.9	28.1	2,227
	Ours	4,074	98.1	147	17.9	50.9	63.5	116 ^{19×}
2^{18}	[54]	0	0	222	17,720	17,728	17,756	17,720
	[73]	15,184	38,304	136	57.2	75.8	87.4	38,361
	Ours	16,291	230	350	40.4 ^{1.41×}	87.6	121	270 ^{65×}
2^{20}	[54]	0	0	873	71,166	71,201	71,306	71,166
	[73]	60,919	597,873	544	231	269	326	598,417
	Ours	65,164	754	1,164	133 ^{1.7×}	234 ^{1.15×}	346	887 ^{80×}
Two-sided-PSU								
2^{16}	[61]	0	1.02	25.1	16.9	29.0	29.1	17.9
	Ours	8,146	140	148	15.9 ^{1.06×}	48.1	61.1	156
2^{18}	[61]	0	4.07	100	68.2	117	117	72.3
	Ours	32,583	401	401	40.1 ^{1.7×}	88.3 ^{1.33×}	127	441
2^{20}	[61]	0	16.5	401	274	460	472	291
	Ours	130,328	1,521	1,419	140 ^{1.9×}	260 ^{1.7×}	395 ^{1.19×}	1,661

Table 1: Performance of various Private Set Operations following our Shuffled Distributed OPRF framework. Cells in green indicate the lowest communication cost or fastest running time for that setting. Our speedup factors are highlighted in green.

large prime subgroup order of 246 bits¹³. We use this prime order as the prime field in our implementation.

Fast malicious PSI-Cardinality without offline phase. For set size of one million, our PSI-Cardinality protocol runs for less than 2 minutes on LAN, outperforming previous works with total running time over 19 hours by 613×. PSI-Cardinality is obtained by running SH-DOPRF on both sides’ inputs followed by local matching, so its performance reflects the efficiency of our SH-DOPRF protocol. Note that our SH-DOPRF protocol relies entirely on VOLE-commitment and VOLE-ZK, and does not require any authenticated triples for multiplication, so both SH-DOPRF and PSI-Cardinality require no offline phase. In contrast, Yang et al. [73] requires an expensive offline phase (see Appendix C for details). Even compared to their online phase alone, our end-to-end time is still 2× faster.

Fast total running time for Circuit-PSI and PSI-Sum. Running SH-DOPRF on both sides’ inputs allows both parties to obtain the resulting indicator vector, which can be multiplied with secret-shared inputs to obtain Circuit-PSI and PSI-Sum. As a result, the cost for these two primitives is similar to PSI-Cardinality, with only one set of multiplications in the end that requires authenticated triples. The cost of preparing triples is pushed to the offline phase, leaving fast online LAN time (1.7 – 1.9× compared to the previous art). Since the offline phase consists solely of triple generation, the total running time of our protocol remains low, requiring less than 15 minutes for one million elements, yielding speedups of 451× in Circuit-PSI and 80× in PSI-Sum.

Fast online running time for PSU. Our PSU protocol is 1.19–1.9× faster for one million elements compared to [61] in all three network settings. The offline phase relies solely on triple generation, requiring two authenticated triples per element. Recent advances in triple generation such as the pseudorandom correlation generator

¹³ $N = 0x29c9c14e5e0a72f05397829c9c14e5dfbd004dfe0f79992fb2540ec7768ce7$

(PCG)-based approach [18] can largely reduce this cost, and we estimate based on [18] that an efficient implementation would bring the cost for generating one million triples down to 160 seconds with sub-linear communication cost. We further note that the multiplication in this protocol is of a special form, where one input is a field element and the other input is a bit (embedded into the large field). Developing optimizations specific to such field-bit multiplications in the malicious setting (one mentioned for PSI-Sum, Appendix L) is an interesting direction for future work.

6 Acknowledgements

Lisa Kohl was supported by a NWO (Dutch Research Council) grant with number VI.Veni.222.348 and by NWO Gravitation project QSC with number 024.003.037. Aron van Baarsen and Peter Scholl have been supported by funding from *Agentur für Innovation in der Cybersicherheit GmbH* under the project “Multi-Party Computation in the Confidential Cloud (MPCC)” within the Encrypted Computing – Analysis & Processing of Encrypted Data (EC2) Program. Peihan Miao and Phuoc Van Long Pham were supported in part by NSF CAREER Award 2442384, Meta Research Award, Google Research Scholar Award, and Amazon Research Award.

Part of this work was conducted while Goel, Kohl, Miao, Pham and Scholl were visiting the Simons Institute for the Theory of Computing.

References

- [1] Amit Agarwal, Carsten Baum, Lennart Braun, and Peter Scholl. 2025. Low-Bandwidth Mixed Arithmetic in VOLE-Based ZK from Low-Degree PRGs (*LNCS*). Springer, Cham, 396–426. doi:10.1007/978-3-031-91134-7_14
- [2] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. 2024. Improved Alternating-Moduli PRFs and Post-quantum Signatures. In *CRYPTO 2024, Part VIII (LNCS, Vol. 14927)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 274–308. doi:10.1007/978-3-031-68397-8_9
- [3] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. 2024. Crypto Dark Matter on the Torus - Oblivious PRFs from Shallow PRFs and TFHE. In *EUROCRYPT 2024, Part VI (LNCS, Vol. 14656)*, Marc Joye and Gregor Leander (Eds.). Springer, Cham, 447–476. doi:10.1007/978-3-031-58751-1_16
- [4] Diego F. Aranha, Aron van Baarsen, Adam Blatchley Hansen, Kent Nielsen, and Peter Scholl. 2026. A Maliciously-Secure Post-Quantum OPRF from Crypto Dark Matter. Cryptology ePrint Archive, Paper 2026/533. <https://eprint.iacr.org/2026/533>
- [5] Diego F. Aranha, Aron van Baarsen, and Peter Scholl. 2026. Cryptanalysis of the Weak-to-Strong PRF Transformation in the Alternating Moduli Paradigm (to appear). Cryptology ePrint Archive, Paper 2026/XXXX. <https://eprint.iacr.org/2026/XXXX>
- [6] Irati Manterola Ayala and Håvard Raddum. 2025. Zeroed Out: Cryptanalysis of Weak PRFs in Alternating Moduli. *IACR Trans. Symm. Cryptol.* 2025, 2 (2025), 1–15. doi:10.46586/tosc.v2025.i2.1-15
- [7] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. 2024. One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures. In *ASIACRYPT 2024, Part I (LNCS, Vol. 15484)*, Kai-Min Chung and Yu Sasaki (Eds.). Springer, Singapore, 463–493. doi:10.1007/978-981-96-0875-1_15
- [8] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. 2021. Mac’n’Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions. In *CRYPTO 2021, Part IV (LNCS, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, Cham, Virtual Event, 92–122. doi:10.1007/978-3-030-84259-8_4
- [9] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Berlin, Heidelberg, 263–280. doi:10.1007/978-3-642-29011-4_17
- [10] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO ’91 (LNCS, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, Berlin, Heidelberg, 420–432. doi:10.1007/3-540-46766-1_34
- [11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011 (LNCS, Vol. 6632)*, Kenneth G. Paterson (Ed.). Springer, Berlin, Heidelberg, 169–188. doi:10.1007/978-3-642-20465-4_11
- [12] Václav Beneš. 1964. Optimal rearrangeable multistage connecting networks. *Bell system technical journal* 43, 4 (1964), 1641–1656.
- [13] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 301–318. <https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock>
- [14] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. 1994. Cryptographic Primitives Based on Hard Learning Problems. In *CRYPTO ’93 (LNCS, Vol. 773)*, Douglas R. Stinson (Ed.). Springer, Berlin, Heidelberg, 278–291. doi:10.1007/3-540-48329-2_24
- [15] Dan Boneh, Yuval Ishai, Alain Passetegue, Amit Sahai, and David J. Wu. 2018. Exploring Crypto Dark Matter: New Simple PRF Candidates and Their Applications. In *TCC 2018, Part II (LNCS, Vol. 11240)*, Amos Beimel and Stefan Dziembowski (Eds.). Springer, Cham, 699–729. doi:10.1007/978-3-030-03810-6_25
- [16] Ellette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing Vector OLE. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 896–912. doi:10.1145/3243734.3243868
- [17] Ellette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Cham, 489–518. doi:10.1007/978-3-030-26954-8_16
- [18] Ellette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2020. Efficient Pseudorandom Correlation Generators from Ring-LPN. In *CRYPTO 2020, Part II (LNCS, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Cham, 387–416. doi:10.1007/978-3-030-56880-1_14
- [19] Ellette Boyle, Lisa Kohl, and Peter Scholl. 2019. Homomorphic Secret Sharing from Lattices Without FHE. In *EUROCRYPT 2019, Part II (LNCS, Vol. 11477)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Cham, 3–33. doi:10.1007/978-3-030-17656-3_1
- [20] Prasad Buddharapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlasov. 2020. Private Matching for Compute. Cryptology ePrint Archive, Report 2020/599. <https://eprint.iacr.org/2020/599>
- [21] Jan Camenisch and Gregory M. Zaverucha. 2009. Private Intersection of Certified Sets. In *FC 2009 (LNCS, Vol. 5628)*, Roger Dingledine and Philippe Golle (Eds.). Springer, Berlin, Heidelberg, 108–127. doi:10.1007/978-3-642-03549-4_7
- [22] Davide Carnemolla, Dario Catalano, Valentina Frasca, and Emanuele Giunta. 2025. On the (Un)biasability of Existing Verifiable Random Functions. Cryptology ePrint Archive, Paper 2025/2176. <https://eprint.iacr.org/2025/2176>
- [23] Gowri R. Chandran, Thomas Schneider, Maximilian Stillger, and Christian Weinert. 2025. Concretely Efficient Private Set Union via Circuit-Based PSI. In *ASIACCS 25*. ACM Press, 149–162. doi:10.1145/3708821.3710839
- [24] Nishanth Chandran, Divya Gupta, and Akash Shah. 2022. Circuit-PSI With Linear Complexity via Relaxed Batch OPRF. *PoPETs* 2022, 1 (Jan. 2022), 353–372. doi:10.2478/popets-2022-0018
- [25] Melissa Chase and Peihan Miao. 2020. Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Cham, 34–63. doi:10.1007/978-3-030-56877-1_2
- [26] Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. 2024. Private Set Operations from Multi-query Reverse Private Membership Test. In *PKC 2024, Part II (LNCS, Vol. 14603)*, Qiang Tang and Vanessa Teague (Eds.). Springer, Cham, 387–416. doi:10.1007/978-3-031-57725-3_13
- [27] Jung Hee Cheon, Wonhee Cho, Jeong Han Kim, and Jiseung Kim. 2021. Adventures in Crypto Dark Matter: Attacks and Fixes for Weak Pseudorandom Functions. In *PKC 2021, Part II (LNCS, Vol. 12711)*, Juan Garay (Ed.). Springer, Cham, 739–760. doi:10.1007/978-3-030-75248-4_26
- [28] Jung Hee Cheon and Daehyun Jang. 2025. Cryptanalysis on Lightweight Verifiable Homomorphic Encryption. In *ASIACRYPT 2025, Part VII (LNCS)*. Springer, Singapore, 366–397. doi:10.1007/978-981-95-5122-4_12
- [29] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. 2009. Efficient Robust Private Set Intersection. In *ACNS 09 International Conference on Applied Cryptography and Network Security (LNCS, Vol. 5536)*, Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud (Eds.). Springer, Berlin, Heidelberg, 125–142. doi:10.1007/978-3-642-01957-9_8
- [30] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2012. Fast and Private Computation of Cardinality of Set Intersection and Union. In *CANS 12 (LNCS, Vol. 7712)*, Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis (Eds.). Springer, Berlin, Heidelberg, 218–231. doi:10.1007/978-3-642-35404-5_17
- [31] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. 2010. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *ASIACRYPT 2010 (LNCS, Vol. 6477)*, Masayuki Abe (Ed.). Springer, Berlin, Heidelberg, 213–231. doi:10.1007/978-3-642-17373-8_13
- [32] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. 2018. PIR-PSI: Scaling Private Contact Discovery. *PoPETs* 2018, 4 (Oct. 2018), 159–178. doi:10.1515/

