Gold OPRF: Post-Quantum Oblivious Power-Residue PRF

Yibin Yang* Fabrice Benhamouda† Shai Halevi‡ Hugo Krawczyk§ Tal Rabin¶

April 22, 2025

Abstract

We propose plausible post-quantum (PQ) oblivious pseudorandom functions (OPRFs) based on the Power-Residue PRF (Damgård CRYPTO'88), a generalization of the Legendre PRF. For security parameter λ, we consider the PRF $\mathsf{Gold}_k(x)$ that maps an integer x modulo a public prime $p = 2^\lambda \cdot g + 1$ to the element $(k + x)^g \bmod p$, where g is public and $\log g \approx 2\lambda$.

At the core of our constructions are efficient novel methods for evaluating Gold within two-party computation (2PC-Gold), achieving different security requirements. Here, the server $\mathcal{P}_s$ holds the PRF key k whereas the client $\mathcal{P}_c$ holds the PRF input x, and they jointly evaluate Gold in 2PC. 2PC-Gold uses standard Vector Oblivious Linear Evaluation (VOLE) correlations and is *information-theoretic* and *constant-round* in the (V)OLE-hybrid model. We show:

- For a semi-honest $\mathcal{P}_s$ and a malicious $\mathcal{P}_c$: a 2PC-Gold that just uses a single (V)OLE correlation, and has a communication complexity of 3 field elements (2 field elements if we only require a uniformly sampled key) and a computational complexity of $\mathcal{O}(\lambda)$ field operations. We refer to this as *half-malicious* security.
- For malicious $\mathcal{P}_s$ and $\mathcal{P}_c$: a 2PC-Gold that just uses $\frac{\lambda}{4} + \mathcal{O}(1)$ VOLE correlations, and has a communication complexity of $\frac{\lambda}{4} + \mathcal{O}(1)$ field elements and a computational complexity of $\mathcal{O}(\lambda)$ field operations.

These constructions support additional features and extensions, e.g., batched evaluations with better amortized costs where $\mathcal{P}_c$ repeatedly evaluates the PRF under the same key.

Furthermore, we extend 2PC-Gold to Verifiable OPRFs and use the methodology from Beullens et al. (Eurocrypt'25) to get strong OPRF security in the universally composable setting.

All the protocols are efficient in practice. We implemented 2PC-Gold—with (PQ) VOLEs—and benchmarked them. For example, our half-malicious (resp. malicious) n-batched PQ OPRFs incur about 100B (resp. 1.9KB) of amortized communication for $\lambda = 128$.

*Georgia Tech, yyang811@gatech.edu. This work was partially done while the author was at AWS.
†Amazon Web Services, fabrice.benhamouda@gmail.com.
‡Amazon Web Services, shai.halevi@gmail.com.
§Amazon Web Services, hugokraw@gmail.com.
¶Amazon Web Services, talr@seas.upenn.edu.

Contents

1 Introduction

OPRF. *Pseudorandom functions* [GGM86] (PRFs) are essential tools in cryptography. Modern applications often require the evaluation of PRFs in a distributed and privacy-preserving manner. This leads to the notion of *oblivious pseudorandom functions* (OPRFs) [FIPR05, NR97].

Concretely, consider a PRF F, a server $\mathcal{P}_s$ that holds a PRF key k, and a client $\mathcal{P}_c$ that holds a PRF input x. Through the OPRF protocol, $\mathcal{P}_c$ learns the output $F_k(x)$ without obtaining any additional information about k while $\mathcal{P}_s$ learns nothing about x (including nothing about the output $F_k(x)$). Alternatively, an OPRF can be viewed as a specialized instance of *secure two-party computation* (2PC) [Yao86] tailored for evaluating a PRF.

It is important to note that OPRFs are sometimes defined with more stringent requirements than a standard 2PC over a PRF, i.e., they demand additional properties. However, the core component—2PC over PRFs—underlies also the stronger notions and we focus on the design of such a component. We then show how to extend the design to obtain stronger forms of OPRFs.

Post-Quantum OPRF. Currently, the most widely deployed OPRF protocols rely on Diffie-Hellman-type assumptions, with the 2HashDH OPRF [JKK14] being a notable example. While these protocols are lightweight and highly efficient, they are *insecure* against quantum adversaries [Sho94, Sho99]. In light of the potential threats posed by quantum computing, it is imperative to develop OPRF protocols that remain secure in the quantum era.

Over the past five years, numerous proposals for *post-quantum* (PQ) OPRFs have emerged, e.g., [ADDS21, BKW20, Bas23, ADDG24, SHB23, HHM+24, DGH+21, APRR24, FOO23, BDFH24, AG24, ESTX24, HKL+25]. However, a significant efficiency gap remains: these PQ OPRFs are primarily of theoretical interest and are not yet practical for widespread deployment. Addressing this efficiency gap is crucial for practical adoption of PQ OPRFs, and our work focuses on constructing novel PQ OPRFs to bridge this gap.

Challenges. Constructing an efficient PQ OPRF presents two primary and intertwined challenges. First, it requires identifying or designing a suitable PQ PRF. Second, it involves developing methods to efficiently evaluate the chosen PQ PRF within 2PC based on PQ assumptions.

1.1 Our Results

In this work, we aim to bridge the efficiency gap by leveraging the *Power-Residue PRF* [Dam90], a generalization of the Legendre PRF. Let λ denote the security parameter (e.g., $\lambda = 128$, which produces 128-bit outputs, targeting the NIST Security Strength Category 1 for PQ cryptography). To get an $\mathcal{O}(\lambda)$-bit output, we consider the following PRF:

$$F_k(x) := (k + x)^g \bmod p$$

where $p = 2^\lambda \cdot g + 1$ is a *non-secret* prime, $k \in \mathbb{F}_p$ is the key, and $x \in \mathbb{F}_p$ is the input to the PRF.

Crucially, when p is sufficiently large ($\log p \approx 3\lambda$), there are currently *no* known quantum attacks against this PRF.[1]

Gold. We name this PRF Gold since it has a private base and a public exponent, whereas the well-known discrete logarithm has a public base and a private exponent—reversing the order of characters in dlog gives Gold.

[1]There exists an efficient quantum distinguisher if it is allowed to make quantum queries [RS04, vDH00]. In this work, we restrict ourselves to (quantum) attacks with classical queries and justify this choice in Section 2.5.

2PC-Gold. Our **first major contribution** is developing a family of efficient 2PC protocols, called 2PC-Gold, for evaluating the Gold function across various settings between a server $\mathcal{P}_s$ that inputs a key k and a client $\mathcal{P}_c$ with input x. 2PC-Gold protocols leverage standard *Vector Oblivious Linear Evaluation* (VOLE) correlations (recalled in Section 2.6) in a *black-box* manner and are *information-theoretically* secure and *constant-round* in the VOLE-hybrid model.[2] Our 2PC-Gold protocols include:

- A protocol secure against an *unbounded semi-honest* $\mathcal{P}_s$ and an *unbounded malicious* $\mathcal{P}_c$ that uses a single (V)OLE correlation, and has a communication complexity of 3 field elements (2 if only a uniformly sampled key is required) and a computational complexity of $\mathcal{O}(\lambda)$ field operations (essentially, the cost of a single exponentiation). This protocol requires 3 rounds (2 if only a uniformly sampled key is required). We refer to this as our *half-malicious* protocol. It is useful in scenarios where the server is trusted not to depart from its intended behavior.

- An enhanced protocol secure against *unbounded malicious* $\mathcal{P}_s$ and $\mathcal{P}_c$ that uses $\lambda + 8$ VOLE correlations, and has a communication complexity of $\lambda + 15$ field elements and a computational complexity of $\mathcal{O}(\lambda)$ field operations. We further show that for any small constant ϕ that, w.l.o.g., divides λ, the communication complexity can be improved to $\frac{\lambda}{\phi} + 2^{\phi} + 13$ field elements, and the number of VOLE correlations required is reduced to $\frac{\lambda}{\phi} + \mathcal{O}(1)$. This protocol requires 5 rounds (3 with the Fiat-Shamir transformation [FS87]). We refer to this as our *malicious* protocol.

The above protocols have additional features and extensions useful in different applications, including:

- **Offline-Online Mode:** Our half-malicious and malicious 2PC-Gold support an offline-online mode. The generation of VOLE correlations can be performed during an *input/key-independent* offline phase. Moreover, in our malicious 2PC-Gold, the majority of the work can be further shifted to the offline phase, resulting in an *online communication complexity of only* 6 $\mathbb{F}_p$ *elements.*

- **Batching:** Our protocols support batched evaluations for better amortized costs, allowing $\mathcal{P}_c$ to repeatedly evaluate the PRF under the *same* key with arbitrary inputs. For example, the online communication of our malicious 2PC-Gold can be reduced to amortized $2 + \frac{4}{n}$ $\mathbb{F}_p$ elements for n-batched evaluations.

- **Classical and PQ Instantiations:** Our half-malicious and malicious 2PC-Gold rely solely on standard VOLE correlations and are *information-theoretically secure* in the VOLE-hybrid model.[3] By employing appropriate methods to generate these VOLE correlations, our unmodified protocols can be instantiated to achieve either classical or post-quantum security in the plain model. We implement both options and report performance in Section 7. This performance will further improve in the future with ongoing optimizations in VOLE generation.

- **Key Verification:** Malicious 2PC-Gold supports *zero-knowledge proofs* [GMR85] of *any* NP relation over the key. Notably, we build efficient verifiable OPRFs based on this property.

See Section 3 for a concise technical overview of 2PC-Gold.

[2] To clarify, our 2PC protocols for evaluating Gold is *information-theoretically* secure in the VOLE-hybrid model. However, when we employ this protocol as an OPRF, we rely on computational assumptions to guarantee Gold being a secure PRF and to generate VOLE correlations.

[3] We note that looking ahead, our simulation does not need *rewinding* in the VOLE-hybrid model.

O-Gold and UC-Gold. OPRFs have been defined in multiple ways in the literature. In its basic form, just the 2PC over a PRF between server and client (i.e., 2PC-Gold) suffices for the OPRF definition. This, however, is not sufficient for some applications and stronger OPRF notions have been formulated. Our **second major contribution** is augmenting 2PC-Gold to achieve these notions.

Let H_1 be a hash function mapping arbitrary strings to $\mathbb{F}_p$ elements and H_2 be another hash function producing $\{0,1\}^{2\lambda}$ elements. We define the function $\mathsf{O\text{-}Gold}_k(x)$ as $\mathsf{H}_2(x, \mathsf{Gold}_k(\mathsf{H}_1(x)))$. A simple 2PC over O-Gold (leaking[4] $\mathsf{Gold}_k(\mathsf{H}_1(x))$ to $\mathcal{P}_c$) can be implemented by $\mathcal{P}_c$ inputting $\mathsf{H}_1(x)$ to 2PC-Gold, obtaining $y := \mathsf{Gold}_k(\mathsf{H}_1(x))$ and locally computing $\mathsf{H}_2(y)$. O-Gold function inherits the 2PC security of Gold and, in addition, supports arbitrary inputs, avoids collisions under the same k, and allows modeling its outputs as a random oracle. It also serves as a basis for a *verifiable* OPRF and to achieve strong *universal-composable* security.

We show how different forms of *verifiability* can be efficiently implemented for our OPRFs, including the approach to achieve standard verifiable OPRFs [JKK14]; see Section 6.1. Finally, we show how to extend O-Gold to achieve strong UC security by following the methodology from [BDFH24] in Section 6.1; we call the resultant OPRF UC-Gold.[5]

We refer to the three constructions—2PC-Gold, O-Gold, and UC-Gold—as OPRFs, each distinguished by its features.

Implementation. All our protocols are efficient in practice. *We implemented half-malicious and malicious* 2PC-Gold*, with classical and PQ VOLEs.*[6] We report the performance in Section 7. The cost of O-Gold is similar to 2PC-Gold, and we estimate concrete overheads for UC-Gold.

Consider $\lambda = 128$. Our half-malicious (resp. malicious) non-batched PQ 2PC-Gold needs 774KB (resp. 970KB) of communication, 568ms (resp. 1.1s) of wall-clock time in a WAN-like network, and 163ms (resp. 510ms) of wall-clock time in a LAN-like network. Most of the overhead comes from generating standard PQ VOLE correlations: it only needs 96B (resp. 2.7KB) in the VOLE-hybrid world. Indeed, by deploying the sublinear generation of large enough PQ VOLE correlations ($n \approx 10^7$ for half-malicious and $n \approx 3 \times 10^5$ for malicious in our experiments, but n can be much smaller; see Section 7.2) from [BCGI18], our half-malicious (resp. malicious) batched PQ 2PC-Gold only needs 100B (resp. 1.9KB) of amortized communication, 87µs (resp. 1.6ms) of amortized wall-clock time in a WAN-like network, and 57µs (resp. 1ms) of amortized wall-clock time in a LAN-like network. Finally, achieving UC-Gold requires an additional (amortizable) 899KB of communication.