- popets-2018-0037
- [33] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. 2021. MPC-Friendly Symmetric Cryptography from Alternating Moduli: Candidates, Protocols, and Applications. In *CRYPTO 2021, Part IV (LNCS, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, Cham, Virtual Event, 517–547. doi:10.1007/978-3-030-84259-8_18
 - [34] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *PKC 2005 (LNCS, Vol. 3386)*, Serge Vaudenay (Ed.). Springer, Berlin, Heidelberg, 416–431. doi:10.1007/978-3-540-30580-4_28
 - [35] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. 2016. Efficient Set Intersection with Simulation-Based Security. *Journal of Cryptology* 29, 1 (Jan. 2016), 115–155. doi:10.1007/s00145-014-9190-0
 - [36] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient Private Matching and Set Intersection. In *EUROCRYPT 2004 (LNCS, Vol. 3027)*, Christian Cachin and Jan Camenisch (Eds.). Springer, Berlin, Heidelberg, 1–19. doi:10.1007/978-3-540-24676-3_1
 - [37] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. 2021. Private Set Operations from Oblivious Switching. In *PKC 2021, Part II (LNCS, Vol. 12711)*, Juan Garay (Ed.). Springer, Cham, 591–617. doi:10.1007/978-3-030-75248-4_21
 - [38] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to Construct Random Functions. *Journal of the ACM* 33, 4 (Oct. 1986), 792–807. doi:10.1145/6490.6503
 - [39] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. doi:10.1145/28395.28420
 - [40] Dov Gordon, Carmit Hazay, Phi Hung Le, and Mingyu Liang. 2022. More Efficient (Reusable) Private Set Union. Cryptology ePrint Archive, Report 2022/713. <https://eprint.iacr.org/2022/713>
 - [41] Carmit Hazay and Yehuda Lindell. 2008. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *TCC 2008 (LNCS, Vol. 4948)*, Ran Canetti (Ed.). Springer, Berlin, Heidelberg, 155–175. doi:10.1007/978-3-540-78524-8_10
 - [42] Carmit Hazay and Kobbi Nissim. 2010. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC 2010 (LNCS, Vol. 6056)*, Phong Q. Nguyen and David Pointcheval (Eds.). Springer, Berlin, Heidelberg, 312–331. doi:10.1007/978-3-642-13013-7_19
 - [43] Kai Hu, Gregor Leander, Håvard Raddum, Arne Sandrić, and Aleksei Udovenko. 2026. Cryptanalysis of Two Alternating Moduli Weak PRFs. Cryptology ePrint Archive, Paper 2026/482. <https://eprint.iacr.org/2026/482>
 - [44] Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS 2012*. The Internet Society.
 - [45] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 2020. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *2020 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, 370–389. doi:10.1109/EuroSP48549.2020.00031
 - [46] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. 2017. Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions. Cryptology ePrint Archive, Report 2017/738. <https://eprint.iacr.org/2017/738>
 - [47] Stanislaw Jarecki and Xiaomin Liu. 2009. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC 2009 (LNCS, Vol. 5444)*, Omer Reingold (Ed.). Springer, Berlin, Heidelberg, 577–594. doi:10.1007/978-3-642-00457-5_34
 - [48] Lea Kissner and Dawn Xiaodong Song. 2005. Privacy-Preserving Set Operations. In *CRYPTO 2005 (LNCS, Vol. 3621)*, Victor Shoup (Ed.). Springer, Berlin, Heidelberg, 241–257. doi:10.1007/11535218_15
 - [49] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 818–829. doi:10.1145/2976749.2978381
 - [50] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. 2019. Scalable Private Set Union from Symmetric-Key Techniques. In *ASIACRYPT 2019, Part II (LNCS, Vol. 11922)*, Steven D. Galbraith and Shihoh Moriai (Eds.). Springer, Cham, 636–666. doi:10.1007/978-3-030-34621-8_23
 - [51] Phi Hung Le, Samuel Ranellucci, and S. Dov Gordon. 2019. Two-party Private Set Intersection with an Untrusted Third Party. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2403–2420. doi:10.1145/3319535.3345661
 - [52] Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. 2021. Private Join and Compute from PIR with Default. In *ASIACRYPT 2021, Part II (LNCS, Vol. 13091)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Cham, 605–634. doi:10.1007/978-3-030-92075-3_21
 - [53] Yehuda Lindell. 2016. How To Simulate It - A Tutorial on the Simulation Proof Technique. Cryptology ePrint Archive, Report 2016/046. <https://eprint.iacr.org/2016/046>
 - [54] Peihian Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. 2020. Two-Sided Malicious Security for Private Intersection-Sum with Cardinality. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Cham, 3–33. doi:10.1007/978-3-030-56877-1_1
 - [55] C. Andrew Neff. 2001. A Verifiable Secret Shuffle and Its Application to e-Voting. In *ACM CCS 2001*, Michael K. Reiter and Pierangela Samarati (Eds.). ACM Press, 116–125. doi:10.1145/501983.502000
 - [56] Torben P. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91 (LNCS, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, Berlin, Heidelberg, 129–140. doi:10.1007/3-540-46766-1_9
 - [57] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Cham, 401–431. doi:10.1007/978-3-030-26954-8_13
 - [58] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2020. PSI from PaXoS: Fast, Malicious Private Set Intersection. In *EUROCRYPT 2020, Part II (LNCS, Vol. 12106)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Cham, 739–767. doi:10.1007/978-3-030-45724-2_25
 - [59] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. 2009. Secure Two-Party Computation Is Practical. In *ASIACRYPT 2009 (LNCS, Vol. 5912)*, Mitsuru Matsui (Ed.). Springer, Berlin, Heidelberg, 250–267. doi:10.1007/978-3-642-10366-7_15
 - [60] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPT 2019, Part III (LNCS, Vol. 11478)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Cham, 122–153. doi:10.1007/978-3-030-17659-4_5
 - [61] Sihang Pu, Jiahui Gao, and Ni Trieu. 2026. Malicious Private Set Union with Two-Sided Output. In *EUROCRYPT*. <https://eprint.iacr.org/2026/204>
 - [62] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 2505–2517. doi:10.1145/3548606.3560658
 - [63] Peter Rindal and Mike Rosulek. 2017. Improved Private Set Intersection Against Malicious Adversaries. In *EUROCRYPT 2017, Part I (LNCS, Vol. 10210)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer, Cham, 235–259. doi:10.1007/978-3-319-56620-7_9
 - [64] Peter Rindal and Mike Rosulek. 2017. Malicious-Secure Private Set Intersection via Dual Execution. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1229–1242. doi:10.1145/3133956.3134044
 - [65] Peter Rindal and Philipp Schoppmann. 2021. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In *EUROCRYPT 2021, Part II (LNCS, Vol. 12697)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Cham, 901–930. doi:10.1007/978-3-030-77886-6_31
 - [66] Antoine Sidem and Qingju Wang. 2025. General Key Recovery Attack on Pointwise-Keyed Functions - Application to Alternating Moduli Weak PRFs. In *ASIACRYPT 2025, Part I (LNCS)*, Springer, Singapore, 131–154. doi:10.1007/978-981-95-5018-0_5
 - [67] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. 2019. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 1556–1571. <https://www.usenix.org/conference/usenixsecurity19/presentation/thomas>
 - [68] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. 2007. Privacy preserving error resilient dna searching through oblivious automata. In *ACM CCS 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM Press, 519–528. doi:10.1145/1315245.1315309
 - [69] Aron van Baarsen and Marc Stevens. 2024. Amortizing Circuit-PSI in the Multiple Sender/Receiver Setting. *CiC 1*, 3 (2024), 2. doi:10.62056/a0fhsgvtw
 - [70] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. <https://github.com/emp-toolkit>
 - [71] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1074–1091. doi:10.1109/SP40001.2021.00056
 - [72] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2986–3001. doi:10.1145/3460120.3484556
 - [73] Yaxi Yang, Xiaojian Liang, Xiangfu Song, Ye Dong, Linting Huang, Hongyu Ren, Changyu Dong, and Jianying Zhou. 2025. Maliciously Secure Circuit Private Set Intersection via SPDZ-Compatible Oblivious PRF. *PopETS 2025*, 2 (April 2025), 680–696. doi:10.56553/popets-2025-0082
 - [74] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th FOCS*. IEEE Computer Society Press, 162–167. doi:10.1109/SFCS.1986.225

- [75] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. 2023. Linear Private Set Union from Multi-Query Reverse Private Membership Test. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 337–354. <https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-cong>

A Open Science

The artifacts of this paper involve the source code for the protocol and the code for generating synthetic data. In accordance with the open source policy, the repository containing the code can be found via the link: <https://anonymous.4open.science/r/circuit-psi-3DC6>.

B Ethical Considerations

This research was conducted to design maliciously secure private set operation protocols. The study is in alignment with ethical guidelines, ensuring responsible practices in methodology, data handling, and reporting. The results are based on theoretical analysis and artificial evaluation, posing no negative effects to any individuals or groups to the best of our knowledge.

C Related Work

In this section, we present relevant works related to maliciously secure Private Set Operations.

Maliciously secure PSI and PSU with one-sided output. There has been a long line of work that has resulted in concretely efficient maliciously secure protocols achieving PSI [21, 29, 31, 35, 41, 47, 48, 58, 62–65] and PSU [40, 42] where only one party receives the output.

Maliciously secure PSO protocols with two-sided output Achieving two-sided output in the malicious setting has been a challenging problem for all private set operations (PSI, PSU, PSI-Sum, Circuit-PSI, etc.). Generic techniques to achieve such functionalities result in concretely inefficient protocols. For example, garbling based approaches [44] would incur overheads proportional to $O(n \log n)$ where n is the set size due to the size of the sorting circuits. Furthermore, they incur multiplicative overhead proportional to the security parameter since they generally operate with bit representations of elements. Miao et al. [54] was the first work to achieve malicious security with two-sided output for PSI-Cardinality and PSI-Sum. Furthermore, they introduced the notion of Shuffled Distributed OPRF used to achieve these functionalities.

Yang et al. [73] then improved on [54] and further achieved maliciously secure Circuit-PSI, which by extension, gave maliciously secure protocols for PSI, PSU, PSI-Cardinality, and PSI-Sum with two sided output. Their construction relies on a secure secret-shared shuffling protocol. The specific parameter choices result in an optimal online phase ($O(n)$ computation cost) but an expensive offline phase ($O(n^2)$ computation cost). At a conceptual level, we replace the expensive distributed shuffling protocol with local shuffling augmented by zero-knowledge proofs. This results in an efficient total computation cost of $O(n)$. Note that a more balanced parameter choice in [73] would incur $O(n \log n)$ computation in both phases, still worse than ours. Another difference is that their SH-DOPRF requires authenticated multiplication triples generated in the offline phase, while our protocol relies entirely on VOLE and therefore requires no offline phase.

A recent work [61] achieved maliciously secure PSU with two-sided output with concretely better efficiency. We remark that there are a few other works [51, 69] achieving maliciously secure Circuit-PSI. However, they either require the help of an untrusted third party or achieve a weaker functionality where the final shares are not authenticated.

D Secure Multi-Party Computation

We follow standard MPC literature and consider the real-ideal paradigm. In the ideal world, there is a trusted third party (TTP) that computes a function f that cannot be corrupted. We denote by $\text{IDEAL}_{f, \text{Sim}(z), I}(\vec{x})$ the joint output of both parties where $\vec{x}$ is the vector of inputs of both parties and Sim is an adversary that corrupts the I^{th} party and receives auxiliary input z . In the real world, parties interact with each other according to some protocol π . We denote by $\text{REAL}_{\pi, \mathcal{A}(z), I}(\vec{x})$ the joint output of both parties where $\vec{x}$ is the vector of inputs of both parties and $\mathcal{A}$ is a malicious adversary that corrupts the I^{th} party and receives auxiliary input z .

Definition D.1 (Security with abort against malicious adversaries [53]). We say that a protocol π computing a two-party functionality f achieves security with abort against malicious adversaries if for every non-uniform PPT adversary $\mathcal{A}$, there exists a non-uniform PPT simulator Sim , such that for every $I \in \{0, 1\}$, for all possible inputs $\vec{x}$, for all auxiliary input z ,

$$\text{IDEAL}_{f, \text{Sim}(z), I}(\vec{x}) \approx \text{REAL}_{\pi, \mathcal{A}(z), I}(\vec{x})$$

E Pseudorandom Function

We recall the definition of pseudorandom functions [38].

Definition E.1 (Pseudorandom Functions). We say that a function $F : \{0, 1\}^\lambda \times \{0, 1\}^n \rightarrow \{0, 1\}^m$ is a pseudorandom function if

- **Efficiency:** Given $K \in \{0, 1\}^\lambda$, $x \in \{0, 1\}^n$ $F(K, x)$ can be computed in time $\text{poly}(\lambda)$
- **Security:** For any PPT (probabilistic polynomial time) adversary $\mathcal{A}$, there is a negligible function $\text{negl}(\cdot)$ such that

$$\left| \Pr[\mathcal{A}^{F(K, \cdot)}(1^\lambda)] - \Pr[\mathcal{A}^{R(\cdot)}(1^\lambda)] \right| \leq \text{negl}(\lambda)$$

where $K \xleftarrow{\$} \{0, 1\}^\lambda$ and R is a randomly chosen function from the set of all functions from $\{0, 1\}^n$ to $\{0, 1\}^m$.

F Degree- d VOLE-based Commitments

As stated in Section 2.6, VOLE-ZK requires the underlying VOLE-based commitments to support certain homomorphic operations. To accommodate this, we require degree- d VOLE commitments which are a generalized notion of the VOLE-commitment presented in Section 2.4.

Generalized Degree- d VOLE Commitments For $x \in \mathbb{F}_q$, where q is a prime, and $i \in \{0, 1\}$, we denote $\llbracket x \rrbracket_{d, i \oplus 1}^{(q)} = ((x, \rho_x(t)), (\Delta, \gamma_x))$ to represent a degree- d VOLE commitment to x . That is, $\mathcal{P}_{i \oplus 1}$ holds a commitment polynomial $\rho_x(t) := \rho_0 + \rho_1 \cdot t + \dots + \rho_d \cdot t^d$, where $\rho_d = x$ and $\rho_0, \dots, \rho_d \in \mathbb{F}_{q^r}$, and $\mathcal{P}_i$ holds the evaluation $\gamma_x := \rho_x(\Delta)$ for a global key $\Delta \in \mathbb{F}_{q^r}$ that is unknown to $\mathcal{P}_{i \oplus 1}$. As in the main body, we generally omit the degree, index, etc. wherever clear.

Homomorphic Properties. It can now be seen that these degree- d VOLE commitments (under the same global key Δ and with the same receiving party $\mathcal{P}_{i \oplus 1}$) support the required local, homomorphic operations (see [1, 7] for details):

- **Add:** $\llbracket x + y \rrbracket_d^{(q)} = \llbracket x \rrbracket_{d_1}^{(q)} + \llbracket y \rrbracket_{d_2}^{(q)}$ defined as $\rho_z(t) := t^{d_2-d_1} \cdot \rho_x(t) + \rho_y(t)$ and $\gamma_z := \Delta^{d_2-d_1} \cdot \gamma_x + \gamma_y$ if (w.l.o.g.) $d_1 \leq d_2$.
- **Affine function:** $\llbracket cx + y \rrbracket_d^{(q)} = c \cdot \llbracket x \rrbracket_d^{(q)} + y$, where $c, y \in \mathbb{F}_q$ are public, defined as $\rho_z(t) := c \cdot \rho_x(t) + t^d \cdot y$ and $\gamma_z := c \cdot \gamma_x + \Delta^d \cdot y$.
- **Multiply:** $\llbracket x \cdot y \rrbracket_{d_1+d_2}^{(q)} = \llbracket x \rrbracket_{d_1}^{(q)} \cdot \llbracket y \rrbracket_{d_2}^{(q)}$, defined as $\rho_z(t) := \rho_x(t) \cdot \rho_y(t)$ and $\gamma_z := \gamma_x \cdot \gamma_y$.

Note that all of the above can easily be extended to commit to values in the extension field $\mathbb{F}_{q^r}$ (i.e., where the global key Δ lives) by committing to $\llbracket \vec{x} \rrbracket_d^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle \vec{x}, q, r \rangle, (\Delta, q, r))$ for $x \in \mathbb{F}_{q^r}$ and computing $\llbracket x \rrbracket_d^{(q^r)} := \text{elt}(\llbracket \vec{x} \rrbracket_d^{(q)})$, which can be obtained through a sequence of additions and affine functions as defined above. We will denote this succinctly as $\llbracket x \rrbracket_d^{q^r} \leftarrow \Pi_{\text{Commit}}(\langle \vec{x}, q^r, 1 \rangle, (\Delta, q^r, 1))$.

G Additional Functionalities

In this section, we formally present some additional ideal functionalities that we use in our protocols.

G.1 Doubly-Authenticated Bits

We use the ideal functionality from [4] for generating local as well as secret-shared (mod 2, mod 3) doubly-authenticated bits in Figure 8.

Ideal (mod 2, mod 3)-daBits functionality $\mathcal{F}_{\text{daBits}}^{r^{(2)}, r^{(3)}}$
The ideal functionality builds over $\mathcal{F}_{\text{VOLE-ZK}}^{q,r}$ (Fig. 3) for parameters $q \in \{2, 3\}$, $r \in \{r^{(2)}, r^{(3)}\}$. We include the modulus q in these commands if it is not clear from the context.
Local daBits: Upon receiving $(\text{sid}, \text{laBits}, \ell, i)$ from $\mathcal{P}_0$ and $\mathcal{P}_1$, with $i \in \{0, 1\}$ and $\ell \in \mathbb{N}$:
• If $\mathcal{P}_i$ is honest, sample $\vec{b} \xleftarrow{\\$} \{0, 1\}^\ell$. Otherwise, receive $\vec{b} \in \{0, 1\}^\ell$ from $\mathcal{A}$. • For each $q \in \{2, 3\}$, interpret $\vec{b}$ as an element of $\mathbb{F}_q^\ell$ and sample an authentication $\llbracket \vec{b} \rrbracket_i^{(q)}$. • Return $(\llbracket \vec{b} \rrbracket_i^{(2)}, \llbracket \vec{b} \rrbracket_i^{(3)})$.
Shared daBits: Upon receiving $(\text{sid}, \text{daBits}, \ell)$ from $\mathcal{P}_0$ and $\mathcal{P}_1$, with $\ell \in \mathbb{N}$:
• Sample $\vec{b} \xleftarrow{\\$} \{0, 1\}^\ell$. • For each $q \in \{2, 3\}$, interpret $\vec{b}$ as an element of $\mathbb{F}_q^\ell$ and sample a secret-sharing $\langle \vec{b} \rangle^{(q)}$. • Return $(\langle \vec{b} \rangle^{(2)}, \langle \vec{b} \rangle^{(3)})$.

Figure 8: Ideal functionality for doubly-authenticated bits (daBits).

G.2 Random Coins

We use a standard ideal functionality for sampling random coins, unpredictable to the parties, detailed in Figure 9.

Random Coins Functionality $\mathcal{F}_{\text{Coins}}$
Upon receiving (sid, ℓ, q) from both parties, where $\ell \in \mathbb{N}$ and q is a prime power, the functionality samples $\chi_i \xleftarrow{\\$} \mathbb{F}_q$ for each $i \in [\ell]$ and returns $(\chi_i)_{i \in [\ell]}$ to both parties.

Figure 9: Ideal functionality for random coins.

H Additional Protocols

H.1 Authenticated VOLE

We give a protocol $\Pi_{\text{aVOLE}}^{q,r}$ in Figure 10, realizing the authenticated VOLE functionality $\mathcal{F}_{\text{aVOLE}}^{q,r}$ from Figure 2 in the $\mathcal{F}_{\text{coins}}, \mathcal{F}_{\text{VOLE-ZK}}^{q,r}$ hybrid model.

H.2 VOLE-ZK Proofs

We recall the protocol from [4] realizing the Prove command from $\mathcal{F}_{\text{VOLE-ZK}}^{q,r}$ (Fig. 3) in the $\mathcal{F}_{\text{sVOLE}}^{q,r}$ -hybrid model in Figure 11. The arithmetic circuits $C_1, \dots, C_m : \mathbb{F}_q^\ell \rightarrow \mathbb{F}_q$ are evaluated on the commitments using the homomorphic properties detailed in Appendix F.

H.3 VOLE-ZK Shuffled Exponentiation Proofs

Here, we specify the detailed protocols for proof of shuffled exponentiation using VOLE-ZK over cyclic groups. First, in Figure 12, we detail the protocol for proving relations concerning public group elements and committed scalars.

THEOREM H.1. *The protocol $\Pi_{\text{VOLE-ZK-G}}^{q,r}$ in Figure 12 securely realizes the functionality $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,r}$ (Figure 4).*

PROOF. When both parties are honest or the verifier is corrupted, correctness and security is straightforward. In particular, the value W sent by an honest prover can already be computed by the verifier, so can be perfectly simulated using its VOLE correlations that were generated by the simulator.

If the prover is corrupted, we simulate the proving stage as follows. Since the simulator knows all the prover's committed values, it can test whether the relations $\prod_j G_{j,k} = H_j$ all hold. If they do, and $\mathcal{P}_i$ additionally sends the correct value of W , the simulator sends an arbitrary guess to $\mathcal{F}_{\text{VOLE-ZK-G}}$, which responds with success. Otherwise, if the relations are correct but $\mathcal{P}_i$ sends some $W' \neq W$, the simulator always sends abort to the functionality. Finally, if any relation is *incorrect*, the simulator extracts a guess $(G, H) \in \mathbb{G}^2$ such that $H = G^\Delta$, as follows. Let $E_j = \prod_k G_{j,k}^{x_{j,k}} H_j^{-1}$ be the error in the j -th statement (so $E_j \neq 1_{\mathbb{G}}$ for at least one j). Let $G = \prod_j E_j^{x_j}$, and $H = W' \cdot W^{-1}$, where W is defined as the value an honest prover would send. The simulator sends the guess (G, H) to the functionality, and aborts if it aborts. It is easy to see that if the verifier would accept in the real protocol then the extracted guess satisfies $H = G^\Delta$ as required. $\square$

Authenticated VOLE Protocol $\Pi_{\text{aVOLE}}^{q,r}$

Input: The parties hold inputs $[\![y]\!]_0^{(q,r)}$ for $y \in \mathbb{F}_q^\ell$ committed under the key corresponding to sid_{MAC} for $\mathcal{F}_{\text{sVOLE}}^{q,r}$.

Initialize: The parties send $(\text{sid}_{\text{MAC}}, \text{init}, i)$ to $\mathcal{F}_{\text{sVOLE}}^{q,r}$ and get $\Delta_i \in \mathbb{F}_{q^r}$ for $i \in \{0, 1\}$. $\mathcal{P}_1$ sends $(\text{sid}_{\text{VOLE}}, \text{init}, 1)$ to $\mathcal{F}_{\text{sVOLE}}^{q,r}$ and gets $k \in \mathbb{F}_{q^r}$. Then the parties run $[\![k]\!]_1^{(q,r)} \leftarrow \Pi_{\text{Commit}} \langle (k, q^r, 1), (\Delta_0, q^r, 1) \rangle$ and output $[\![k]\!]_1^{(q,r)}$.