1.2 Related Work

OPRF and Its Applications. The concept of obliviously evaluating PRFs dates back to [NR97] and was later refined and formalized as Oblivious PRF (OPRF) by [FIPR05].

Informally, OPRFs increase the entropy of the client's inputs, making them essential components in many real-world privacy-preserving applications. Notable examples include private set intersection (e.g., [CM20, KKRT16]), key management (e.g., [JKR19]), anonymous tokens (e.g., [DGS+18, KLOR20]), and password-based key-exchange (e.g., [JKX18]). Many OPRF applications (e.g., PSI and key management) can benefit from the batched OPRF.

[4] This leakage to $\mathcal{P}_c$ is usually harmless in OPRF applications.

[5] We note that [BDFH24]'s proof considers the classical random oracle model; extension to the Quantum ROM [BDF+11] is an interesting topic for future work.

[6] Our implementation: https://github.com/gconeice/PR-OPRF.

One of the most successful OPRF constructions is the 2HashDH OPRF [JKK14]. This construction relies on Diffie-Hallman-type assumptions and is extremely efficient: the basic version requires only 2 group elements of communication per evaluation. For more details, we refer the reader to a *Systematization of Knowledge* work on OPRFs [CHL22].

PQ OPRF Constructions. Research on PQ OPRF schemes has emerged in the past five years.

One approach is to build PQ OPRFs based on *isogenies*. This was initiated by [BKW20], relying on the SIDH and CSIDH assumptions. Subsequent work [Bas23, HHM+24] optimized these constructions and addressed vulnerabilities exposed by SIDH attacks [BKM+21, MM22] over [BKW20]'s constructions. However, isogeny-based OPRFs remain computationally intensive.

Another approach is to build PQ OPRFs based on *lattices*, initiated by the work [ADDS21]. This is merely a feasibility result: [ADDS21]'s constructions require more than 2MB (resp. 128GB) of communication per evaluation for semi-honest (resp. malicious) security. Very recent work [AG24, ESTX24] optimized the communication cost to amortized under 200KB for malicious security over batched evaluations. To our best knowledge, the only available implementation of lattice-based OPRF protocols is the one in [ADDS21] for semi-honest security.

An alternative, straightforward approach to constructing PQ OPRFs is to apply Yao's *Garbled Circuits* [Yao86] to, e.g., AES with the help of post-quantum *oblivious transfers* (e.g., [MR19, DCZ+24]). This idea was explored in [FOO23]. However, the size of the garbled circuit for AES is relatively large, making further optimization inherently hard.

This approach, however, opens up the possibility of building PQ OPRFs based on PQ MPC-friendly PRFs. The literature has explored this method using two such PRFs:

- **"Crypto Dark Matter" PRFs [BIP+18]:** These PRFs are specifically designed for efficient use within MPC. Several proposals have explored the oblivious evaluations of these PRFs. In particular, [DGH+21] studied preprocessing 2PC over them, and [ADDG24] studied TFHE over them. Most recently, [APRR24] has aggressively improved the constructions of these PRFs and showed that the amortized communication cost per evaluation over batched evaluations can only be ≈1Kbit for semi-honest security. However, these PRF constructions rely on new assumptions that need further study. Moreover, achieving malicious security may require substantial effort, e.g., [ADDG24] relies on a heuristic argument to achieve it. Furthermore, these PRFs are only *weak* PRFs (i.e., they are only secure when the input is chosen uniformly at random). In the semi-honest case, this is not an issue as the input can be hashed; however, in the malicious case, additional mechanisms are required.

- **Legendre PRFs [Dam90]:** Another type of MPC-friendly PRF is the *Legendre PRF*, originally proposed by Damgård in 1988. This PRF was first identified as MPC-friendly by [GRR+16] and has since been employed to construct PQ OPRFs in works [SHB23, BDFH24]. The state-of-the-art construction presented in the recent work of [BDFH24] incurs approximately 900KB of (non-amortizable) communication per evaluation for malicious security.

Our work constructs PQ OPRFs using the generalized Legendre PRF, i.e., the Power-Residue PRF also introduced in [Dam90]. See Section 2.5 for more detailed discussions of the Legendre and Power-Residue PRFs.

Recent work [KCM24] considers building PQ distributed OPRF from the Legendre PRF. More specifically, [KCM24] considers distributed OPRF with multiple ($\geq$3) non-colluding servers (i.e., the threshold setting), while we consider the classical OPRF setting with a single server. Meanwhile, for

n servers, the cost in [KCM24] is exponential in n, making it unsuitable for many servers. Extending our protocols to the distributed OPRF setting is an interesting future direction.

We note that our work may appear to extend the recent study by [BDFH24], which also utilizes VOLE correlations; however, this is not the case. In particular, our protocols use VOLE correlations in the opposite direction to build a novel customized 2PC for PRF evaluation, enabling efficient extension to batched (amortized) evaluations and malicious security. Moreover, our protocols rely solely on black-box usage of the *standard* VOLE correlations, whereas [BDFH24] requires non-black-box modifications of the VOLE functionality. Finally, our work provides a complete end-to-end implementation along with a comprehensive benchmark. At the same time, our work borrows from [BDFH24] the elegant methodology for building the strong UC variant of Gold.

VOLE and VOLE-Based ZK. Our protocols leverage the standard VOLE correlations; see Section 2.6. In particular, our malicious OPRFs utilize efficient zero-knowledge proofs based on VOLE, commonly referred to as VOLE-based ZK; see Section 2.7.

Concrete Performance Comparison. Section 7.3 includes a concrete performance comparison between our protocols and prior work.

2 Preliminaries

2.1 Notation

- λ is the security parameter (e.g., 128).
- The server is $\mathcal{P}_s$. We refer to $\mathcal{P}_s$ by she, her, hers...
- The client is $\mathcal{P}_c$. We refer to $\mathcal{P}_c$ by he, his, him...
- $x := y$ denotes that y is *assigned* to x.
- We denote sets by upper-case letters. We denote that x is uniformly drawn from a set S by $x \xleftarrow{\$} S$.
- $[m] := \{1, \ldots, m\}$ and $[a, b) := \{a, a+1, \ldots, b-1\}$.
- We use $\mathbb{F}$ to denote a finite field. For a prime p, we use $\mathbb{F}_p$ to denote the standard modular-p arithmetic field.
- We use $\mathbb{Z}$ to denote the integer ring.
- We use $\mathbb{G}$ to denote a cyclic group.
- We denote row vectors by bold lower-case letters (e.g., $\boldsymbol{a}$), where a_i (or $a[i]$) denotes the i-th component of $\boldsymbol{a}$ (starting from 1). $|\boldsymbol{a}|$ denotes the length of $\boldsymbol{a}$.
- For vectors $\boldsymbol{a}, \boldsymbol{b}$ where $|\boldsymbol{a}| = |\boldsymbol{b}|$. $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ denotes the inner product; $\boldsymbol{a} \odot \boldsymbol{b}$ denotes the element-wise product.
- $x\boldsymbol{a} := (xa_1, \ldots xa_m)$ for some $\boldsymbol{a} = (a_1, \ldots, a_m)$.
- We use n to denote the number of OPRF evaluations.

2.2 Schwartz-Zippel Lemma

The security of our protocols relies on the well-known *Schwartz-Zippel* lemma [Sch80, Zip79] and its corollary:

Lemma 1 (Schwartz-Zippel)**.** *Let $\mathbb{F}$ be a finite field and $P \in \mathbb{F}[X]$ be a polynomial of degree d (in X). Then*

$$\Pr\left[P(v) = 0 \mid v \xleftarrow{\$} \mathbb{F}\right] \leq \frac{d}{|\mathbb{F}|}$$

Corollary 1. *Let $\mathbb{F}$ be a field and d be a positive integer, for any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^d$, if $\boldsymbol{a} \neq \boldsymbol{b}$, then*

$$\Pr\left[\langle \boldsymbol{a}, \boldsymbol{c}\rangle = \langle \boldsymbol{b}, \boldsymbol{c}\rangle \mid \beta \xleftarrow{\$} \mathbb{F}, \boldsymbol{c} := (1, \beta, \ldots, \beta^{d-1})\right] \leq \frac{d-1}{|\mathbb{F}|}$$

Indeed, our work only requires the Schwartz-Zippel lemma for the univariate case.

2.3 Security Model

We formalize our protocols in the universally composable (UC) framework [Can01]. We consider static corruptions. In particular, our basic OPRF protocol is secure against a semi-honest $\mathcal{P}_s$ and a malicious $\mathcal{P}_c$. We refer to this protocol as the *half-malicious* OPRF. On the other hand, our malicious protocol is secure against malicious $\mathcal{P}_s$ and $\mathcal{P}_c$. For simplicity, we omit standard UC (sub-)session IDs. We also omit that each time a party sends a message to the functionality, it provides a receipt to the simulator, and that whenever the functionality needs to deliver an output to a party, it waits for the simulator's instruction to do so. Note, this includes the standard *security with abort*.

2.4 Pohlig-Hellman Discrete Logarithm Algorithm

Our work uses the well-known *Pohlig-Hellman* algorithm [PH78] for solving (special-case) discrete logarithms (DLOGs). Consider a finite cyclic group $\mathbb{G}$ of order m generated by $h \in \mathbb{G}$, where m has a prime factorization as $m = \prod_{i=1}^{r} p_i^{e_i}$. Then there exists a deterministic algorithm to solve the DLOG of any element $x \in \mathbb{G}$ in base h in $\mathcal{O}\left(\sum_i e_i \cdot (\log m + \sqrt{p_i})\right)$ group operations.

Our work uses the Pohlig-Hellman algorithm in a finite cyclic group of order $m = 2^\lambda$, in which the DLOG can be solved in $\mathcal{O}(\lambda^2)$ group operations.

2.5 Legendre and Power-Residue PRFs

Pseudorandom Functions (PRFs) [GGM86] are deterministic keyed functions that are indistinguishable from random functions, when the key is chosen uniformly at random.

Our protocols essentially obliviously evaluate the Power-Residue PRF, which is a generalization of the Legendre PRF. Both PRFs can be traced back to the work of Damgård [Dam90] in 1988. This section briefly reviews these two PRFs and existing (classical and quantum) attacks.

Legendre PRF. Let p be a prime. For $a \in \mathbb{F}_p$, the Legendre symbol $\left(\frac{a}{p}\right)$ is defined as: $\left(\frac{a}{p}\right) := a^{\frac{p-1}{2}} \pmod p$.

It is conjectured that for a sufficiently large *public* prime p, the function $F_k(x) := \left(\frac{k+x}{p}\right)$ is a PRF defined over $\mathbb{F}_p \mapsto \{1, -1\}$, with key $k \in \mathbb{F}_p$. The output of F_k on $-k$ is zero, which is neither 1 nor -1. However, since k is supposed to be chosen uniformly, an adversary cannot find the single input $x = -k$ that results in an output of zero.[7]

Power-Residue PRF. A glaring downside of the Legendre PRF is that it only produces a single-bit output. In many scenarios, (oblivious) PRF is only useful when it can produce $\mathcal{O}(\lambda)$ bits. A direct way to achieve this is by repeating the Legendre PRF $\mathcal{O}(\lambda)$ times (e.g., on $\mathcal{O}(\lambda)$ different keys; see [BDFH24]). This is clearly undesirable. Instead, we exploit the higher order residues, a natural generalization of the Legendre PRF also proposed by [Dam90].

[7] We note that in (UC) OPRF constructions, the UC environment may choose the input $x = -k$. This is also true for OPRFs based on the Power-Residue PRF, and we handle this case carefully.

In more detail, consider a sufficiently large prime $p = eg + 1$ where p, e, g are public. For a random key $k \xleftarrow{\$} \mathbb{F}_p$, the PRF F on input $x \in \mathbb{F}_p$ is defined as:

$$F_k(x) := (k + x)^{\frac{p-1}{e}} = (k + x)^g \pmod p$$

This PRF can produce e different non-zero outputs (and a unique input results in a zero output). This is because, as indicated by its name, each non-zero output is a g-th residue (aka an e-th root of unity). This is crucial for efficiency, as now one evaluation can produce a $\lfloor \log e \rfloor$-bit output. Note, for $e = 2$, the Power-Residue PRF is the Legendre PRF.