Evaluate:

- $\mathcal{P}_0$ samples $y_0 \xleftarrow{\$} \mathbb{F}_{q^r}$ and commits to it as $[\![y_0]\!]_0^{(q,r)} \leftarrow \Pi_{\text{Commit}} \langle (y_0, q, r), (\Delta_1, q, r) \rangle$.
- The parties send $(\text{sid}_{\text{VOLE}}, 1, \ell + 1, \text{id}_1, \dots, \text{id}_{\ell+1})$ to $\mathcal{F}_{\text{sVOLE}}^{q,r}$, where $\text{id}_1, \dots, \text{id}_{\ell+1}$ are fresh identifiers. $\mathcal{P}_0$ receives $w'_0 \parallel \vec{w}', u_0 \parallel \vec{u} \in \mathbb{F}_q^{\ell+1}$ and $\mathcal{P}_1$ receives $v'_0 \parallel \vec{v}' \in \mathbb{F}_q^{\ell+1}$, satisfying $(v'_0 \parallel \vec{v}') = (w'_0 \parallel \vec{w}') + k \cdot (u_0 \parallel \vec{u})$. $\mathcal{P}_0$ sends $(z_0 \parallel \vec{z}) := (y_0 \parallel \vec{y}) - (u_0 \parallel \vec{u})$ to $\mathcal{P}_1$. $\mathcal{P}_0$ puts $(w_0 \parallel \vec{w}) := -(w'_0 \parallel \vec{w}')$ and $\mathcal{P}_1$ puts $(v_0 \parallel \vec{v}) := k \cdot (z_0 \parallel \vec{z}) + (v'_0 \parallel \vec{v}')$, i.e., such that $(v_0 \parallel \vec{v}) + (w_0 \parallel \vec{w}) = k \cdot (y_0 \parallel \vec{y})$.
- $\mathcal{P}_0$ does $[\![w_0 \parallel \vec{w}]\!]_0^{(q,r)} \leftarrow \Pi_{\text{Commit}} \langle (w_0 \parallel \vec{w}, q, r), (\Delta_1, q, r) \rangle$, $\mathcal{P}_1$ does $[\![v_0 \parallel \vec{v}]\!]_1^{(q,r)} \leftarrow \Pi_{\text{Commit}} \langle (v_0 \parallel \vec{v}, q, r), (\Delta_0, q, r) \rangle$.
- The parties send (sid, ℓ, q^r) to $\mathcal{F}_{\text{coins}}$ and get $\chi_i \xleftarrow{\$} \mathbb{F}_{q^r}$ for each $i \in [\ell]$.
- The parties open $\tilde{y} \leftarrow [\![y_0]\!]_0^{(q,r)} + \sum_{i=1}^{\ell} \chi_i \cdot [\![y_i]\!]_0^{(q,r)}$ and $\tilde{w} \leftarrow [\![w_0]\!]_0^{(q,r)} + \sum_{i=1}^{\ell} \chi_i \cdot [\![w_i]\!]_0^{(q,r)}$, as well as $\tilde{v} \leftarrow [\![v_0]\!]_1^{(q,r)} + \sum_{i=1}^{\ell} \chi_i \cdot [\![v_i]\!]_1^{(q,r)}$.
- $\mathcal{P}_1$ aborts if $\tilde{v} + \tilde{w} \neq k \cdot \tilde{y}$. Otherwise, they prove to $\mathcal{P}_0$ that $\tilde{v} + \tilde{w} - k \cdot \tilde{y} \cdot [\![k]\!]_1^{(q,r)}$ opens to zero.
- Output $\langle k \cdot \tilde{y} \rangle^{(q,r)} \leftarrow ([\![\tilde{w}]\!]_0^{(q,r)}, [\![\tilde{v}]\!]_1^{(q,r)})$.

Figure 10: Protocol realizing authenticated VOLE.

Proof of Shuffled Exponentiation. In Figure 13, we present the proof of shuffled exponentiation, which was described in Section 3.1.1. Since this protocol does not directly implement an ideal functionality, we do not present a security proof here; this is instead handled via the proof of the distributed, shuffled OPRF protocol.

Proof of Shuffle Over Field. In Figure 14, we present the standard shuffling protocol over finite fields, which we use in our Dark Matter based distributed, shuffled OPRF. This subprotocol uses the standard Noff shuffle technique based on polynomial identity testing [55]. The goal is to prove that the public vector $(z_j)_{j \in [n]}$ is a shuffle of the committed vector $(y_j)_{j \in [n]}$, according to a permutation π known to the prover. The protocol additionally outputs commitments to the permutation, $\{\pi(j)\}_j$.

I Proof of Theorem 3.1

Before describing the simulators, we discuss some subroutines that are used multiple times throughout the proof. We state these subroutines once and the simulator will perform the corresponding steps whenever the subroutine is called. These subroutines correspond to

Protocol Prove($[\![x_1]\!]_{1,i \oplus 1}^{(q)}, \dots, [\![x_\ell]\!]_{1,i \oplus 1}^{(q)}, C_1, \dots, C_m$)

Parameters: Two parties, $\mathcal{P}_0$ and $\mathcal{P}_1$, holding $[\![x_1]\!]_{1,i \oplus 1}^{(q)}, \dots, [\![x_\ell]\!]_{1,i \oplus 1}^{(q)}$, such that $\mathcal{P}_i$ holds the global key $\Delta \in \mathbb{F}_{q^r}$ associated to session sid of $\mathcal{F}_{\text{sVOLE}}^{q,r}$ and $\mathcal{P}_{i \oplus 1}$ holds $x_1, \dots, x_\ell \in \mathbb{F}_q$. Arithmetic circuits $C_1, \dots, C_m : \mathbb{F}_q^\ell \rightarrow \mathbb{F}_q$ of degree $d_1, \dots, d_m \in \mathbb{N}$, respectively. Base field $\mathbb{F}_q$ and extension field $\mathbb{F}_{q^r}$, integers $\ell, m \in \mathbb{N}$ and $d := \max\{d_1, \dots, d_m\}$.

Subprotocol BatchEval($[\![x_1]\!]_{1,i \oplus 1}^{(q)}, \dots, [\![x_\ell]\!]_{1,i \oplus 1}^{(q)}, C_1, \dots, C_m$): The parties evaluate the circuits $C_1, \dots, C_m$ gate-by-gate on the commitments $[\![x_1]\!]_{1,i \oplus 1}^{(q)}, \dots, [\![x_\ell]\!]_{1,i \oplus 1}^{(q)}$ to obtain the outputs $[\![z_1]\!]_{d_1, i \oplus 1}^{(q)}, \dots, [\![z_m]\!]_{d_m, i \oplus 1}^{(q)}$. Here we assume the input commitments have degree 1, but this can readily be extended to higher degrees.

Subprotocol BatchVer($[\![z_1]\!]_{d_1, i \oplus 1}^{(q)}, \dots, [\![z_m]\!]_{d_m, i \oplus 1}^{(q)}$):

- The parties send $(\text{sid}, \text{eval}, r \cdot (d - 1))$ to $\mathcal{F}_{\text{sVOLE}}^{q,r}$, receive $[\![\vec{s}^j]\!]_{1, i \oplus 1}^{(q)}$ with $\vec{s}^j \in \mathbb{F}_q^r$ and $j \in [d - 1]$ and compute $[\![s^j]\!]_{1, i \oplus 1}^{(q,r)} := \text{elt}_\alpha([\![\vec{s}^j]\!]_{1, i \oplus 1}^{(q)})$.
- $\mathcal{P}_i$ sends a challenge $(\chi_1, \dots, \chi_m) \xleftarrow{\$} \mathbb{F}_{q^r}^m$.
- The parties compute $[\![z]\!]_{d, i \oplus 1}^{(q,r)} := \sum_{k=1}^m \chi_k \cdot [\![z_k]\!]_{d_k, i \oplus 1}^{(q)}$. $\mathcal{P}_{i \oplus 1}$ masks the corresponding polynomial as $\pi(t) := \rho_z(t) + \sum_{j=1}^{d-1} t^{j-1} \cdot \rho_{s^j}(t)$ and sends it to $\mathcal{P}_i$.
- $\mathcal{P}_i$ computes $\varepsilon := \gamma_z + \sum_{j=1}^{d-1} \Delta^{j-1} \cdot \gamma_{s^j}$ and checks if $\pi(\Delta) = \varepsilon$ and if π has degree at most $d - 1$, and outputs 1 if all checks succeed, and 0 otherwise.

Figure 11: Protocol realizing the VOLE-ZK prove command from $\mathcal{F}_{\text{VOLE-ZK}}^{q,r}$ (Fig. 3)

the cases that can happen when committing to and opening values (via invocation to the VOLE-ZK functionality).

- **Simulator commits a value to the adversary:** To commit to a vector $\vec{x}$ of length l , the simulator, upon receiving $(\text{sid}, \text{eval}, i, l, \text{id}_1, \dots, \text{id}_l, \vec{v})$ from the adversary, samples random $\vec{u}$ and $\vec{w}$ satisfying $\vec{w} = \vec{v} - \Delta_{\text{adv}} \cdot \vec{u}$ where Δ_{adv} is either Δ_0 or Δ_1 depending upon which party is the adversary. The simulator then sends $\vec{x} - \vec{u}$ to the adversary. Note that, in the proof, we refer the commitment held by the adversary for $\vec{x}$ as $\text{Rand}_{\vec{x}}$ and note that it is known to the simulator.
- **Adversary commits a value to the simulator:** Assume that the adversary wants to commit to a value $\vec{x}$ of length l . The simulator, upon receiving $(\text{sid}, \text{eval}, i, l, \text{id}_1, \dots, \text{id}_l, \vec{u})$ from the adversary, samples random $\vec{x}$ and sends it to the adversary. Upon further receiving $\vec{x} - \vec{u}$ from the adversary, the simulator extracts $\vec{x}$ since it knows $\vec{u}$ via the previous call. The simulator stores $\vec{x}$ along with $(\vec{u}, \vec{w})$ that we refer to as $\text{Rand}_{\vec{x}}$ again, abusing notation.
- **Simulator opens a value/ gives a ZK proof to the adversary:** Note that, as mentioned in the main body, opening a committed value can be seen as sending the value along with a ZK proof of consistency with the commitment. The simulator can give

Protocol $\Pi_{\text{VOLE-ZK-G}}^{q,r}$ for VOLE-ZK Group Operations

Prove Group Relations: Given a batch of group elements $\{H_j, G_{j,1}, \dots, G_{j,\ell}\}$ and authenticated values $\{\llbracket x_{j,1} \rrbracket^{(q)}, \dots, \llbracket x_{j,\ell} \rrbracket^{(q)}\}$, for $j \in [m]$, the prover $\mathcal{P}_i$ shows that $G_{j,1}^{x_{j,1}} \dots G_{j,\ell}^{x_{j,\ell}} = H_j, \forall j \in [m]$ as follows:

- (1) $P_{i \oplus 1}$ sends a random challenge $(\chi_1, \dots, \chi_m) \leftarrow \mathbb{Z}_q^m$
- (2) Write $\llbracket x_{j,k} \rrbracket^{(q)} = ((x_{j,k}, w_{j,k}), (\Delta, v_{j,k}))$, where P_i holds $x_{j,k}, w_{j,k}$ and $P_{i \oplus 1}$ holds $\Delta, v_{j,k} = w_{j,k} + x_{j,k} \cdot \Delta$.
- (3) P_i sends

$$W = \prod_{j=1}^m \left(\prod_{k=1}^{\ell} G_{j,k}^{w_{j,k}} \right)^{\chi_j}.$$

- (4) $P_{i \oplus 1}$ checks if

$$W \stackrel{?}{=} \prod_{j=1}^m \left(H_j^{-\Delta} \cdot \prod_{k=1}^{\ell} G_{j,k}^{v_{j,k}} \right)^{\chi_j}$$

and aborts if the check fails. Otherwise, accept.

Figure 12: Protocol for proving VOLE-ZK group operations

fake ZK proof via the following approach: The simulator upon receiving the values associated with the commitment $\text{Rand}_{\vec{x}}$ from the adversary, verifies if they are consistent with the ones sent earlier during committing. If not, it sends abort. Else, it sends success.

- **Adversary opens a value/ gives a ZK proof to the simulator:** Similar to the prior approach, the simulator upon receiving the values associated with the commitment $\text{Rand}_{\vec{x}}$ from the adversary, verifies if they are consistent with the ones sent earlier during committing. If not, it sends abort. Else, it sends success if the relation holds (Note that the simulator holds all values in the clear and can thus, check the relation).

We now give a formal description of the simulators and corresponding hybrids to show security for each case depending upon the party the adversary corrupts.

Corrupt $\mathcal{P}_0$: Consider the simulator Sim_0 :

- **Initialize:**

- (1) Upon receiving $(\text{sid}_{\text{MAC}}, \text{init}, 0)$ and Δ_0 from $\mathcal{P}_0$, Sim_0 sends back Δ_0 . Sim_0 sends $(\text{init}, \text{sid}, \Delta_0)$ to the ideal functionality $\mathcal{F}_{\text{SH-DOPRF}}$.
- (2) When $\mathcal{P}_0$ commits to k_0 , Sim_0 extracts k_0 . Let the values corresponding to the commitment for k_0 held by the adversary be Rand_{k_0} .

- **Evaluate:**

- (1)
 - (a) When $\mathcal{P}_0$ commits to $\vec{x}$, Sim_0 extracts $\vec{x}$. Sim_0 sends $(\text{eval}, \text{sid}, \vec{x})$ to the ideal functionality $\mathcal{F}_{\text{SH-DOPRF}}$ and gets back $(\pi(F_k(\vec{x})), [\vec{\pi}]_1, \langle \vec{x} \rangle_0)$. In this particular case, instead of forwarding the commitment value $\text{Rand}_{\vec{x}}$, the simulator sends the value $\langle \vec{x} \rangle_0$ to the adversary.
 - (b) When $\mathcal{P}_0$ commits to $\vec{r}$, Sim_0 extracts $\vec{r}$.

Shuffled Exponentiation Proof

This subprotocol runs in the $\mathcal{F}_{\text{VOLE-ZK-G}}$ -hybrid model. We denote $\mathcal{P}$ as the proving party and $\mathcal{V}$ the verifier, who holds the MAC key Δ .

Input: Public vectors of group elements $\vec{G}, \vec{H} \in \mathbb{G}^n$. Committed scalars $\llbracket x_1 \rrbracket_1^{(q)}, \dots, \llbracket x_n \rrbracket_1^{(q)}$, where $x_j \in \mathbb{F}_q$ is known to the prover, $\mathcal{P}$, who additionally knows a permutation π such that $H_i = G_{\pi(i)}^{x_{\pi(i)}}$.

Protocol $\Pi_{\text{ShuffleExp}}$:

- (1) For each $i \in [n]$, $\mathcal{P}$ commits to the permutation as $\llbracket \pi(i) \rrbracket_1^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle \pi(i), q, 1 \rangle, (\Delta, q, 1))$.
- (2) $\mathcal{V}$ samples and sends $c \xleftarrow{\$} \mathbb{F}_q$ to $\mathcal{P}$.
- (3) $\mathcal{P}$ locally computes $\tilde{y}$, where for each $i \in [n]$, $y_i = c^i$. Let $\llbracket y_{\pi(i)} \rrbracket_1^{(q)} \leftarrow \Pi_{\text{Commit}}(\langle y_{\pi(i)}, q, 1 \rangle, (\Delta_0, q, 1))$.
- (4) $\mathcal{V}$ samples and sends $(c_0, c_1) \xleftarrow{\$} \mathbb{F}_q^2$ to $\mathcal{P}$.
- (5) The parties prove, by invoking the prove command of $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,1}$, that:

$$\prod_{i \in [n]} (c_0 + i + c_1 \cdot c^i) = \prod_{i \in [n]} (c_0 + \llbracket \pi(i) \rrbracket_1^{(q)} + c_1 \cdot \llbracket y_{\pi(i)} \rrbracket_1^{(q)}),$$

and abort if the proof fails.

- (6) The parties prove, by invoking the prove-group command of $\mathcal{F}_{\text{VOLE-ZK-G}}^{q,1}$, that

$$\left(\prod_{i \in [n]} H_i \llbracket y_{\pi(i)} \rrbracket_1^{(q)} \right) = \left(\prod_{i \in [n]} G_i^{c^i \cdot \llbracket x_i \rrbracket_1^{(q)}} \right),$$

and abort if the proof fails.

Figure 13: Protocol for shuffled proof of exponentiation

- (c) When $\mathcal{P}_0$ commits to $\vec{a}_0$, Sim_0 extracts $\vec{a}_0$. Upon further receiving $(\text{sid}_{\text{MAC}}, \text{prove}, (\llbracket \vec{x} \rrbracket_0^q, \llbracket \vec{r} \rrbracket_0^q, \llbracket k_0 \rrbracket_0^q, \llbracket \vec{a}_0 \rrbracket_0^q), C)$ from $\mathcal{P}_0$, Sim_0 checks if the values are consistent with the prior extracted values and the random values assigned. If not, Sim_0 sends back abort. Else, it sends success.
- (d) Upon receiving $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{Random}_{\vec{r}})$ from $\mathcal{P}_0$, Sim_0 sends back $(\vec{r}_0, \text{Random}_{k_1}, \text{Random}_{\vec{r}_1})$.
- (e) Upon receiving $\vec{a}_0 + \vec{r}_0$ along with the associated opening proof value, Sim_0 checks consistency with prior values and if the associated opening proof values are the same as the random values $\text{Rand}_{\vec{r}}$ assigned prior. If either does not match, Sim_0 sends back abort.
- (2)
 - (a) Upon receiving $\vec{r}$ and $(\text{sid}_{\text{MAC}}, \text{prove}, (\llbracket \vec{r} \rrbracket_0^q), C)$ from $\mathcal{P}_0$, Sim_0 checks if for each $i \in [n]$, $R_i = G^{r_i}$ and if the associated committing values $\llbracket \vec{r} \rrbracket_0^q$ are consistent with the same random values $\text{Rand}_{\vec{r}}$. If either check fails, Sim_0 sends back abort. Else, it sends success.
 - (b) The simulator commits to a random $\vec{a}'$ and gives a fake opening proof that $\vec{a}' \odot \vec{a} - 1$ equals 0.

VOLE-ZK Field Shuffle Proof

This subprotocol runs in the $\mathcal{F}_{\text{VOLE-ZK}}$ -hybrid model over the field $\mathbb{F}_{q^r}$. We denote by $\mathcal{P}$ the proving party and by $\mathcal{V}$ the verifier, who holds the MAC key Δ .

Input: The verifier and prover hold commitments $\llbracket y_j \rrbracket_1^{(q^r)}$ to field elements $y_j \in \mathbb{F}_{q^r}$, as well as public field elements $z_j = y_{\pi(j)}$, for $j \in [n]$. The prover additionally holds the permutation $\pi : [n] \rightarrow [n]$.

Protocol Π_{Shuffle} :

- (1) $\mathcal{P}$ commits to their permutation by committing to the vector $\vec{\pi} := (\pi(1), \dots, \pi(n))$, with entries interpreted as elements of $\mathbb{F}_{q^r}$:

$$\llbracket \vec{\pi} \rrbracket_1^{(q^r)} \leftarrow \Pi_{\text{Commit}}((\vec{\pi}, q^r, 1), (\Delta_0^{(q)}, q^r, 1)).$$

- (2) $\mathcal{V}$ samples $\rho, \sigma \xleftarrow{\$} \mathbb{F}_{q^r}$ and sends (ρ, σ) to $\mathcal{P}$.

- (3) The parties compute and open the committed value

$$\prod_{j=1}^n \left(\llbracket y_j \rrbracket_1^{(q^r)} + \rho \cdot j - \sigma \right) - \prod_{j=1}^n \left(z_j + \rho \cdot \llbracket \pi(j) \rrbracket_1^{(q^r)} - \sigma \right).$$

- (4) The parties abort if the opening is non-zero. Otherwise, they accept and output $\llbracket \vec{\pi} \rrbracket_1^{(q^r)}$.

Figure 14: VOLE-ZK proof that a vector of field elements is a permutation of a committed vector.

- (3)
 - (a) Sim_0 forwards the shuffled OPRF values $\pi(F_k(\vec{x}))$ and the commitments to the permutation $\llbracket \vec{\pi} \rrbracket_1$ to $\mathcal{P}_0$.
 - (b) Sim_0 receives c from $\mathcal{P}_0$.
 - (c) Sim_0 commits to a random $\vec{y}_\pi$.
 - (d) Sim_0 receives (c_0, c_1) from $\mathcal{P}_0$.
 - (e) Upon receiving $(\text{sid}_{\text{MAC}}, \text{prove}, (\llbracket \vec{\pi} \rrbracket_1^q, \llbracket \vec{y}_\pi \rrbracket_1^q), C)$ from $\mathcal{P}_0$, Sim_0 gives a fake opening proof.
 - (f) Upon receiving $(\text{sid}_{\text{MAC}}, \text{prove-group}, (\llbracket \vec{x} \rrbracket_1^q, \llbracket \vec{y}_\pi \rrbracket_1^q), C)$ from $\mathcal{P}_0$, Sim_0 gives a fake opening proof.

We now show that the joint output of Sim_0 and protocol output is computationally indistinguishable from the view of $\mathcal{P}_0$ and protocol output. We show this via a sequence of hybrids:

- $\mathcal{H}_0$: Transcript is generated as in the real protocol.
- $\mathcal{H}_1$: We replace the zk proof in step (3)(f) with the fake opening proof as in the simulator.
- $\mathcal{H}_2$: We replace the zk proof in step (3)(e) with the fake opening proof as in the simulator.
- $\mathcal{H}_3$: We replace the commitment to $\vec{y}$ in step (3)(c) with a commitment to a random $\vec{y}$ value as in the simulator. Note that this includes changing the associated $\text{Rand}_{\vec{y}}$ values as well.
- $\mathcal{H}_4$: We replace the real output with the simulated output (output sent by the simulator in steps (1)(a) and (3)(a)).
- $\mathcal{H}_5$: We replace the zk proof in step (2)(b) with the fake opening proof as in the simulator.

- $\mathcal{H}_6$: We replace the commitment to $\vec{E}'$ in step (2)(b) with a commitment to a random $\vec{E}'$ value as in the simulator.
- $\mathcal{H}_7$: We replace the zk proof in step (2)(a) with the correctness checks done by the simulator when an adversary opens a value as described earlier.
- $\mathcal{H}_8$: We replace the opening proof in step (1)(e) with the correctness checks done by the simulator when an adversary opens a value as described earlier.
- $\mathcal{H}_9$: We replace the real call to $\mathcal{F}_{\text{aVOLE}}$ with step (1)(d) in the simulator.
- $\mathcal{H}_{10}$: We replace the zk proof in step (1)(c) with the correctness checks done by the simulator when an adversary opens a value as described earlier. We now arrive at the simulated view.

Now, we argue (computational) indistinguishability between consecutive hybrids. Note that we leave arguments for probability of abort being similar wherever obvious (intuitively, the zk proofs only leak success/abort; commitments are hiding in nature)

- $\mathcal{H}_0 \approx_c \mathcal{H}_1$: Zero knowledge
- $\mathcal{H}_1 \approx_c \mathcal{H}_2$: Zero knowledge
- $\mathcal{H}_2 \approx_c \mathcal{H}_3$: Hiding of commitment scheme
- $\mathcal{H}_3 \approx_c \mathcal{H}_4$: Observe that, here, we simply replace the real output with the simulated output. It is sufficient to note that

$$\begin{aligned} \vec{a} &= \vec{a}_0 + \vec{r}_0 + \vec{r}_1 \\ &= \vec{r} \cdot (\vec{x} + k_0) + k_1 \cdot \vec{r} \\ &= \vec{r} \cdot (\vec{x} + (k_0 + k_1)). \end{aligned}$$

Furthermore,

$$X = R_i^{a_i} = (G^{r_i})^{\frac{1}{a_i}} = G^{\frac{1}{x_i + k_0 + k_1}}$$

Since k_1 and π were chosen uniformly at random unknown to the adversary, the distribution of $\pi(\vec{Y})$ is perfectly indistinguishable from the simulated output. Note that the commitments to $\vec{X}$ and π are also changed accordingly.