In this work, we set e to be 2^λ and name this PRF Gold. One evaluation of Gold can produce λ-bit entropy. See Section 4.1 for how to transfer the output to $\{0,1\}^\lambda$, or $\mathcal{O}(\lambda)$ bits in general, and for associated formal assumptions.

Statistical Properties. The study of the distribution of Legendre symbols dates to Davenport, 1931 [Dav31], followed by extensive work (e.g., [Per92, MS97, GMS14, Tót07, DHS98, DE52]). It points to statistical properties of Legendre symbols that resemble the behavior of fair coin tosses. Some of these properties can be extended to Power-Residue symbols [DE52].

Attacks and Quantum Resistance. Several works attempt to break the Legendre and Power-Residue PRFs. This trend has become popular recently, mainly because of Legendre's candidate usage in the Ethereum 2.0 blockchain.

Both Legendre and Power-Residue PRF can be broken in quantum polynomial time with a single *quantum superposition query access* to the PRF [RS04, vDH00]. However, currently, cryptographic protocols and algorithms are only implemented on classical computers, including OPRFs. As long as all systems holding the OPRF key are classical, an adversary cannot make a quantum superposition of queries. Thus, we restrict ourselves to attacks with classical queries.

For the Legendre PRF, classical and quantum attacks include [BBUV20, KKK20b, MZ22, SHB23, KKK20a, Kho19, FS21]. In particular, the best-known (quantum) attack by [FS21] has time complexity $2^{\mathcal{O}(\log p)} \cdot p^{1/3}$. For the Power-Residue PRF, the additional (quantum) attack by [BBUV20] has time complexity $\mathcal{O}(\frac{p \log^2 p}{Le \log^2 e})$ where $L \leq p^{1/4}$ denotes the number of queries.

Finally, a recent elegant work [CW24b] considers an extension of the Legendre PRF to any modulo N instead of a prime modulo p. The authors break the pseudorandomness of such a function when N is composite and can be factored, which implies PQ insecurity [Sho94]. However, this factoring-based attack (and subsequent work [CW24a]) does *not* apply to the Legendre and Power-Residue PRFs modulo a prime p.

Legendre vs. Power-Residue with $e = 2^\lambda$. When $e = 2^\lambda$, breaking the Power-Residue PRF is not harder than breaking the Legendre PRF. Indeed, one can trivially compute $(k + x)^{\frac{p-1}{2}}$ from $(k + x)^{\frac{p-1}{e}}$. Conversely, we do not know of any reduction from breaking the Legendre PRF to the Power-Residue PRF. We leave it as an interesting open problem.

Nevertheless, considering the previously cited works, we believe that our Power-Residue PRF Gold is a natural, reasonable assumption (see the formal definition in Section 4.1) and encourage further study on its pseudorandomness.

2.6 Vector OLE

Our OPRFs use Vector Oblivious Linear Evaluation (VOLE) correlations: they are in the VOLE-hybrid model. VOLE correlations of size m (or m VOLE correlations) over a field $\mathbb{F}$ are correlated

Functionality $\mathcal{F}_{\mathsf{VOLE}}^{a,b}$

$\mathcal{F}_{\mathsf{VOLE}}^{a,b}$, parameterized by a field $\mathbb{F}$, running with a sender $\mathcal{P}_a$, a receiver $\mathcal{P}_b$ and an adversary $\mathcal{S}$, proceeds as follows:

Initialize. Upon receiving (`init`) from $\mathcal{P}_a$ and $\mathcal{P}_b$, if $\mathcal{P}_b$ is honest, set $\Delta^{(b)} \xleftarrow{\$} \mathbb{F}$, else receive $\Delta^{(b)}$ from $\mathcal{S}$. Store $\Delta^{(b)}$. Send $\Delta^{(b)}$ to $\mathcal{P}_b$. Ignore subsequent (`init`).

Extend. Upon receiving $(\texttt{extend}, m)$ from $\mathcal{P}_a$ and $\mathcal{P}_b$:

- If $\mathcal{P}_b$ is honest, set $\boldsymbol{v} \xleftarrow{\$} \mathbb{F}^m$, else receive $\boldsymbol{v} \in \mathbb{F}^m$ from $\mathcal{S}$.
- If $\mathcal{P}_a$ is honest, set $\boldsymbol{u} \xleftarrow{\$} \mathbb{F}^m$ and compute $\boldsymbol{w} := \boldsymbol{v} - \boldsymbol{u}\Delta^{(b)} \in \mathbb{F}^m$, else receive $\boldsymbol{u} \in \mathbb{F}^m$ and $\boldsymbol{w} \in \mathbb{F}^m$ from $\mathcal{S}$ and compute $\boldsymbol{v} := \boldsymbol{w} + \boldsymbol{u}\Delta^{(b)} \in \mathbb{F}^m$.
- Send $(\boldsymbol{u}, \boldsymbol{w})$ to $\mathcal{P}_a$ and $\boldsymbol{v}$ to $\mathcal{P}_b$.

Figure 1: The VOLE correlation functionality [WYKW21].

random vectors held by a sender $\mathcal{P}_a$ and a receiver $\mathcal{P}_b$. The sender $\mathcal{P}_b$ obtains a uniformly sampled scalar $\Delta^{(b)} \xleftarrow{\$} \mathbb{F}$ and a vector $\boldsymbol{v} \xleftarrow{\$} \mathbb{F}^m$. The receiver $\mathcal{P}_a$ obtains a uniformly sampled vector $\boldsymbol{u} \xleftarrow{\$} \mathbb{F}^m$ and a correlated vector $\boldsymbol{w} := \boldsymbol{v} - \boldsymbol{u}\Delta^{(b)}$.

Our protocols rely solely on a weaker form of VOLE correlations known as *endemic* [MR19] VOLE correlations, where the adversary is allowed to choose its own correlated shares. See Figure 1 for the formal definition.

Looking ahead, our half-malicious OPRFs use $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$: $\mathcal{P}_s$ plays the role of the VOLE receiver, whereas $\mathcal{P}_c$ plays the role of the VOLE sender. Our malicious OPRFs also use $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ (with $\mathcal{P}_s$'s and $\mathcal{P}_c$'s roles are switched in VOLE).

Multiple ways exist to realize $\mathcal{F}_{\mathsf{VOLE}}$ with malicious security. In this work, we adopt the approaches in [BDFH24] and [WYKW21]. Namely, when we only need a small amount of VOLE correlations (e.g., evaluating OPRF once), we realize it using the subset VOLEs [Roy22, BBD+23] and randomly linear combinations [BDFH24], via *Oblivious Transfers* (OTs) [Rab05]. In contrast, when we need many VOLE correlations (e.g., evaluating multiple inputs over a fixed-key OPRF), we deploy the "silent VOLE extension" technique to save (amortized) costs. This is also known as pseudorandom correlation generator (PCG), stemming from the seminal works [BCGI18, BCG+19].

The above approaches only rely on minicrypt-type cryptographic primitives (e.g., PRF, PRG) and/or the *Learning Parity with Noise* (LPN) assumption [BFKL94], in the OT-hybrid model. When using post-quantum OTs (e.g., [MR19, DCZ+24]), our instantiations of $\mathcal{F}_{\mathsf{VOLE}}$ are PQ secure:

Lemma 2 (Informal)**.** *Assuming the existence of one-way functions or the hardness of LPN, there exists a protocol that UC-emulates $\mathcal{F}_{\mathsf{VOLE}}$ (Figure 1) in the OT-hybrid model.*

A more straightforward way to realize $\mathcal{F}_{\mathsf{VOLE}}$ is via *linearly homomorphic encryptions*. With lattice-type assumptions, post-quantum security can be achieved. Our estimation shows that the performance difference is nuanced (see discussions in, e.g., [BDSW23]). We leave the exploration of this approach as a valuable future work. Crucially, our OPRFs are black-box in the VOLE functionality and can benefit from any future improvement over VOLE constructions.

2.7 VOLE-Based ZK

Our malicious OPRFs also use VOLE correlations to build lightweight ZKPs to protect against a malicious $\mathcal{P}_s$, aligning with recent progress on VOLE-based ZK (e.g., [DIO21, YSWW21, HY24]).

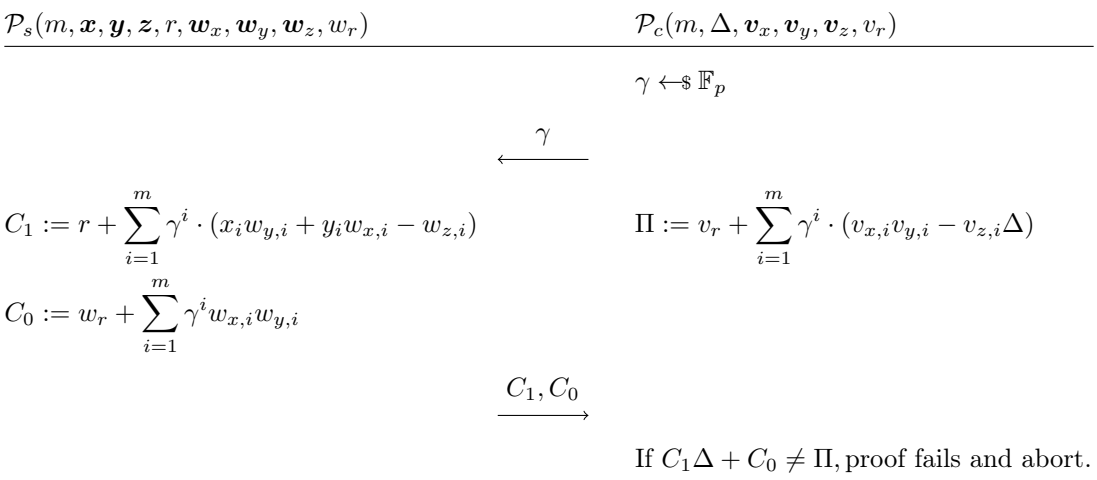

Figure 2: The diagram of the batch LPZK technique. Here, $\mathcal{P}_s$ is the prover, and $\mathcal{P}_c$ is the verifier.

Here, $\mathcal{P}_s$ is the prover, whereas $\mathcal{P}_c$ is the verifier. We review related techniques in this section.

IT-MAC. Consider a single VOLE correlation generated by $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ defined over $\mathbb{F}_p$. That is, $\mathcal{P}_s$ holds u and w_u whereas $\mathcal{P}_c$ holds $\Delta^{(c)}$ and v_u, such that $v_u = w_u + u\Delta^{(c)}$. This correlation can be rather interpreted as the *information-theoretic message authentication code* (IT-MAC) [BDOZ11, NNOB12] commitment over the value u, from $\mathcal{P}_s$ to $\mathcal{P}_c$. We denote this correlation as $\langle (u, w_u), v_u \rangle_{\Delta^{(c)}}$ or $[u]_{\Delta^{(c)}}$ in short. IT-MAC commitments have the following properties:

- **Perfect Hiding:** v_u and $\Delta^{(c)}$, held by $\mathcal{P}_c$, are independent of the committed value u.
- **Statistical Binding:** $\mathcal{P}_s$ can open $[u]_{\Delta^{(c)}}$ by sending u and w_u, where $\mathcal{P}_c$ would check if $v_u \stackrel{?}{=} w_u + u\Delta^{(c)}$. For a malicious $\mathcal{P}_s$ to open it to a different value $u' \neq u$, she has to guess $\Delta^{(c)}$—succeed with probability $\frac{1}{p}$.
- **Linear Homomorphism:** Suppose $\mathcal{P}_s$ and $\mathcal{P}_c$ hold $\langle (x, w_x), v_x \rangle_{\Delta^{(c)}}$ and $\langle (y, w_y), v_y \rangle_{\Delta^{(c)}}$. For any public $\alpha, \beta, \gamma \in \mathbb{F}_p$, parties can locally generate $[\alpha x + \beta y + \gamma]_{\Delta^{(c)}}$ as follows: $\mathcal{P}_s$ computes $\alpha w_x + \beta w_y$, whereas $\mathcal{P}_c$ computes $\alpha v_x + \beta v_y + \gamma\Delta^{(c)}$.

Note, the linear homomorphism property implies that, for a random IT-MAC $[u]_{\Delta^{(c)}}$ (from VOLE), $\mathcal{P}_s$ can send $z - u$ to commit to z, where u acts as a one-time pad [Bea95].

Line-Point Zero-Knowledge. Our malicious OPRFs need $\mathcal{P}_s$ to prove in ZK that three IT-MAC commitments form a multiplication triple. Namely, $\mathcal{P}_s$ and $\mathcal{P}_c$ hold $[x]_{\Delta^{(c)}} = \langle (x, w_x), v_x \rangle_{\Delta^{(c)}}$, $[y]_{\Delta^{(c)}} = \langle (y, w_y), v_y \rangle_{\Delta^{(c)}}$, $[z]_{\Delta^{(c)}} = \langle (z, w_z), v_z \rangle_{\Delta^{(c)}}$, where $\mathcal{P}_s$ wants to prove in ZK to $\mathcal{P}_c$ that $z = xy$. This can be done using the *line-point zero-knowledge* (LPZK) technique [YSWW21, DIO21]. LPZK relies on the following equality (let $\Delta = \Delta^{(c)}$ for simplicity):

$$\begin{aligned} &\overbrace{v_x v_y - v_z \Delta}^{\text{known by } \mathcal{P}_c} \\ =&(x\Delta + w_x)(y\Delta + w_y) - (z\Delta + w_z)\Delta \end{aligned}$$

$$= \underbrace{(xy - z)}_{0} \Delta^2 + \underbrace{(xw_y + yw_x - w_z)}_{\text{known by } \mathcal{P}_s} \Delta + \underbrace{w_x w_y}_{\text{known by } \mathcal{P}_s}$$

Hence, if ZK is not required, $\mathcal{P}_s$ can send two coefficients $C_1 = xw_y + yw_x - w_z$ and $C_0 = w_x w_y$, and $\mathcal{P}_c$ will check if $v_x v_y - v_z \Delta \stackrel{?}{=} C_1 \Delta + C_0$. This is sound because the equality holds only when Δ happens to be the root of a quadratic equation if $xy \neq z$. If Δ has full entropy to $\mathcal{P}_s$, this will only happen with probability $\frac{2}{p}$ since this is a degree-2 polynomial [Sch80, Zip79]. Of course, this is not ZK as C_1, C_0 are correlated with x, y, z. ZK can be recovered by consuming one random IT-MAC $[r]_{\Delta^{(c)}}$ (from VOLE). That is, $\mathcal{P}_s$ sends $C_1 = xw_y + yw_x - w_z + r$ and $C_0 = w_x w_y + w_r$ and $\mathcal{P}_c$ checks if $v_x v_y - v_z \Delta + v_r \stackrel{?}{=} C_1 \Delta + C_0$.

LPZK technique can be optimized in the batched settings. Namely, parties hold $[\boldsymbol{x}]_{\Delta^{(c)}}, [\boldsymbol{y}]_{\Delta^{(c)}}$, $[\boldsymbol{z}]_{\Delta^{(c)}}$ each of length m, and $\mathcal{P}_s$ wants to prove in ZK that $\boldsymbol{z} = \boldsymbol{x} \odot \boldsymbol{y}$. In this case, instead of sending $2m$ coefficients, the m verifications can be batched as follows: $\mathcal{P}_c$ sends a random linear combination, $\mathcal{P}_s$ evaluates the linear combination on the m coefficients C_1 and m coefficients C_0, and then sends resulting 2 aggregated coefficients. One random IT-MAC $[r]_{\Delta^{(c)}}$ is needed as before for ZK. In this work, this random linear combination is generated as the powers of a random field element, achieving information-theoretical security. The full batched LPZK is depicted in Figure 2.

Generalized LPZK. As observed by [YSWW21], the LPZK technique can be generalized to higher degree polynomials. In this work, we use this generalized technique to improve concrete efficiency in the following setting: $\mathcal{P}_s$ and $\mathcal{P}_c$ hold $[x]_{\Delta^{(c)}} = \langle (x, w_x), v_x \rangle_{\Delta^{(c)}}, [y]_{\Delta^{(c)}} = \langle (y, w_y), v_y \rangle_{\Delta^{(c)}}$, where $\mathcal{P}_s$ wants to prove in ZK to $\mathcal{P}_c$ that $y = x^e$ for some public positive integer e. The core of this technique is the following (generalized) equality:

$$\begin{aligned}
&\overbrace{v_x^e - v_y \Delta^{e-1}}^{\text{known by } \mathcal{P}_c} \\
=&(x\Delta + w_x)^e - (y\Delta + w_y)\Delta^{e-1} \\
=&(x^e - y)\Delta^e + (ex^{e-1} w_x - w_y)\Delta^{e-1} + \sum_{i=0}^{e-2} \binom{e}{i} x^i w_x^{e-i} \Delta^i
\end{aligned}$$

where $\binom{e}{i}$ denotes the binomial coefficient "e chooses i". Crucially, all coefficients before each Δ term in the last row are known by $\mathcal{P}_s$. Hence, if we do not need ZK, $\mathcal{P}_s$ can send e coefficients where $\mathcal{P}_c$ checks the equality. Similar to the LPZK, a malicious $\mathcal{P}_s$ can only create a wrong proof with probability $\frac{e}{p}$. ZK can be recovered using $e - 1$ random IT-MACs (from VOLE). See [YSWW21, HHK+24] for details. The batched optimization can also be applied: with m e-th exponential proofs, the total communication cost is $e + 1$ $\mathbb{F}_p$ elements.

The generalized LPZK can be used to improve the following task (used by our malicious OPRFs): parties hold $[x]_{\Delta^{(c)}}$ and want to obtain $[x^{2^{128}}]_{\Delta^{(c)}}$. The naïve approach would require $\mathcal{P}_s$ to commit to 128 intermediate results with a batched multiplication LPZK proof—the total communication is 131 $\mathbb{F}_p$ elements. By deploying the generalized LPZK, $\mathcal{P}_s$ can only commit to, e.g., $x^{2^4}, x^{2^8}, x^{2^{12}}, \cdots$, with a batched 2^4-th exponential relation proof—the total communication decreases to $128/4 + 2^4 + 1 = 49$ $\mathbb{F}_p$ elements. Note that this optimization does not improve communication asymptotically. It also increases the computation concretely.

All the proof techniques shown in this section provide *information-theoretic* security in the VOLE-hybrid model.

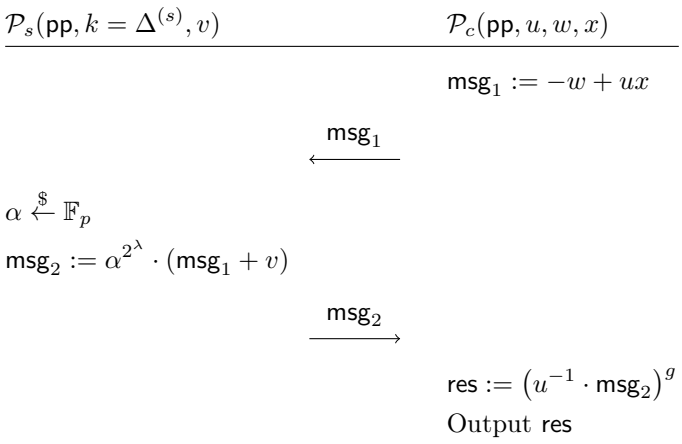

Figure 3: The diagram of our half-malicious 2PC-Gold between $\mathcal{P}_s$ and $\mathcal{P}_c$. $\mathsf{pp} := (1^\lambda, p, g)$ denotes the public parameters. $\mathcal{P}_s$ holds the PRF key k whereas $\mathcal{P}_c$ holds the PRF input x. The objective is to allow $\mathcal{P}_c$ obliviously obtain $(k+x)^g$. Recall that k, v, u, w form a single VOLE correlation, i.e., $v = w + uk$.

3 Technical Overview

In this section, we present our half-malicious and malicious 2PC-Gold with sufficient detail to understand our first major contribution. Other Gold OPRFs, formalization, and detailed analysis are presented in subsequent sections.

We focus on efficiently and obliviously evaluating Gold in the VOLE-hybrid model (see Figure 1). Namely, for a public prime $p = 2^\lambda \cdot g + 1$ where g is of $\mathcal{O}(\lambda)$ bits, $\mathcal{P}_s$ holds a PRF key $k \in \mathbb{F}_p$ whereas $\mathcal{P}_c$ holds a PRF input $x \in \mathbb{F}_p$, and the objective is to allow $\mathcal{P}_c$ to learn $(k+x)^g$ over $\mathbb{F}_p$ obliviously, with the help of VOLE correlations over $\mathbb{F}_p$. See Section 4 for the choice of g and how to efficiently convert the output into a $\mathcal{O}(\lambda)$-bit element.

We first address scenarios where $\mathcal{P}_s$ uses a (fresh) uniformly sampled PRF key. Later, we show how to adapt the protocol to accommodate a server-specified key at the cost of $\mathcal{P}_s$ sending a single extra field element.

We note that Section 2.1 includes the used notation.

3.1 Overview of Half-Malicious 2PC-Gold

We give an overview of our half-malicious 2PC-Gold, namely, providing security against a malicious client and a semi-honest server. The protocol is depicted in Figure 3.

Random Root Technique. Our protocol relies heavily on a simple observation [GRR+16, FKN94]: To evaluate $(k+x)^g$ obliviously, it is sufficient to let $\mathcal{P}_c$ obliviously learn $\alpha^{2^\lambda} \cdot (k+x)$, where $\alpha \in \mathbb{F}_p$ is uniformly sampled and *unknown* to $\mathcal{P}_c$. Then, $\mathcal{P}_c$ can locally compute $\left(\alpha^{2^\lambda} \cdot (k+x)\right)^g$ over $\mathbb{F}_p$. This works because (1) for any $\alpha \in \mathbb{F}_p^*$, $\alpha^{2^\lambda \cdot g} = 1$ over $\mathbb{F}_p$ (which ensures correctness), and (2) $\alpha^{2^\lambda} \cdot (k+x)$ is distributed as a uniformly chosen root of the equation $X^g = (x+k)^g$ over $\mathbb{F}_p$, which can be simulated by choosing such a random root (see Section 4.2).

Protocol. Our half-malicious 2PC-Gold (see Figure 3) only requires two messages in the VOLE-hybrid model. Informally, the first message is sent by $\mathcal{P}_c$ and encrypts the evaluated input x. Then, the second message is sent by $\mathcal{P}_s$ and allows $\mathcal{P}_c$ to recover $\alpha^{2^\lambda} \cdot (k + x)$ over $\mathbb{F}_p$ where α is sampled uniformly by $\mathcal{P}_s$.

In detail, $\mathcal{P}_s$ and $\mathcal{P}_c$ use $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ (Figure 1) to obtain one VOLE correlation: $\mathcal{P}_s$ holds $\Delta^{(s)}, v$ whereas $\mathcal{P}_c$ holds u, w such that $v = w + u\Delta^{(s)}$. Since $\Delta^{(s)}$ is uniformly sampled, it can be used as the PRF key k.[8] We make the following remarks w.r.t. correctness, security, and performance:

- **Correctness:** Recall that $v = w + u\Delta^{(s)} = w + uk$. The correctness relies on the following equality:

$$\begin{aligned} u^{-1} \cdot \mathsf{msg}_2 &= u^{-1} \cdot \alpha^{2^\lambda} \cdot (\mathsf{msg}_1 + v) \\ &= u^{-1} \cdot \alpha^{2^\lambda} \cdot (-w + ux + v) \\ &= u^{-1} \cdot \alpha^{2^\lambda} \cdot (uk + ux) \\ &= \alpha^{2^\lambda} \cdot (k + x) \end{aligned}$$

  Note that u is zero with negligible probability $\frac{1}{p}$ since it is uniformly sampled by $\mathcal{F}_{\mathsf{VOLE}}$.

- **Semi-honest $\mathcal{P}_s$:** This protocol is secure against a semi-honest $\mathcal{P}_s$ in the VOLE-hybrid model. Indeed, consider $\mathsf{msg}_1 + v = u \cdot (k + x)$. If $k + x \neq 0$, since u is uniformly random in the view of $\mathcal{P}_s$, so is $u \cdot (k + x)$. Thus, it can be simulated by a random value in $\mathbb{F}_p$. In case $k + x = 0$, $\mathcal{P}_s$ learns this fact, which we formalize as a 1-bit leakage in our UC treatment (Figure 5). See Section 3.4 for simple and efficient ways to eliminate this leakage.