- $\mathcal{H}_4 \approx_c \mathcal{H}_5$: Zero knowledge
- $\mathcal{H}_5 \approx_c \mathcal{H}_6$: Hiding of commitment scheme
- $\mathcal{H}_6 \approx_c \mathcal{H}_7$: Soundness and binding of commitment scheme
- $\mathcal{H}_7 \approx_c \mathcal{H}_8$: Soundness and binding of commitment scheme
- $\mathcal{H}_8 \approx_c \mathcal{H}_9$: aVOLE output is random
- $\mathcal{H}_9 \approx_c \mathcal{H}_{10}$: Soundness and binding of commitment scheme
- $\mathcal{H}_i \approx_c \mathcal{H}_{i+1} \forall i \in [0, 3]$: Observe that, in each step, we replace a call to the ZK functionality with a check by the simulator. Since the prover $\mathcal{P}_1$ is honest, the distributions are computationally indistinguishable (Probability of abort is same as view before the call is identical and in case of no abort, both hybrids give same output success).
- $\mathcal{H}_i \approx_c \mathcal{H}_{i+1} \forall i \in [4, 5]$: Observe that, in each step, we replace a call to the ZK functionality with a check by the simulator. If $\mathcal{P}_0$ behaves honestly by providing correct values, then distribution is same. If $\mathcal{P}_0$ provides incorrect values, then either the relation is incorrect on the correct statement (which happens with negligible probability by correctness of the underlying prove command in the VOLE-ZK functionality) or the adversary provided an incorrect statement (which implies that the adversary knows Δ_1 ; since the view so far is

independent of Δ_1 , this guess happens with negligible probability). Thus, in either case, indistinguishability holds with probability $1 - \text{negl}(\cdot)$.

Corrupt $\mathcal{P}_1$: Consider the simulator Sim_1 :

• **Initialize:**

- (1) Upon receiving $(\text{sid}_{\text{MAC}}, \text{init}, 1)$ and Δ_1 from $\mathcal{P}_1$, Sim_1 sends back Δ_1 . Sim_1 sends $(\text{init}, \text{sid}, \Delta_1)$ to the ideal functionality $\mathcal{F}_{\text{SH-DOPRF}}$.
- (2) Upon receiving $(\text{sid}_{\text{PRF}}, \text{init}, 1, k_1)$ from $\mathcal{P}_1$, send back k_1 .
- (3) Sim_1 samples random k_0 and commits to it by sending Rand_{k_0} .

• **Evaluate:**

- (1)(a) Sim_1 queries the ideal functionality to get $[\vec{x}]_0$ and $F_k(\vec{x})$. Sim_1 forwards $[\vec{x}]_0$ to $\mathcal{P}_1$.
 - (b) Sim_1 samples a random vector $\vec{r}$ and commits to it.
 - (c) Sim_1 samples a random vector $\vec{a}_0$ and commits to it.
 - (d) Sim_1 provides a fake opening proof that $\vec{a}_0 - \vec{r} \odot (\vec{x} + k_0)$ is 0.
 - (e) Upon receiving $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{Random}_{\vec{r}})$ from $\mathcal{P}_1$, Sim_1 sends back $(\text{Random}_{k_1}, \text{Random}_{\vec{r}_1})$ to $\mathcal{P}_1$.
 - (f) The simulator opens $\vec{a}_0 + \vec{r}_0$ honestly.
- (2)(a) Sim_1 sends $(F_k(\vec{x}))^{\vec{a}}$ instead of $G^{\vec{r}}$ along with a fake zk proof.
 - (b) When $\mathcal{P}_1$ commits to $\vec{a}'$, Sim_1 extracts $\vec{a}'$.
 - (c) Upon receiving the associated proof values for the relation $\vec{a}' \odot \vec{a} - 1 = 0$, Sim_1 checks consistency with prior values and sends success if the relation is true; abort otherwise.
- (3)(a) When $\mathcal{P}_1$ commits to $\vec{\pi}$, Sim_1 extracts π and sends it to the ideal functionality.
 - (b) Sim_1 samples random c and sends it to $\mathcal{P}_1$.
 - (c) When $\mathcal{P}_1$ commits to $\vec{\gamma}$, Sim_1 extracts $\vec{\gamma}$.
 - (d) Sim_1 samples random (c_0, c_1) and sends it to $\mathcal{P}_1$.
 - (e) Upon receiving $\text{Rand}_{\vec{r}}$ and $\text{Rand}_{\vec{\gamma}}$, Sim_1 checks if these are consistent with the values sent earlier and if the relation

$$\prod_{i \in [n]} (c_0 + i + c_1 \cdot c^i) - \prod_{i \in [n]} (c_0 + \pi(i) + c_1 \cdot \gamma_i)$$

equals 0. If either check fails, Sim_1 sends abort.

- (f) Upon receiving $\text{Rand}_{\vec{a}}$ and $\text{Rand}_{\vec{\gamma}}$, Sim_1 checks if these are consistent with the values sent earlier and if the relation

$$\left(\prod_{i \in [n]} \vec{X}_{\pi(i)}^{Y_i} \right) \cdot \left(\prod_{i \in [n]} G_i^{c^i \cdot a_i} \right)^{-1},$$

equals 1_G . If either check fails, Sim_1 sends abort.

We now show that the joint output of Sim_1 and protocol output is computationally indistinguishable from the view of $\mathcal{P}_1$ and protocol output. We show this via a sequence of hybrids:

- $\mathcal{H}_0$: Transcript is generated as in the real protocol.
- $\mathcal{H}_1$: We replace all proofs given by adversary.
- $\mathcal{H}_2$: We change commitments made with respect to the changed proofs in the previous step.
- $\mathcal{H}_3$: We change $G^{\vec{r}}$ to $(F_k(\vec{x}))^{\vec{a}}$.

- $\mathcal{H}_4$: We change all proofs/openings given by the simulator iteratively.
- $\mathcal{H}_5$: We change all remaining commitments iteratively.
- $\mathcal{H}_6$: We change the final output to the simulated output.

Now,

- $\mathcal{H}_0 \approx_c \mathcal{H}_1$: Soundness of zk proof and binding of commitment scheme
- $\mathcal{H}_1 \approx_c \mathcal{H}_2$: Hiding of commitment scheme
- $\mathcal{H}_2 \approx_c \mathcal{H}_3$: The distributions $G^{\vec{r}}$ and $(F_k(\vec{x}))^{\vec{a}}$ are distributed uniformly.
- $\mathcal{H}_3 \approx_c \mathcal{H}_4$: Zero knowledge
- $\mathcal{H}_4 \approx_c \mathcal{H}_5$: Hiding of commitment schemes
- $\mathcal{H}_5 \approx_c \mathcal{H}_6$: It is easy to see that the output is consistent if no abort happened via correctness of the zk proof.

J Security Proof of Post-Quantum Variant

THEOREM J.1. *The protocol $\Pi_{\text{SH-DOPRF}}^{\text{DM}}$ in Figure 7 realizes the ideal functionality $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ defined in Figure 5 for the strong PRF family in (4) with input-space $X = \mathbb{F}_2^t$ against malicious adversaries as per Definition D.1 in the $\mathcal{F}_{\text{daBits}}^{r,t}$ -hybrid model,¹⁴ assuming $r \geq \lambda$, $t \geq \lambda/\log_2 3$, $2r' + 1 = r$ and $3^t \geq n$.*

PROOF. Since the proof is similar to the proof of the original OPRF construction from [4] on which our construction is based, we focus on providing a sketch of the intuition behind the proof rather than giving a fully detailed description of the simulators.

Correctness: First we show that the protocol is correct in the case where both parties behave honestly. In the first part of the protocol, the parties compute the encoded input $\vec{y} := \text{elt}(\text{bin}(G \odot \vec{x}) \| 1)$ on the receiver's committed inputs $[\vec{x}]_0^{(2)}$. To do so, the parties first convert these to commitments modulo 3 using local doubly-authenticated bits from $\mathcal{F}_{\text{daBits}}^{r,t}$ to obtain $[\vec{x}]_0^{(3)}$. Now multiplication with the public linear code G over $\mathbb{F}_3$ can simply be done locally to obtain $[\vec{z}]_0^{(3)} := G \odot [\vec{x}]_0^{(3)}$. To compute the binary decomposition of these committed values, we note that for $z = z_0 + z_1 \cdot 2 \in \mathbb{F}_3$ with $z_0, z_1 \in \{0, 1\}$ and $z_0 \wedge z_1 = 0$ it holds that

$$2z + 2z^2 \equiv z_1 \pmod{3} \text{ and } z + 2z^2 \equiv z_0 \pmod{3},$$

which allows to evaluate $\vec{y} := \text{bin}(\vec{z})$ on the committed values. Now since $[\vec{y}]_0^{(3)}$ are commitments to $\{0, 1\}$ -valued vectors modulo 3, the parties can again use local daBits to convert these to $[\vec{y}]_0^{(2)}$. Finally, the parties can compute commitments to the encoded inputs $[\vec{y}_i]_0^{(2r)} \leftarrow \text{elt}([\vec{y}_i \| 1]_0^{(2)})$ through linear operations on the commitments.

In the next phase of the protocol, we want the parties to obtain authenticated shares of the product $(k_0 + k_1) \cdot \vec{y}$, where $\vec{y}$ is the vector of encoded inputs $\vec{y} = (y_i)_{i \in [n]}$, authenticated as $[\vec{y}]_0^{(2r)}$, and $k_0, k_1 \in \mathbb{F}_{2^r}$ are the parties PRF key shares, authenticated as $[\![k_0]\!]_0^{(2r)}$ and $[\![k_1]\!]_1^{(2r)}$, respectively. To do this, $\mathcal{P}_0$ can simply commit to $[\![k_0 \odot \vec{y}]\!]_0^{(2r)}$, and prove that it is consistent with the commitments to k_0 and $\vec{y}$. Then to get authenticated shares of $k_1 \odot \vec{y}$, the parties can simply use the authenticated VOLE functionality (Fig. 2, which is embedded in $\mathcal{F}_{\text{daBits}}^{r,t}$). The modulus conversion

¹⁴Recall that $\mathcal{F}_{\text{daBits}}^{r,t}$ adopts $\mathcal{F}_{\text{VOLE-ZK}}^{q,r(q)}$, $\mathcal{F}_{\text{aVOLE}}^{q,r(q)}$ and $\mathcal{F}_{\text{sVOLE}}^{q,r(q)}$ as sub-functionalities for $(q, r^{(q)}) \in \{(2, r), (3, t)\}$ to ensure that consistent global keys are used throughout.

and matrix multiplication step is identical to the original protocol from [4], using doubly authenticated bits for the conversion and performing the matrix multiplication locally.

After this step, $\mathcal{P}_1$ holds the PRF outputs $F_k(\vec{x}) \in \mathbb{F}_3^t$, which are committed to as $\llbracket F_k(\vec{x}) \rrbracket_1^{(3^t)}$. Now $\mathcal{P}_1$ permutes the outputs according to their input permutation $\pi : [n] \rightarrow [n]$, sends them to $\mathcal{P}_0$, and proves that these are consistent with the commitments on the outputs. To do this, $\mathcal{P}_1$ first commits to $\vec{\pi} := (\pi(1), \dots, \pi(n))$ in the extension field $\mathbb{F}_{3^t}$ (i.e., such that the field is large enough to encode the values $\pi(i)$) and the parties convert the commitments on the outputs to the same field as $\llbracket \tilde{F}_k(\vec{x}) \rrbracket_1^{(3^t)} \leftarrow \text{elt}(\llbracket F_k(\vec{x}) \rrbracket_1^{(3^t)})$ to accommodate the shuffling proof. Now the proof of shuffling follows from a standard product argument for set-equality checking [55], where if $\{(\tilde{F}_k(x_{\pi(i)}), \pi(i))_{i \in [n]}\}$ is not equal to $\{(\tilde{F}_k(x_i), i)_{i \in [n]}\}$, then

$$\prod_{i=1}^n (\tilde{F}_k(x_i) + \rho \cdot i - \sigma) - \prod_{i=1}^n (\tilde{F}_k(x_{\pi(i)}) + \rho \cdot \pi(i) - \sigma),$$

is a non-zero polynomial of total degree $\leq n$ in ρ, σ over $\mathbb{F}_3^t$, and so for random $\rho, \sigma \xleftarrow{\$} \mathbb{F}_3^t$ will be equal to zero with negligible probability $\leq n \cdot 3^{-t} \leq n \cdot 2^{-\lambda}$ by the Schwartz-Zippel lemma.