- **Malicious $\mathcal{P}_c$:** This protocol is secure against a malicious $\mathcal{P}_c$ in the VOLE-hybrid model. Indeed, the only place a malicious $\mathcal{P}_c$ can cheat is by choosing an arbitrary $\widetilde{\mathsf{msg}_1}$.

  Let us first assume that $u \neq 0$. Then the simulator $\mathcal{S}$ can extract a corresponding input $\widetilde{x}$ from $\widetilde{\mathsf{msg}_1}$ as follows: since $\mathcal{S}$ knows u and w (by emulating $\mathcal{F}_{\mathsf{VOLE}}$), it can compute $\widetilde{x} := (\widetilde{\mathsf{msg}_1} + w) \cdot u^{-1}$. Then, $\mathcal{S}$ gets from the ideal functionality the PRF output $(k + \widetilde{x})^g$. Finally, $\mathcal{S}$ can use the random root technique described above to simulate msg_2 as a random root X of $X^g = (k + \widetilde{x})^g$.

  If $u = 0$, we remark that $v = w$. Thus, $\mathcal{S}$ knows v and can trivially simulate $\mathsf{msg}_2 = \alpha^{2^\lambda} \cdot (\widetilde{\mathsf{msg}_1} + v)$.

- **Performance:** Our half-malicious 2PC-Gold is extremely efficient. It has near-optimal communication: 2 messages, each consisting of a single $\mathbb{F}_p$ element. It consumes a single VOLE correlation per OPRF invocation, and the time complexity of each party is dominated by the time to perform a single exponentiation modulo p (with $\mathcal{O}(\lambda)$-bit exponents). In more detail, $\mathcal{P}_c$ and $\mathcal{P}_s$ also perform one multiplication modulo p, and $\mathcal{P}_c$ performs an inversion modulo p, which can be done using the standard extended GCD algorithm in time $\tilde{O}(\log^2 p)$.

Note that the security of our protocol in the VOLE-hybrid model is information-theoretic.

Batched Evaluations. Our protocol can support n-batched evaluations using n VOLE correlations (which can be generated more efficiently as a batch). That is, we can execute the protocol in Figure 3 in n parallel instances: for each $i \in [n]$, $\mathcal{P}_c$ sends $-w_i + u_i x_i$; then $\mathcal{P}_s$ sends $\alpha_i^{2^\lambda} \cdot u_i \cdot (k + x_i)$;

[8] In applications where $\mathcal{P}_s$ inputs an arbitrary key k^* (e.g., a key committed earlier), $\mathcal{P}_s$ will send $k^* - \Delta^{(s)}$, allowing $\mathcal{P}_c$ to adjust the correlation as $v = w + uk^*$. That is, $\mathcal{P}_c$ sets $w := w - (k^* - \Delta^{(s)}) \cdot u$.

$\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{u} \in \mathbb{F}_p^n$ are output by $\mathcal{F}_{\mathsf{VOLE}}$ s.t. $\boldsymbol{v} = \boldsymbol{w} + \boldsymbol{u}k$. As noted in footnote 8, if $\mathcal{P}_s$ wants to adjust the key to a given value k^* instead of using the $\mathcal{F}_{\mathsf{VOLE}}$'s $\Delta^{(s)}$, it suffices to send $k^* - \Delta^{(s)}$ only once. We remark that these n inputs do not need to be different.

3.2 Overview of Malicious 2PC-Gold

We give an overview of our malicious 2PC-Gold, which is information-theoretically secure against a malicious $\mathcal{P}_s$ and a malicious $\mathcal{P}_c$ in the VOLE-hybrid model.

Here, we focus on 2PC-Gold with malicious security. This is different from the *public verifiability* property of an OPRF, which is covered in Section 6.1.

Malicious $\mathcal{P}_s$ Attack Surface. To see how our malicious 2PC-Gold works, it is instructive to analyze what a malicious $\mathcal{P}_s$ can do in our half-malicious 2PC-Gold. The only place the malicious $\mathcal{P}_s$ can inject an error is in the second message msg_2 sent from $\mathcal{P}_s$ to $\mathcal{P}_c$. As a result, to protect against a malicious $\mathcal{P}_s$, it suffices to require $\mathcal{P}_s$ to prove in ZK that msg_2 is generated correctly. That is,

1. $\mathcal{P}_s$ adds the value v, output by $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$, to msg_1, yielding the intermediate result $\mathsf{int} = \mathsf{msg}_1 + v$.
2. $\mathcal{P}_s$ multiplies int with α^{2^λ} to generate $\mathsf{msg}_2 = \mathsf{int} \cdot \alpha^{2^\lambda}$ for some $\alpha \in \mathbb{F}_p$.

Even with such zero-knowledge proof, we note that a malicious $\mathcal{P}_s$ is still able to learn whether $k + x$ equals zero. This is the 1-bit leakage we mentioned in Section 3.1. See Section 3.4 for details on how to eliminate it efficiently.

Deploying ZK. Malicious 2PC-Gold exploits the VOLE-based ZK techniques (see Section 2.7) to ensure a well-formed msg_2. Namely, $\mathcal{P}_s$ and $\mathcal{P}_c$ will use another VOLE correlation functionality, *with reversed roles* as $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, in addition to the correlations required by the half-malicious 2PC-Gold from Figure 3. These new correlations can be viewed as IT-MAC commitments from $\mathcal{P}_s$ to $\mathcal{P}_c$ over a $\mathcal{P}_s$-known random element $r \in \mathbb{F}_p$, denoted as $[r]_{\Delta^{(c)}}$. Note, $\Delta^{(c)}$ is known by $\mathcal{P}_c$ and can be viewed as the verifier private coins in the ZK proof. (Instead, $\Delta^{(s)}$ in our half-malicious protocol is used as the OPRF key.) If parties can get $[v]_{\Delta^{(c)}}$ and $[\alpha]_{\Delta^{(c)}}$, since msg_1 and msg_2 are public, parties can directly use a VOLE-based ZK to prove that msg_2 is equal to $\alpha^{2^\lambda} \cdot (\mathsf{msg}_1 + v)$. In other words, with $[v]_{\Delta^{(c)}}$, $[\alpha]_{\Delta^{(c)}}$ and msg_1, parties can generate $[\mathsf{msg}_2 = \alpha^{2^\lambda} \cdot (\mathsf{msg}_1 + v)]_{\Delta^{(c)}}$; then $\mathcal{P}_s$ can bindingly open msg_2 to $\mathcal{P}_c$. While this blueprint is relatively straightforward, there are several challenges to tackle:

1. $[v]_{\Delta^{(c)}}$ needs to be generated correctly, even with a malicious $\mathcal{P}_s$. Note, v is not an arbitrary value but rather is part of the VOLE correlation from $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$, used by our half-malicious protocol. That is, we not only need $\mathcal{P}_s$ to commit to v but to commit to a consistent v. More importantly, recall that $v = w + uk$ where u will be used as a one-time pad during the OPRF evaluation; hence, we must generate $[v]_{\Delta^{(c)}}$ without revealing any information correlated to u.
2. $[\alpha]_{\Delta^{(c)}}$ needs to be generated. This is easy as α is only used to protect $\mathcal{P}_s$'s OPRF key. In fact, $[\alpha]_{\Delta^{(c)}}$ can be directly generated as one correlation from $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ since the committed value α is pseudorandom.
3. Generating $[\alpha^{2^\lambda}]_{\Delta^{(c)}}$ from $[\alpha]_{\Delta^{(c)}}$ requires $\lambda + 3$ field elements of communication: λ of them are used to commit to the intermediate results (i.e., $[\alpha^{2^1}]_{\Delta^{(c)}}$, $[\alpha^{2^2}]_{\Delta^{(c)}}$, …, $[\alpha^{2^\lambda}]_{\Delta^{(c)}}$), and 3 of them

are used to deploy the batched LPZK technique (see Section 2.7) to ensure each squaring is computed correctly.

We could further reduce this cost via the generalized LPZK technique (see Section 2.7). Namely, w.l.o.g., for any constant ϕ that divides λ, $\mathcal{P}_s$ commits to $\frac{\lambda}{\phi}$ intermediate results as $[\alpha^{2^\phi}]_{\Delta^{(c)}}$, $[\alpha^{2^{2\phi}}]_{\Delta^{(c)}}$, ..., $[\alpha^{2^\lambda}]_{\Delta^{(c)}}$ and proves in ZK that each 2^ϕ-powering is computed correctly. Now, the communication cost to generate $[\alpha^{2^\lambda}]_{\Delta^{(c)}}$ is reduced to $\frac{\lambda}{\phi} + 2^\phi + 1$ field elements.

Generating Consistent $[v]_{\Delta^{(c)}}$. We now show how to tackle Challenge 1. Recall that $v = w + uk$ where v, k are known by $\mathcal{P}_s$ and u, w are known by $\mathcal{P}_c$. We aim to generate $[v]_{\Delta^{(c)}}$. We first present an ineffective yet insightful approach. $\mathcal{P}_s$ commits to $[v]_{\Delta^{(c)}}$ and $[k]_{\Delta^{(c)}}$; then $\mathcal{P}_c$ reveals u and $w = v - uk$ and requires $\mathcal{P}_s$ to show that $[v]_{\Delta^{(c)}} - w - u[k]_{\Delta^{(c)}} = [v - w - uk]_{\Delta^{(c)}}$ commits to zero. We now argue why this check ensures a correct $[v]_{\Delta^{(c)}}$ and $[k]_{\Delta^{(c)}}$. Assume $\mathcal{P}_s$ committed to $[\widetilde{v} \neq v]_{\Delta^{(c)}}$ or $[\widetilde{k} \neq k]_{\Delta^{(c)}}$. Then, if the IT-MAC $[\widetilde{v}]_{\Delta^{(c)}} - w - u[\widetilde{k}]_{\Delta^{(c)}}$ commits to zero, based on the binding property of IT-MAC, we know w.h.p. $\widetilde{v} - w - u\widetilde{k} = 0$. This implies $\widetilde{v} - u\widetilde{k} = v - uk$. If $k = \widetilde{k}$, then $v = \widetilde{v}$. Otherwise, we have $u = (v - \widetilde{v})/(k - \widetilde{k})$. This means that $\mathcal{P}_s$ can successfully guess u *before* u is revealed. Note that since u is uniformly sampled when $\mathcal{P}_c$ is honest by $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ (see Figure 1), this can only happen with probability $\frac{1}{p}$. However, although this approach guarantees a well-formed $[v]_{\Delta^{(c)}}$, it is ineffective. This is because u is revealed and, crucially, is later used as a one-time pad.

Therefore, we must ensure a well-formed $[v]_{\Delta^{(c)}}$ without revealing any information correlated to u (and w). We notice that this can be done by exploiting a random linear combination to "sacrifice" another u_{sa}, which will be discarded immediately after the check to ensure u remains full entropy. In detail, let parties hold another VOLE correlation from $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$: $v_{\mathsf{sa}} = w_{\mathsf{sa}} + u_{\mathsf{sa}}k$ where v_{sa}, k are known by $\mathcal{P}_s$ and $u_{\mathsf{sa}}, w_{\mathsf{sa}}$ are known by $\mathcal{P}_c$. Now, let $\mathcal{P}_s$ commit to $[v_{\mathsf{sa}}]_{\Delta^c}$, $[v]_{\Delta^c}$ and $[k]_{\Delta^c}$. Then, $\mathcal{P}_c$ samples a uniform $\chi \xleftarrow{\$} \mathbb{F}_p$ and reveals $\chi, \chi u + u_{\mathsf{sa}}$ and $\chi w + w_{\mathsf{sa}}$. Finally, $\mathcal{P}_c$ requires $\mathcal{P}_s$ to show that the following IT-MAC

$$\begin{aligned}&\chi[v]_{\Delta^{(c)}} + [v_{\mathsf{sa}}]_{\Delta^{(c)}} - (\chi w + w_{\mathsf{sa}}) - (\chi u + u_{\mathsf{sa}})[k]_{\Delta^{(c)}}\\ =&[\chi(v - w - uk) + (v_{\mathsf{sa}} - w_{\mathsf{sa}} - u_{\mathsf{sa}}k)]_{\Delta^{(c)}}\end{aligned}$$

commits to zero. Note, this can be viewed as a consistency check over the value $\chi v + v_{\mathsf{sa}}$. That is, it ensures that $\chi\widetilde{v} + \widetilde{v_{\mathsf{sa}}} = \chi v + v_{\mathsf{sa}}$ and $\widetilde{k} = k$, where $\widetilde{v}, \widetilde{v_{\mathsf{sa}}}, \widetilde{k}$ are arbitrary (potentially inconsistent) values committed by $\mathcal{P}_s$. Moreover, since χ is sampled and revealed only after $\widetilde{v}, \widetilde{v_{\mathsf{sa}}}$ are chosen, $\chi\widetilde{v} + \widetilde{v_{\mathsf{sa}}} = \chi v + v_{\mathsf{sa}}$ implies $\widetilde{v} = v$ and $\widetilde{v_{\mathsf{sa}}} = v_{\mathsf{sa}}$ with overwhelming probability based on the well-known *Schwartz-Zippel* lemma [Sch80, Zip79]. Crucially, the values revealed by $\mathcal{P}_c$ (i.e., $\chi, \chi u + u_{\mathsf{sa}}$ and $\chi w + w_{\mathsf{sa}}$) are *independent* of u, so u remains full entropy.

One essential remark is that we also need to ensure a well-formed $\chi, \chi u + u_{\mathsf{sa}}$ and $\chi w + w_{\mathsf{sa}}$ for a malicious $\mathcal{P}_c$. These new messages also provide a new opportunity for a malicious $\mathcal{P}_c$ to exploit—indeed, a malicious $\mathcal{P}_c$ can learn the OPRF key with ill-formed messages. Interestingly, this can be resolved as follows: recall that u and u_{sa} can also be viewed as committed values inside IT-MAC but rather from $\mathcal{P}_c$ to $\mathcal{P}_s$; thus, $\chi w + w_{\mathsf{sa}}$ can be used as a proof to force correct opening of $\chi u + u_{\mathsf{sa}}$. That is, an honest $\mathcal{P}_s$ can check if $\chi v + v_{\mathsf{sa}}$ is equal to $\chi w + w_{\mathsf{sa}} + (\chi u + u_{\mathsf{sa}})k$.

We emphasize that this check relies on k with enough entropy *to the UC environment*. Hence, when k is input from $\mathcal{P}_s$, this does not work. Instead, parties can perform the consistency check with the committed $\Delta^{(s)}$ rather than k (see Footnote 8 and Section 5.3). However, note that $\Delta^{(s)}$ will also be revealed to the UC environment (1) as $\mathcal{P}_s$'s output when the OPRF uses a uniformly

sampled key, or (2) from $k^* - \Delta^{(s)}$ when the OPRF uses a server-specified key. Therefore, the step of generating consistent $[v]_{\Delta^{(c)}}$ must take place before the above step (1) or (2) is performed. This can be easily enforced if parties only need to execute the OPRF once but it requires additional care in other cases, e.g., batched offline phases presented in Section 3.3.

Finally, we remark that $[k]_{\Delta^{(c)}}$ is an IT-MAC of the OPRF key and thus can be used to prove any NP relation over it using regular VOLE-based ZK. For example, it can be used to prove in ZK that the server reuses the same committed OPRF key across different invocations as how we do to achieve verifiable OPRFs in Section 6.1.

Batched Evaluations. Our malicious protocol can be extended to the batched variant naturally using more VOLE correlations in both directions (i.e., $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ and $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$). For n-batched evaluations, the step to generate IT-MACs of $[v_1]_{\Delta^{(c)}}, \ldots, [v_n]_{\Delta^{(c)}}$ can be improved (i.e., addressing Challenge 1) by "sacrificing" a single correlation. Namely, after $\mathcal{P}_s$ commits to $\boldsymbol{v}$, $\mathcal{P}_c$ can reveal $\chi \xleftarrow{\$} \mathbb{F}_p$, $\left(\sum_{i=1}^{n} \chi^i u_i\right) + u_{\mathsf{sa}}$ and $\left(\sum_{i=1}^{n} \chi^i w_i\right) + w_{\mathsf{sa}}$, reducing the communication of this step from $6n+1$ to $n+6$ field elements. Note that the $\Delta^{(s)}$ must remain full entropy to the UC environment at this point. Furthermore, the batched (generalized) LPZK can be applied across n evaluations to generate $[\alpha^{2^\lambda}_{i\in[n]}]_{\Delta^{(c)}}$ and prove the correct $\mathsf{msg}_{2,i}$.

3.3 Offline-Online Mode

Our protocols support an *offline-online* mode where most portions of computation and communication can be pushed into an *input-independent* (and PRF-key-*independent*) offline (aka preprocessing) phase. Clearly, the generation of (V)OLE correlations is a hybrid input-independent functionality, so it can be pushed into the offline phase. This is true even when $\mathcal{P}_s$ wants to select its own OPRF key k^*, since this can be done via sending $k^* - \Delta^{(s)}$ in the online phase.

Moreover, for our malicious protocol, the parts generating $[v]_{\Delta^{(c)}}$ and $[\alpha^{2^\lambda}]_{\Delta^{(c)}}$ can also be pushed into the offline phase. Again, this is true even when the $\mathcal{P}_s$ wants to select its own OPRF key k^*. Crucially, this does *not* affect the ability to obtain the committed key. That is, the parties in the offline phase would obtain $[\Delta^{(s)}]_{\Delta^{(c)}}$, which can be locally adjusted to $[k^*]_{\Delta^{(c)}}$ via $k^* - \Delta^{(s)}$. The online phase cost of our malicious protocol is very lean compared to the online phase cost of our half-malicious protocol: it only needs 3 more field elements (i.e., a batched LPZK), regardless of n.

In all, our protocols are very efficient in the amortized offline-online setting and reasonably efficient even *with* offline cost included. See Section 7.

Batched Offline Phases. Our protocols support batched offline phases. That is, offline phases of n_{batch} evaluations can be prepared together for better amortized costs. Then, each n-sub-batched evaluation—where various values of n add up to n_{batch}—can be processed immediately without needing the entire n_{batch} batch, paying an n-amortized online phase. Note that once the online phase starts, the UC environment learns $\Delta^{(s)}$, making the consistency check in our malicious 2PC-Gold no longer UC-secure; see Section 3.2. Thus, in our malicious OPRFs, n_{batch} must be predetermined, and the batched offline phases must be completed in advance. We formalize batched offline and sub-batched online phases as our ideal functionality in Section 5.1.

In certain applications, fixing n_{batch} in advance is either impractical or impossible. In Section 6.1, we demonstrate how to remove this limitation easily via *private verifiability*.

3.4 Eliminating 1-bit Leakage in Our Protocols

The protocols presented in Sections 3.1 and 3.2 incur a 1-bit leakage per evaluation. In particular, for each evaluation, $\mathcal{P}_s$ learns $u \cdot (k + x)$, which allows her to determine whether $k + x$ equals zero.

Namely, our ideal functionality to capture these protocols has to leak this 1-bit information to the simulator if $\mathcal{P}_s$ is corrupted. Note, if we consider UC security, this 1-bit leakage is unavoidable even for a semi-honest $\mathcal{P}_s$ with a uniformly sampled key. This is because the environment can always set the honest $\mathcal{P}_c$'s input as $-k$.

While this leakage may not be a problem for many applications (in particular, this leakage also exists in Dodis-Yampolskiy PRF [DY05] and some corresponding OPRFs), we now demonstrate how to eliminate this leakage, depending on whether we need an OPRF with a uniformly sampled key or one specified by $\mathcal{P}_s$.

Uniformly Sampled Key. If only a uniformly sampled PRF key is needed, we first show how to enforce a malicious $\mathcal{P}_s$ to use a uniformly sampled key. Note, in $\mathcal{F}_{\mathsf{VOLE}}$ (see Figure 1), the malicious receiver (i.e., $\mathcal{P}_s$) is allowed to select an arbitrary scalar $\Delta^{(s)}$ (i.e., an arbitrary PRF key). Our key observation is that this scalar $\Delta^{(s)}$ can be randomized by $\mathcal{P}_c$. That is, $\mathcal{P}_c$ can send a uniformly sampled $\Delta' \xleftarrow{\$} \mathbb{F}_p$ to set the scalar as $\Delta^{(s)} + \Delta'$—which will be used as the PRF key that is guaranteed to be uniformly sampled—and adjust his VOLE shares locally.

By enforcing a uniformly sampled key k, this leakage can be naturally eliminated in the standalone setting. However, in the UC setting, the leakage persists because the environment can always set x as $-k$ after learning k. To further eliminate it in the UC setting, we can let $\mathcal{P}_c$ hash the input before executing the OPRF protocol(s). Intuitively, this prevents the leakage as the environment cannot find an x s.t. $\mathsf{H}(x) = -k$ when $\mathsf{H}(\cdot)$ is modeled as a random oracle.

Server-Specified Key. If we need an OPRF where the key is fully specified by $\mathcal{P}_s$, the previously mentioned fix does not apply. Instead, we can eliminate the leakage by restricting the PRF key to be chosen from elements in $\mathbb{F}_p$ that have two leading zero bits (i.e., the two most significant bits are 0s) and by limiting the PRF input to the subset of $\mathbb{F}_p$ consisting of all non-zero elements with two leading zero bits. In essence, this ensures that $k + x$ is never zero.

If $\mathcal{P}_s$ is malicious, we can further require $\mathcal{P}_s$ to provide a VOLE-based ZK proof over IT-MAC $[k^*]_{\Delta^{(c)}}$ to show it has two leading zero bits. This proof requires communicating $\mathcal{O}(\lambda)$ field elements and performing $\mathcal{O}(\lambda)$ field operations. It can be amortized in batched evaluations.

Note that this fix requires a minor modification of the underlying PRF assumption to sample the key accordingly; see Section 4.1.

4 Gold PRF Basics

4.1 Formal Hardness Assumptions

We define the underlying computation problem—the Decisional Shifted Power-Residue Symbol—as follows:

Definition 1 (Decisional Shifted Power-Residue Symbol (DSPRS) Problem)**.** *Let $p = p(\lambda)$ be a family of prime numbers, where each $p = e(\lambda) \cdot g(\lambda) + 1$. Let k be uniformly sampled from $\mathbb{F}_p$. Let $\mathcal{D} := \{a^g \bmod p \mid a \in \mathbb{F}_p^*\}$. Let O_{PR} be an oracle that on input $x \in \mathbb{F}_p$ outputs $(x + k)^g \bmod p$, and O_{R} be a random oracle that maps elements from $\mathbb{F}_p$ to $\mathcal{D}$. The DSPRS problem asks to distinguish between O_{PR} and O_{R} given 1^λ and p, g, with classical queries.*

SolveEq$(1^\lambda, p, g, a)$

1 : Apply Fact 2 to get a generator $h = r^g$ of $\mathbb{G}$
2 : Apply Fact 1 to get z such that $h^z = a$
3 : **return** r^z

Figure 4: An efficient algorithm to find a solution X of the equation $X^g \equiv a \pmod p$ where $a \in \mathbb{G}$, $\mathbb{G} := \{x^g \bmod p \mid x \in \mathbb{F}_p^*\}$, and $p = 2^\lambda \cdot g + 1$ is a prime of length $\mathcal{O}(\lambda)$. Correctness comes from: $h^z = (r^g)^z = (r^z)^g = a$.

The post-quantum security of our Gold PRF relies on the following hardness assumption:

Assumption 1. *For any $p = p(\lambda) = e(\lambda) \cdot g(\lambda) + 1$, where $e(\lambda)$ is smooth and $g(\lambda) = \Omega(\lambda)$, there is no probabilistic polynomial time (quantum) algorithm for the DSPRS problem (Definition 1) with non-negligible advantage in λ.*

In this work, we set $e = 2^\lambda$ and $\log g = 2\lambda + \mathcal{O}(1)$. Gold is essentially O_{PR} in Definition 1. Existing attacks presented in Section 2.5 guide this choice. These parameters are very conservative—the best-known quantum attack for Gold by [BBUV20] has time complexity $\mathcal{O}(\frac{g}{L})$ using $L \le p^{1/4}$ queries; the best-known quantum attack for the Legendre PRF over p (which also applies for Gold) by [FS21] has time complexity $2^{\mathcal{O}(\log p)} \cdot p^{1/3}$ using $p^{1/3}$ queries.

Finally, we note that a minor modification of Definition 1 is needed to eliminate the one-bit leakage when the OPRF key is chosen by $\mathcal{P}_s$ (see Section 3.4). Namely, we only need to sample the PRF key from the subspace of $\mathbb{F}_p$ consisting of all elements with two leading zero bits.