Security against a malicious $\mathcal{P}_0$: For the calls to the ideal $\mathcal{F}_{\text{daBits}}^{r,t}$ functionality (recall that this functionality adopts all the commands from the other VOLE functionalities), the simulator acts exactly as the ideal functionality by letting the adversary choose their outputs (commitment components) in the same way the ideal functionality does. All the while, the simulator keeps track of identifiers id which link the outputs (commitment components) to the inputs (committed values) stored in the ideal functionalities. In particular, when the corrupt $\mathcal{P}_0$ performs arithmetic operations on committed values, the simulator stores the result with some id linked to the values held by $\mathcal{P}_0$, and can in this way check if the openings are performed correctly, aborting if this is not the case. Whenever $\mathcal{P}_0$ calls the Prove command for the $\mathcal{F}_{\text{daBits}}^{r,t}$ functionality for a circuit over $\mathbb{F}_3$, the simulator forwards the adversary's key guesses to the $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ functionality as global-key queries, and continues if they receive success, while aborting otherwise. Note that if the adversary issues key guesses to the $\mathcal{F}_{\text{daBits}}^{r,t}$ functionality for a circuit over $\mathbb{F}_2$, the simulator can simply abort. Since the MAC key $\Delta_1^{(2)} \in \mathbb{F}_{2^r}$ is never output by the $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ functionality, it is never seen by the environment, and the adversary has negligible probability of guessing it correctly. Finally, if the adversary issues a key-guess $k'_1 \in \mathbb{F}_{2^r}$ for $\mathcal{P}_1$'s PRF key share k_1 (associated to sid_{PRF}) to the ideal $\mathcal{F}_{\text{daBits}}^{r,t}$ functionality, the simulator can forward $(\text{sid}_{\text{PRF}}, \text{guess}, k')$ for $k' = k_0 + k'_1$ (where $k_0 \in \mathbb{F}_{2^r}$ is $\mathcal{P}_0$'s extracted key share) to the ideal $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ functionality.

The simulator extracts $\mathcal{P}_0$'s PRF key share $k_0 \in \mathbb{F}_{2^r}$ upon the $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{init}, 2)$ call to $\mathcal{F}_{\text{daBits}}^{r,t}$ and extracts their input vector $\vec{x} \in (\mathbb{F}_2^t)^n$, upon the $\Pi_{\text{Commit}}((\vec{x}, 2, r), (\Delta_1^{(2)}, 2, r))$ calls to $\mathcal{F}_{\text{daBits}}^{r,t}$ by removing the $d - t$ trailing zeros and aborting if the input is not of this form. The simulator sends $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{eval}, \vec{x})$ to $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ and receives $(F_k(x_{\pi(i)}))_{i \in [n]}$. It sends these to $\mathcal{P}_0$ in the same step as $\mathcal{P}_1$ does in the real protocol. Note that the simulator can keep track of $\mathcal{P}_0$'s components of the commitments $\llbracket \vec{x} \rrbracket_0^{(3^t)}$

and $\llbracket \vec{\pi} \rrbracket_1^{(3^t)}$ and sends these to the ideal $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ functionality accordingly.

What remains is to simulate the other communication from $\mathcal{P}_1$ to $\mathcal{P}_0$: In the modulus-conversion step the simulator samples $g_i \xleftarrow{\$} \mathbb{F}_{2^r}$ and sends these to $\mathcal{P}_0$, which is indistinguishable from the real protocol since the secret shared daBits $\vec{c}$ are uniformly random from $\mathcal{P}_0$'s point of view; hence $\vec{c} \leftarrow \text{elt}(\vec{c}) \in \mathbb{F}_{2^r}$ and $g_i := k \cdot y_i + c_i$ are as well.

Security against a malicious $\mathcal{P}_1$: We simulate the underlying ideal functionalities in the same way as described above. $\mathcal{P}_1$'s PRF key share $k_1 \in \mathbb{F}_{2^r}$ can be extracted upon the $(\text{sid}_{\text{PRF}}, \text{init}, 1)$ call to $\mathcal{F}_{\text{daBits}}^{r,t}$ and the simulator extracts the permutation $\vec{\pi}$ upon the corresponding Π_{Commit} call to $\mathcal{F}_{\text{daBits}}^{r,t}$. The simulator sends $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{eval}, \pi)$ to $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ and receives $F_k(\vec{x})$. Recall that the simulator keeps track of $\mathcal{P}_1$'s committed values throughout the protocol, so knows $\vec{F}_1$, and can send $\vec{F}_0 := F_k(\vec{x}) - \vec{F}_1$ to $\mathcal{P}_1$ to simulate the opening of the PRF outputs. The simulator can similarly keep track of $\mathcal{P}_1$'s components of the commitments $\llbracket \vec{x} \rrbracket_0^{(3^t)}$ and $\llbracket \vec{\pi} \rrbracket_1^{(3^t)}$ and sends these to the ideal $\mathcal{F}_{\text{SH-DOPRF}}^{3,t}$ functionality accordingly.

It remains to simulate the other communication from $\mathcal{P}_0$ to $\mathcal{P}_1$. During the input encoding, the simulator samples $\vec{e} \xleftarrow{\$} (\mathbb{F}_2^d)^n$ and $\vec{f} \xleftarrow{\$} (\{0, 1\}^{2r'})^n$ (and sends these as elements of $\mathbb{F}_3^{2r'}$), which are indistinguishable from the real protocol execution since $\mathcal{P}_0$'s local daBits $b[1, d]$ and $b[d + 1, 2r' + d]$ are uniformly random from $\mathcal{P}_1$'s point of view. In the modulus conversion step, the simulator can again sample $\vec{g} \xleftarrow{\$} \mathbb{F}_{2^r}$ since the shared daBits $\vec{c}$ are uniformly random from $\mathcal{P}_1$'s point of view. $\square$

K Multi-Query Reverse Private Membership Test (mqRPMT)

In this section, we present the ideal functionality, our protocol, and proof for mqRPMT introduced in Section 4.1

K.1 Security Proof for mqRPMT

THEOREM K.1. *The protocol Π_{mqRPMT} in Figure 16 securely realizes the mqRPMT functionality $\mathcal{F}_{\text{mqRPMT}}$ defined in Figure 15 against malicious adversaries as per Definition D.1 in the $\mathcal{F}_{\text{SH-DOPRF}}$ -hybrid model.*

PROOF. Since the protocol is relatively straight-forward, we give a concise formal description of the simulators and corresponding hybrids to show security for each case depending upon the party the adversary corrupts.

Corrupt $\mathcal{P}_0$: The simulator Sim_0 receives Δ_0 from the adversary which is sent to the ideal functionality during the Initialize phase. In the Evaluate phase, the simulator receives $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{eval}, \vec{x})$ from the adversary. It sends this vector $\vec{x}$ to the ideal functionality and receives back $|\vec{x} \cap \vec{y}|$, $\llbracket \vec{x} \rrbracket_0$ and $\llbracket \vec{b} \rrbracket_1$. The simulator sends $|X|$ randomly sampled values instead of the permuted PRF values. Upon further receiving $(\text{sid}_{\text{PRF}}, \text{sid}_{\text{MAC}}, \text{eval}, \rho)$ from the adversary, Sim_0 sends back $|\vec{y}| - |\vec{x} \cap \vec{y}|$ new randomly sampled values along with $|\vec{x} \cap \vec{y}|$ random values repeated from the ones previously sent

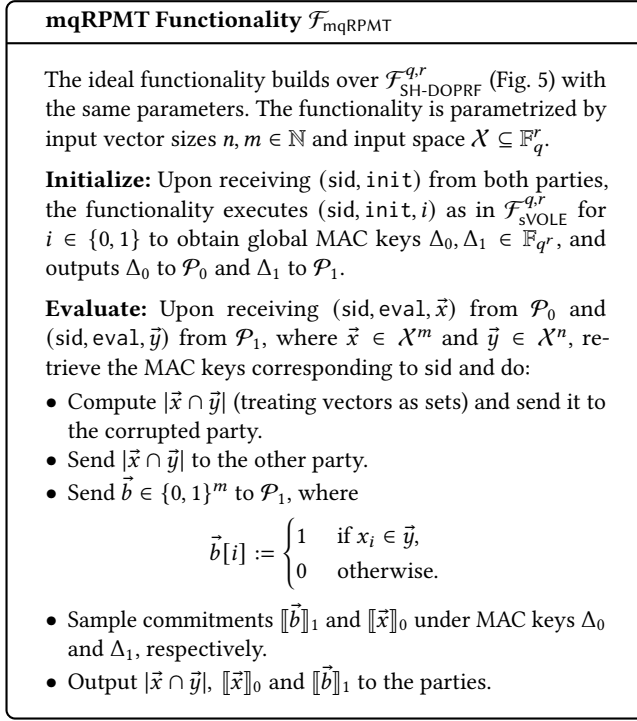


Figure 15: Ideal functionality for mqRPMT

in a random order. This is accompanied by garbage commitments (commitments to random values). Finally, Sim_0 sends $[\vec{b}]_1$ to the adversary and simulates the zk opening proof to always output success conditional on the adversary providing correct commitments.

Hybrid Argument. Starting from the real world, we first change the zk proof relying on the zero knowledge property and binding of commitment schemes. We then replace all garbage commitments relying on the hiding property of commitment schemes. Finally, we change the PRF outputs obtained to random values relying on the fact that the PRF outputs pseudo-randomly distributed values when the key k is unknown (The only thing learned is the number of repeated values which is precisely $|\vec{x} \cap \vec{y}|$ in both the real and ideal world assuming abort did not happen).

Corrupt $\mathcal{P}_1$: The simulator Sim_1 is essentially identical to the previous case since the protocol is almost symmetric with the only difference being that $\mathcal{P}_1$ acts as the prover instead of verifier in the final zk proof. Since the simulator extracts all values apriori, it can check the zk relation in the clear and abort accordingly.

Hybrid Argument. The hybrid proofs are pretty much identical to the previous case with the only difference being that we rely on the soundness of the zk proof and binding of commitment scheme when changing the final proof. $\square$

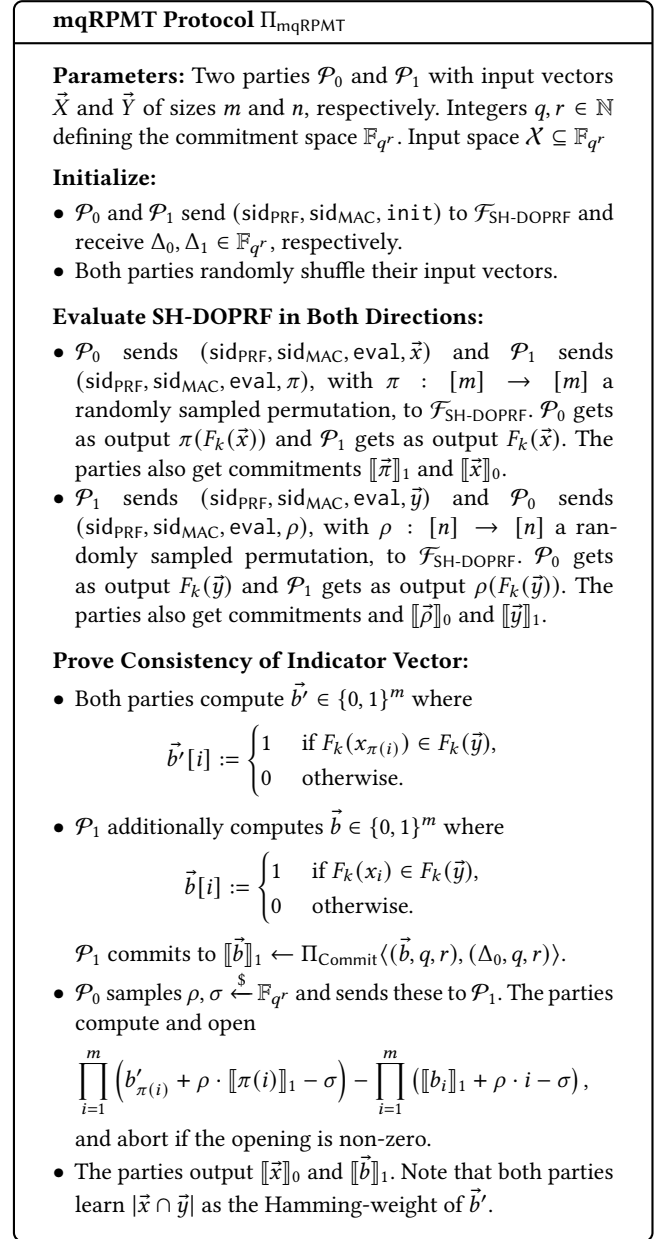


Figure 16: Protocol for mqRPMT

L Private Set Operations

In this section, we provide additional details about our maliciously-secure protocols for other private set operations as discussed in Section 4.2.