4.2 Power-Residue Subgroup

Let $p = 2^\lambda \cdot g + 1$ be a prime of $\mathcal{O}(\lambda)$ bits. We review several useful properties of the g-th power-residue subgroup $\mathbb{G}$ of the multiplicative group $\mathbb{F}_p^*$ of $\mathbb{F}_p$. That is,

$$\mathbb{G} := \{x^g \bmod p \mid x \in \mathbb{F}_p^*\}$$

where the group operation is multiplication modulo p.

Fact 1. *$\mathbb{G}$ is a finite cyclic group of order 2^λ. Let h be a generator of $\mathbb{G}$. Then, for any element $x \in \mathbb{G}$, the Pohlig-Hellman algorithm [PH78] can solve the discrete logarithm (DLOG) of x in base h in $O(\lambda^2)$ group operations.*

Fact 2. *There exist $2^{\lambda-1}$ generators in $\mathbb{G}$. In particular, there exists an efficient way to find a generator h: repeatedly sample $r \xleftarrow{\$} \mathbb{F}_p^*$, and set $h := r^g$, until $h^{2^{\lambda-1}} \neq 1$.*

Converting Output. Fact 1 implies a straightforward deterministic way to transfer the output of 2PC-Gold from $\mathbb{G}$ to $\{0,1\}^\lambda$. I.e., $\mathcal{P}_c$ can locally solve the DLOG of 2PC-Gold output to a publicly agreed base h. Note that when $k + x = 0$, $\mathcal{P}_c$ will abort, so there is no need to solve the DLOG. Of course, there exist other standard ways to post-process output, such as applying a hash function over it, in particular, if more than λ bits of output are required (e.g., O-Gold).

Solving Equation $X^g \equiv a \pmod p$. Facts 1 and 2 can be exploited to design an efficient algorithm (defined in Figure 4) to find a solution X of the equation $X^g \equiv a \pmod p$ for some $a \in \mathbb{G}$. This algorithm is essential for arguing the security of our protocols: the simulator uses it.

Functionality $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$

$\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$, parameterized by a field $\mathbb{F}_p$ where (1) $p = p(\lambda) = 2^\lambda \cdot g + 1$ is a prime and (2) $g = g(\lambda)$ is an integer of $2\lambda + O(1)$ bits, running with a server $\mathcal{P}_s$, a client $\mathcal{P}_c$ and an adversary $\mathcal{S}$, proceeds as follows:

Initialize. Upon receiving $(\texttt{init}, n_{\mathsf{batch}})$ from $\mathcal{P}_s$ and $\mathcal{P}_c$ where $n_{\mathsf{batch}} \in \mathbb{Z}^+$, if $\mathcal{P}_s$ is honest (or semi-honest), sample $k \xleftarrow{\$} \mathbb{F}_p$, else receive k from $\mathcal{S}$. Store n_{batch} and k. Send k to $\mathcal{P}_s$. Set $n_{\mathsf{eval}} := 0$. Ignore subsequent $(\texttt{init}, \cdot)$.

Evaluate. Upon receiving $(\texttt{eval}, n)$ from $\mathcal{P}_s$ and $(\texttt{eval}, n, x_1, x_2, \ldots, x_n)$ from $\mathcal{P}_c$, where $n \in \mathbb{Z}^+$ and each $x_{i\in[n]} \in \mathbb{F}_p$. If $n_{\mathsf{eval}} + n > n_{\mathsf{batch}}$, ignore the instruction; otherwise, set $n_{\mathsf{eval}} := n_{\mathsf{eval}} + n$ and proceed as follows:

1. For each $i \in [n]$, compute $y_i := (k + x_i)^g$ over $\mathbb{F}_p$.
2. (**One-bit leakage**) If $\mathcal{P}_s$ is corrupted, for each $i \in [n]$, compute $\ell_i := (k + x_i) \stackrel{?}{=} 0$, send $(\texttt{eval}, \boldsymbol{\ell})$ to $\mathcal{S}$. If $\exists i \in [n]$ such that ℓ_i is 1, send $(\texttt{abort})$ to $\mathcal{P}_c$ and halt.
3. Send $(\texttt{eval}, \boldsymbol{y})$ to $\mathcal{P}_c$.

Figure 5: The 2PC-Gold functionality.

5 Formalization and Analysis of 2PC-Gold

We formalize our 2PC-Gold protocols in the UC framework. Our other OPRFs and their corresponding ideal functionalities, theorems, and proofs can be derived through straightforward modifications or, in the case of UC-Gold in section 6.1, via the elegant transformation from [BDFH24].

5.1 Ideal Functionality for 2PC-Gold

The ideal functionality $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ achieved by our 2PC-Gold is defined in Figure 5. For simplicity, we assume the PRF key is uniformly sampled when $\mathcal{P}_s$ is honest. We remark that our protocols can support a server-specified key almost for free (see Section 5.3). We directly define $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ (and realizations) with batched offline phases (i.e., `init` instruction) and sub-batched online phases (i.e., `eval` instruction). The non-batched evaluation setting is the particular case where $n_{\mathsf{batch}} = n = 1$.

Note that `init` can be called only once per session with a predetermined n_{batch}, capturing the maximum number of evaluations (if more are needed `init` needs to be called again). See Section 3.3 for the reason behind this restriction and Section 6.1 for how to remove it easily. Our half-malicious 2PC-Gold in fact does not even have this restriction. Specifically, Step 3 in our protocol in Figure 6 can be executed repeatedly, even after the online phase has been performed. However, for simplicity, since our plain malicious protocol has this restriction on `init` (specifically Step 5 cannot be executed again after `init` ends), we adopt a unified $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ that restricts `init` in all settings.

One-Bit Leakage. Recall that protocols presented in Section 3 incur a one-bit leakage per evaluation. We show how to remove this leakage efficiently in Section 3.4. Since this functionality (with the one-bit leakage in Step 2) is already useful for many applications (justified below and in Section 3.4), we stick to this definition. Essentially, the changes to upgrade the functionality/protocols/proofs to the version without the one-bit leakage are inexpensive.

One-Bit Leakage OPRF Applications. For some applications, the one-bit leakage is harmless, for example:

1. **PSI, where $\mathcal{P}_s$ is semi-honest:** In the case of OPRF-based PSI where the server is semi-honest, the server computes an OPRF with the client over the client's set elements. Since the server uses a uniformly random key, it is unlikely to gain additional information via collision. Indeed, a

recent PSI work [SKR+24] bases its protocols on an OPRF (not Gold) that features the same type of leakage, indicating that our OPRF with one-bit leakage can directly instantiate their PSI protocols.

2. **Password-based protocols, where $\mathcal{P}_s$ is malicious:** In the case of password-based protocols from OPRF (e.g., OPAQUE [JKX18]), the client computes OPRF over its password and then stores certain public functions of this value on the server. Since $\mathcal{P}_s$ can simply test the client's password locally, the one-bit leakage is harmless.

5.2 Formal 2PC-Gold Protocols and Theorems

We defer the reader to Section 3 for concise overviews of 2PC-Gold, including intuitive arguments regarding security. In this section, we formalize our protocols as $\Pi_{\mathsf{2PC\text{-}Gold}}$ in Figures 6 and 7 with corresponding theorems.

Theorem 1 (Half-Malicious)**.** *Protocol $\Pi_{\mathsf{2PC\text{-}Gold}}$ (Figures 6 and 7 without gray boxes) information-theoretically UC-realizes $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ (Figure 5) in the $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$-hybrid model (Figure 1), in the presence of static corruptions, where a corrupt $\mathcal{P}_s$ can be semi-honest and a corrupt $\mathcal{P}_c$ can be malicious.*

Proof. By constructing the simulators. When both $\mathcal{P}_s$ and $\mathcal{P}_c$ are honest, simulation is trivial and the ideal world is statistically indistinguishable from the real world: it only fails when $u_i = 0$ for one $i \in [n_{\mathsf{batch}}]$, which happens with probability $\leq \frac{n_{\mathsf{batch}}}{p}$. We now focus on the cases where either $\mathcal{P}_s$ or $\mathcal{P}_c$ is corrupted. In these cases, the simulation is perfect.

Semi-honest $\mathcal{P}_s$: The simulator $\mathcal{S}$ only needs to sample the (honest) transcript seen by $\mathcal{P}_s$. Clearly, simulating the call to `init` is trivial as $\mathcal{S}$ only needs to emulate $\mathcal{F}_{\mathsf{VOLE}}$. Additionally, we can sample each call to `eval` separately, as long as we carefully reuse the same $k = \Delta^{(s)}$ for each call. Thus, we only focus on the scenario with a single `eval`. The transcript in the real world is:

$$\{k, n, \boldsymbol{v}, -\boldsymbol{w} + \boldsymbol{u} \odot \boldsymbol{x}\} \mid k \xleftarrow{\$} \mathbb{F}_p, \boldsymbol{v}, \boldsymbol{u} \xleftarrow{\$} \mathbb{F}_p^n, \text{ s.t. } \boldsymbol{v} = \boldsymbol{w} + \boldsymbol{u} \cdot k$$

where $\boldsymbol{x}$ is the $\mathcal{P}_c$'s input and $\odot$ denotes the element-wise product. As $-\boldsymbol{w} = -\boldsymbol{v} + \boldsymbol{u} \cdot k$, by replacing $\boldsymbol{w}$ and adding $\boldsymbol{v}$ into the fourth entry, we need to sample:

$$\{k, n, \boldsymbol{v}, \boldsymbol{u} \odot (k + \boldsymbol{x})\} \mid k \xleftarrow{\$} \mathbb{F}_p, \boldsymbol{v}, \boldsymbol{u} \xleftarrow{\$} \mathbb{F}_p^n$$

Note that $\mathcal{S}$ gets $\ell_i := (k + x_i) \stackrel{?}{=} 0$ for each $i \in [n]$ from the $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$.[9] Therefore, for each $i \in [n]$, if $\ell_i = 1$, we can replace $u_i(k + x_i)$ with a zero; otherwise, since u_i is uniformly sampled and unknown to $\mathcal{P}_s$, we can replace $u_i(k + x_i)$ with a uniformly sampled $\mathbb{F}_p$ element. That is,

$$\{k, n, \boldsymbol{v}, \boldsymbol{r} \odot (\boldsymbol{1} - \boldsymbol{\ell})\} \mid k \xleftarrow{\$} \mathbb{F}_p, \boldsymbol{v} \xleftarrow{\$} \mathbb{F}_p^n, \boldsymbol{r} \xleftarrow{\$} \mathbb{F}_p^n$$

The final distribution is sampled *without* $\boldsymbol{x}$.

Malicious $\mathcal{P}_c^*$: The simulator $\mathcal{S}$ needs to interact with the malicious $\mathcal{P}_c^*$ to sample the transcript *and* extract the inputs. Similar to the semi-honest $\mathcal{P}_s$ case, we only need to focus on the scenario with a single `eval`. $\mathcal{S}$, emulating $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ for $\mathcal{P}_c^*$, will perform as follows:

[9] We note that in the protocols where the leakage is eliminated, $\boldsymbol{\ell}$ can be viewed as $\boldsymbol{0}$—a vector of zero. See Section 3.4.