PSI-Cardinality: Recall that the functionality and protocol were discussed in Section 4.2.

- *Proof Sketch.* Running SH-DOPRF gives permuted PRF values for both $\vec{x}$ and $\vec{y}$ (apart from commitments, which are hiding in nature). Since the PRF output is pseudorandom, this can be

simulated by sampling random values (where the same random value is present in both PRF outputs for $\vec{x}$ and $\vec{y}$ exactly $|\vec{x} \cap \vec{y}|$ times).

PSU:

- *Functionality.* $\mathcal{P}_0$ holds a set $\vec{x}$ and $\mathcal{P}_1$ holds a set $\vec{y}$. Both parties receive the union of the sets $\vec{x} \cup \vec{y}$ as output.
- *Protocol.* After running our mqRPMT protocol, parties hold commitments to $\vec{x}$ and indicator bit vector $\vec{b}$. Viewing these commitments as authenticated secret shares, parties run a maliciously secure element-wise product protocol between $\vec{x}$ and $(\vec{1} - \vec{b})$. This product vector can then be opened to $\mathcal{P}_1$. Observe that this vector is essentially $\vec{x} \setminus \vec{y}$ when viewed as a set and thus, $\mathcal{P}_1$ can compute $(\vec{x} \setminus \vec{y}) \cup \vec{y} = \vec{x} \cup \vec{y}$ to get the union. Doing this process symmetrically in the other direction allows $\mathcal{P}_0$ to get the output as well (While we do not formalize it as part of the mqRPMT functionality, our protocol for mqRPMT can also output commitments to $\vec{y}$ and the corresponding indicator bit vector with minor modifications. Hence, both outputs (of $\mathcal{P}_0$ and $\mathcal{P}_1$) are consistent with respect to the same vectors $\vec{x}$ and $\vec{y}$).
- *Proof Sketch.* The simulator can extract $\vec{y}$ via the (modified) mqRPMT ideal functionality and use it to compute $\vec{x} \setminus \vec{y} = (\vec{x} \cup \vec{y}) \setminus \vec{y}$ where $\vec{x} \cup \vec{y}$ is given as part of the output and we view vectors as sets. This can then be used to simulate the opening and maliciously secure element-wise product protocol as the authenticated secret shares can just be sampled randomly. Note that the cardinality of the intersection $|\vec{x} \cap \vec{y}|$ is also required to simulate the mqRPMT protocol and can be obtained via $|\vec{x} \cap \vec{y}| = |\vec{x}| - |\vec{x} \setminus \vec{y}|$ where the simulator already has $\vec{x} \setminus \vec{y}$ (as discussed earlier) and $|\vec{x}|$ is just a parameter of the protocol (set size).
- *Remark.* While we do not explicitly mention it in the functionality, in our protocol, parties first obtain the size of the intersection $|\vec{x} \cap \vec{y}|$ followed by the union $\vec{x} \cup \vec{y}$ (if no abort happens). This is better than the standard functionality as parties can decide to open the union only if the intersection size is large enough. This helps avoid issues where a malicious party chooses garbage inputs to learn the set of the other party in the clear (as the intersection size is 0). Our modified protocol can be seen as achieving a new threshold-PSU functionality where the union is revealed only if the intersection size is large enough.

PSI-Sum:

- *Functionality.* $\mathcal{P}_0$ holds two vectors $\vec{x}$ and (a weight vector) $\vec{w}$ and $\mathcal{P}_1$ holds a vector $\vec{y}$. Both parties receive the weight of the intersection $\sum_{x_i \in \vec{y}} w_i$ along with the size of the intersection $|\vec{x} \cap \vec{y}|$ (treating vectors as sets).
- *Protocol.* When running the mqRPMT protocol, $\mathcal{P}_0$ additionally commits to $\vec{w}$, so that the parties obtain $[\vec{w}]_0$. As before, the parties can obtain $|\vec{x} \cap \vec{y}|$ as the weight of $\vec{b}'$. Recall that commitments $[\vec{w}]_0, [\vec{b}]_1$ can be viewed as per-party authenticated secret shares according to BDOZ [11], where $\vec{w}$ is authenticated under Δ_1 and $\vec{b}$ is authenticated under Δ_0 . Hence, the parties can run the BDOZ protocol with inputs $[\vec{w}]_0$ and $[\vec{b}]_1$ to obtain the weight of the intersection as the inner product $\sum_{x_i \in \vec{y}} w_i = \langle \vec{w}, \vec{b} \rangle$.
- *Proof Sketch.* The maliciously secure inner product protocol can be simulated using the weight of the intersection (given as part of the output) and the commitments/ authenticated secret shares

can be replaced by random values (since they are hiding in nature). The mqRPMT protocol can be simulated by sampling a random bit vector $\vec{b}$ conditioned on its hamming weight being equal to $|\vec{x} \cap \vec{y}|$ (given as part of the output).

- *Optimization:* If $\mathcal{X} \subset \mathbb{F}_q$ for a large prime $q \geq 2^{2\sigma}$ (where σ is a statistical security parameter) and if the weight vector $\vec{w}$ has entries from a smaller field $\mathbb{F}_p$ for a prime $p < q$, we can use the following optimization: Choose Δ_i at random from $\{-2^\sigma, \dots, 2^\sigma\}$, i.e., such that $|\Delta_i| \leq 2^\sigma$.¹⁵ Run the mqRPMT protocol over $\mathbb{F}_q$ as before (using $\Delta_i \bmod q$), but commit to the payloads over $\mathbb{F}_p$ using $\Delta_i \bmod p$. Finally, “lift” $[\vec{b}]_1$ (which is given as a commitment over $\mathbb{F}_q$) to $\mathbb{F}_p$, by interpreting the secret shares locally as shares mod p .¹⁶ By the “Lifting Lemma” from [19], the probability that an error occurs in the share conversion can be bounded by $|\Delta_0|/q \leq 2^{-\sigma}$. Note that to enforce that a malicious party $\mathcal{P}_0$ chooses a small Δ_0 , one can use range proofs once at the time of setup. For typical parameters we expect this optimization to result in improved concrete efficiency, as it allows to compute the inner product over the smaller field $\mathbb{F}_p$.

Circuit-PSI: Recall that the functionality and protocol were discussed in Section 4.2.

- *Proof Sketch.* The maliciously secure element-wise product protocol can be simulated using the authenticated secret shares of $\vec{z}$ (given as part of the output for the party acting as the adversary) and the other commitments/authenticated secret shares can be replaced by random values (since they are hiding in nature). The mqRPMT protocol can be simulated by sampling a random bit vector $\vec{b}$ conditioned on its Hamming weight being equal to $|\vec{x} \cap \vec{y}|$ (given as part of the output).
- *Remark.* We note that there are several other variants of Circuit-PSI common in the literature as well (terminology is abused in the literature and the prior functionality and multiple other variants are all referred to as Circuit-PSI). While we achieve several variants, we do not achieve the strongest variant generally termed *Circuit-PSI with doubly associated data* [65] with concrete efficiency. Here, both $\mathcal{P}_0$ and $\mathcal{P}_1$ have as additional input associated data vectors $\vec{A}_x$ and $\vec{B}_y$ respectively. The output $\vec{z}$ satisfies $z_i = \vec{A}_{x,i} \parallel \vec{B}_{y,j}$ if $x_i = y_j$ for some $j \in |\vec{y}|$ and $z_i = 0$ otherwise. We leave designing maliciously secure protocols for this stronger variant of Circuit-PSI as an interesting future work.

M Additional Experimental Details

M.1 Performance of [54]

The paper [54] reports the numbers for one run of Shuffled DOPRF, while the PSI functionality requires two runs of Shuffled DOPRF: one for the sender’s inputs and one for the receiver’s inputs. Hence for PSI-Cardinality, we multiply the reported numbers (communication cost in MB and computation in seconds) by a factor of two. For PSI-SUM, the sender and the receiver exchange two additional sets of ciphertexts (curve prime256v1 elements) along with

¹⁵Note that, in fact, we only need to enforce Δ_0 to be small, as we only need to convert shares of $\vec{b}$, which are authenticated under Δ_0 . However, choosing both Δ_i from this interval still ensures that statistical security is guaranteed via the MAC check.

¹⁶More precisely, we locally lift the shares to shares over the integers, and then take the shares modulo p .

a zero-knowledge proof (sublinear communication). Hence, we keep the an optimistic estimation for computation cost (same as PSI-Cardinality) and add two sets of ciphertexts (32 bytes per ciphertext) for communication cost. For performance in WAN, we estimate based on the reported LAN time and adding time to send data based on bandwidth:

$$\text{WAN\_Time} = \text{LAN\_Time(s)} + \frac{\text{Comm(MB)}}{\text{Bandwidth(Mbps)}} \times 8.$$

M.2 Performance of [73]

In this paper, the number of triples needed in the online phase is equal to the number of elements in one set, with the purpose being the translation from additive to multiplicative shares. This means that the computation cost to prepare triples of this work is similar to the computation cost to prepare triples in our paper (e.g. 1 million triples for 1 million multiplications).

The main bottleneck of the offline phase is the cost to prepare random secret-shared shuffles. In their protocol, the shuffles are used in the following steps: shuffling the sender’s inputs, shuffling the receiver’s inputs, and shuffling the sender’s payloads. For PSI-SUM, the protocol would require preparing three sets of shuffles, while for Circuit-PSI, the set of shuffles for the payload is not required, so the number of shuffles to prepare is two. The offline phase numbers reported in the paper [73] is stated for circuit PSI, hence we multiply the computation cost by a factor of 3/2 (after subtracting the cost of triples) to get the computation cost for PSI-SUM.

Online phase. The computation cost of PSI-Cardinality, Circuit-PSI and PSI-Sum are all roughly similar. The communication cost for PSI-Sum takes an additional two sets of field elements to authenticate the set of payloads and to shuffle the payload. We hence run the online phase for PSI-SUM, then subtract the two sets of elements (32 bytes per element, being the FourQ subgroup order) from the communication cost to obtain the communication cost of PSI-Cardinality and Circuit-PSI.

M.3 Performance of [61]

We modify the code of [61] minimally to add a communication cost tracker (bytes sent/bytes received). The c5a.8xlarge machine is expected to be slower (less modern) than the m7i instance used in [61], hence a slower runtime reported here is expected. We tried our best to turn on every performance optimization flags for the run:

```
CARGO_INCREMENTAL=0 \
CARGO_PROFILE_RELEASE_LTO=fat \
CARGO_PROFILE_RELEASE_CODEGEN_UNITS=1 \
CARGO_PROFILE_BENCH_LTO=fat \
CARGO_PROFILE_BENCH_CODEGEN_UNITS=1 \
RUSTFLAGS='--cfg=curve25519_dalek_backend="simd" \
-C target-cpu=native \
-Z tune-cpu=native' \
N=1000 cargo +nightly test --release \
malicious_psu2_test -- --nocapture --test-threads=1
```