Protocol $\Pi_{\mathsf{2PC\text{-}Gold}}$ (Offline)

$\Pi_{\mathsf{2PC\text{-}Gold}}$, parameterized by a field $\mathbb{F}_p$ where (1) $p = p(\lambda) = 2^\lambda \cdot g + 1$ is a prime and (2) $g = g(\lambda)$ is an integer of $2\lambda + O(1)$ bits, running with a server $\mathcal{P}_s$, a client $\mathcal{P}_c$, with hybrid access to $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ and $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ over $\mathbb{F}_p$, proceeds as follows:

Initialize. $\mathcal{P}_s$ and $\mathcal{P}_c$, each on receiving $(\mathtt{init}, n_{\mathsf{batch}})$ where $n_{\mathsf{batch}} \in \mathbb{Z}^+$, proceed as follows:

1. **Sample OPRF key:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{init})$ to $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$, where $\mathcal{P}_s$ receives the uniformly sampled OPRF key $k := \Delta^{(s)} \in \mathbb{F}_p$.
2. **Sample ZKP private coins:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{init})$ to $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, where $\mathcal{P}_c$ receives the uniformly sampled $\Delta^{(c)} \in \mathbb{F}_p$.
3. **Sample VOLE correlations used for OPRF evaluations:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, n_{\mathsf{batch}})$ to $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$, and then $\mathcal{P}_s$ receives $\boldsymbol{v}$ whereas $\mathcal{P}_c$ receives $\boldsymbol{u}, \boldsymbol{w}$ such that $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{F}_p^{n_{\mathsf{batch}}}, \boldsymbol{v} = \boldsymbol{w} + \boldsymbol{u}\Delta^{(s)}$.
4. **Prepare the committed g-th residues:**
   (a) **Sample the hiding pads:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, n_{\mathsf{batch}})$ to $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, and then parties receive length-n_{batch} random IT-MACs as $[\boldsymbol{\alpha}]_{\Delta^{(c)}}$. (Note: when $\mathcal{P}_s$ is honest, $\boldsymbol{\alpha}$ looks uniform to $\mathcal{P}_c$.)
   (b) **Sample the VOLE correlations used for raising powers:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, n_{\mathsf{batch}} \cdot \lambda)$ to $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, and then parties receive length-$(n_{\mathsf{batch}} \cdot \lambda)$ random IT-MACs as $[\delta_{i,j}]_{\Delta^{(c)}}$ for each $i \in [n_{\mathsf{batch}}], j \in [\lambda]$.
   (c) **Raising powers:** for each $i \in [n_{\mathsf{batch}}], j \in [\lambda]$, $\mathcal{P}_s$ sends $\alpha_i^{2^j} - \delta_{i,j}$, then parties compute $[\alpha_i^{2^j}]_{\Delta^{(c)}} := [\delta_{i,j}]_{\Delta^{(c)}} + (\alpha_i^{2^j} - \delta_{i,j})$.
   (d) **ZKP check on raising powers:** $\mathcal{P}_s$ and $\mathcal{P}_c$ deploy the batched LPZK check (see Section 2.7) to ensure for each $i \in [n_{\mathsf{batch}}], j \in [\lambda]$, $[\alpha_i^{2^{j-1}}]_{\Delta^{(c)}}$ and $[\alpha_i^{2^j}]_{\Delta^{(c)}}$ satisfy the relation: $\left(\alpha_i^{2^{j-1}}\right)^2 = \alpha_i^{2^j}$. Note, this needs a random VOLE correlation from $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$. If the check fails, $\mathcal{P}_c$ aborts the protocol.
5. **Prepare the committed $\boldsymbol{v}$:**
   (a) **Commit the OPRF key $k = \Delta^{(s)}$:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, 1)$ to $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, and then parties receive length-1 random IT-MACs as $[\beta]_{\Delta^{(c)}}$. Next, $\mathcal{P}_s$ sends $\Delta^{(s)} - \beta$ to $\mathcal{P}_c$, then parties compute $[k = \Delta^{(s)}]_{\Delta^{(c)}} := [\beta]_{\Delta^{(c)}} + (\Delta^{(s)} - \beta)$.
   (b) **Commit $\boldsymbol{v}$:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, n_{\mathsf{batch}})$ to $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, and then parties receive length-n_{batch} random IT-MACs as $[\boldsymbol{\zeta}]_{\Delta^{(c)}}$. Next, for each $i \in [n_{\mathsf{batch}}]$, $\mathcal{P}_s$ sends $v_i - \zeta_i$ to $\mathcal{P}_c$, then parties compute $[v_i]_{\Delta^{(c)}} := [\zeta_i]_{\Delta^{(c)}} + (v_i - \zeta_i)$.
   (c) **Sample the sacrificed VOLE correlation:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, 1)$ to $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$, and then $\mathcal{P}_s$ receives v_{sa} whereas $\mathcal{P}_c$ receives $u_{\mathsf{sa}}, w_{\mathsf{sa}}$ such that $v_{\mathsf{sa}}, u_{\mathsf{sa}}, w_{\mathsf{sa}} \in \mathbb{F}_p, v_{\mathsf{sa}} = w_{\mathsf{sa}} + u_{\mathsf{sa}}\Delta^{(s)}$.
   (d) **Commit v_{sa}:** $\mathcal{P}_s$ and $\mathcal{P}_c$ each sends $(\mathtt{extend}, 1)$ to $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, and then parties receive length-1 random IT-MACs as $[\gamma]_{\Delta^{(c)}}$. Next, $\mathcal{P}_s$ sends $v_{\mathsf{sa}} - \gamma$ to $\mathcal{P}_c$, then parties compute $[v_{\mathsf{sa}}]_{\Delta^{(c)}} := [\gamma]_{\Delta^{(c)}} + (v_{\mathsf{sa}} - \gamma)$.
   (e) **Perform the consistency check:** $\mathcal{P}_c$ samples $\chi \in \mathbb{F}_p$, then computes $u_{\mathsf{poly}} := u_{\mathsf{sa}} + \sum_{i=1}^{n_{\mathsf{batch}}} u_i \cdot \chi^i$ and $w_{\mathsf{poly}} := w_{\mathsf{sa}} + \sum_{i=1}^{n_{\mathsf{batch}}} w_i \cdot \chi^i$. Next, $\mathcal{P}_c$ sends $\chi, u_{\mathsf{poly}}, w_{\mathsf{poly}}$ to $\mathcal{P}_s$, and then $\mathcal{P}_s$ computes $v_{\mathsf{poly}} := v_{\mathsf{sa}} + \sum_{i=1}^{n_{\mathsf{batch}}} v_i \cdot \chi^i$; $\mathcal{P}_s$ checks if $v_{\mathsf{poly}} \stackrel{?}{=} w_{\mathsf{poly}} + u_{\mathsf{poly}}\Delta^{(s)}$, if not, $\mathcal{P}_s$ aborts the protocol. Finally, parties *locally* compute the IT-MAC

$$\begin{aligned}[v_{\mathsf{poly}} - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot \Delta^{(s)}]_{\Delta^{(c)}} &= [v_{\mathsf{poly}}]_{\Delta^{(c)}} - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot [\Delta^{(s)}]_{\Delta^{(c)}} \\ &= [v_{\mathsf{sa}}]_{\Delta^{(c)}} + \sum_{i=1}^{n} \chi^i \cdot [v_i]_{\Delta^{(c)}} - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot [\Delta^{(s)}]_{\Delta^{(c)}}\end{aligned}$$

   and $\mathcal{P}_s$ proves to $\mathcal{P}_c$ that it commits to a zero via opening the IT-MAC. If the check fails, $\mathcal{P}_c$ aborts the protocol.

Finally, $\mathcal{P}_s$ and $\mathcal{P}_c$ each sets $n_{\mathsf{eval}} := 0$, ignoring the subsequent $(\mathtt{init}, \cdot)$. In the half-malicious protocol, $\mathcal{P}_s$ samples $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}_p^{n_{\mathsf{batch}}}$ and computes $\alpha_i^{2^\lambda}$ for each $i \in [n_{\mathsf{batch}}]$ (in the malicious case, $\boldsymbol{\alpha}$ is sampled and $\boldsymbol{\alpha}^{2^\lambda}$ is computed in Sub-step 4a). Then, $\mathcal{P}_s$ outputs k.

Figure 6: Our 2PC-Gold protocols (offline phase) in the $(\mathcal{F}_{\mathsf{VOLE}}^{c,s}, \mathcal{F}_{\mathsf{VOLE}}^{s,c})$-hybrid model. The descriptions include our half-malicious and malicious protocols. In particular, the malicious protocol includes the steps (and corresponding sub-steps) in gray boxes.

Protocol $\Pi_{\mathsf{2PC\text{-}Gold}}$ (Online)

$\Pi_{\mathsf{2PC\text{-}Gold}}$, parameterized by a field $\mathbb{F}_p$ where (1) $p = p(\lambda) = 2^\lambda \cdot g + 1$ is a prime and (2) $g = g(\lambda)$ is an integer of $2\lambda + O(1)$ bits, running with a server $\mathcal{P}_s$, a client $\mathcal{P}_c$, with hybrid access to $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ and $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ over $\mathbb{F}_p$, proceeds as follows:

Evaluate. $\mathcal{P}_s$ on receiving $(\texttt{eval}, n)$ and $\mathcal{P}_c$ on receiving $(\texttt{eval}, n, \boldsymbol{x})$, where $n \in \mathbb{Z}^+$ and $\boldsymbol{x} \in \mathbb{F}_p^n$. If $n_{\mathsf{eval}} + n > n_{\mathsf{batch}}$, ignore the instruction; otherwise, proceed as follows:

6. $\mathcal{P}_c$ **sends the first message:** For each $i \in [n]$, $\mathcal{P}_c$ sends $\mathsf{msg}_{1,i} := -w_{n_{\mathsf{eval}}+i} + u_{n_{\mathsf{eval}}+i} \cdot x_i$ to $\mathcal{P}_s$.
7. $\mathcal{P}_s$ **sends the second message:** For each $i \in [n]$, $\mathcal{P}_s$ sends $\mathsf{msg}_{2,i} := \alpha_{n_{\mathsf{eval}}+i}^{2^\lambda} \cdot (\mathsf{msg}_{1,i} + v_{n_{\mathsf{eval}}+i})$ to $\mathcal{P}_c$. If $\exists i \in [n], \mathsf{msg}_{2,i} = 0$, $\mathcal{P}_c$ aborts the protocol. (Note, the computing of $\boldsymbol{\alpha}^{2^\lambda}$ has been performed in the offline phase.)
8. **ZK proofs on a well-formed second message:** $\mathcal{P}_s$ and $\mathcal{P}_c$ deploy the batched LPZK check (see Section 2.7) to ensure for each $i \in [n]$, $[v_{n_{\mathsf{eval}}+i}]_{\Delta^{(c)}}$ (from Sub-step 5b), $[\alpha_{n_{\mathsf{eval}}+i}^{2^\lambda}]$ (from Sub-step 4c), $\mathsf{msg}_{1,i}$ (from Step 6), and $\mathsf{msg}_{2,i}$ (from Step 7) satisfy the relation: $\alpha_{n_{\mathsf{eval}}+i}^{2^\lambda} \cdot (\mathsf{msg}_{1,i} + v_{n_{\mathsf{eval}}+i}) = \mathsf{msg}_{2,i}$. Note, this needs a random VOLE correlation from $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$, which can be generated in the offline phase. If the check fails, $\mathcal{P}_c$ aborts the protocol.
9. $\mathcal{P}_c$ **computes the final output:** for each $i \in [n]$, $\mathcal{P}_c$ computes $y_i := \left(u_{n_{\mathsf{eval}}+i}^{-1} \cdot \mathsf{msg}_{2,i}\right)^g \in \mathbb{F}_p$. $\mathcal{P}_c$ outputs $(\texttt{eval}, \boldsymbol{y})$.

Finally, $\mathcal{P}_s$ and $\mathcal{P}_c$ each sets $n_{\mathsf{eval}} := n_{\mathsf{eval}} + n$.

Figure 7: Our 2PC-Gold protocols (online phase) in the $(\mathcal{F}_{\mathsf{VOLE}}^{c,s}, \mathcal{F}_{\mathsf{VOLE}}^{s,c})$-hybrid model. The descriptions include our half-malicious and malicious protocols. In particular, the malicious protocol includes the steps (and corresponding sub-steps) in gray boxes. Note that for the malicious `eval` with $n = 1$, it suffices to use (cheaper) non-batched LPZK technique in Step 8.

1. In Step 3, $\mathcal{P}_c^*$ would send $(\texttt{extend}, n_{\mathsf{batch}})$ to $\mathcal{S}$. Here, $\mathcal{S}$ emulates the $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ and receives $\boldsymbol{u}, \boldsymbol{w}$ from $\mathcal{P}_c^*$. $\mathcal{S}$ maintains a global counter n_{cur}, initialized to 0. For each call `eval` with n, $\mathcal{S}$ sets $n_{\mathsf{cur}} := n_{\mathsf{cur}} + n$ after completing the following steps.
2. **Extracting the inputs:** In Step 6, for each $i \in [n]$, $\mathcal{P}_c^*$ would send $\widetilde{\mathsf{msg}_{1,i}} \in \mathbb{F}_p$ to $\mathcal{S}$; now, if $u_{n_{\mathsf{cur}}+i} = 0$, set $x_i := 0$; otherwise, set $x_i := (\widetilde{\mathsf{msg}_{1,i}} + w_{n_{\mathsf{cur}}+i}) \cdot u_{n_{\mathsf{cur}}+i}^{-1}$ in $\mathbb{F}_p$. $\mathcal{S}$ sends $(\texttt{eval}, \boldsymbol{x})$ to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ and receives $\boldsymbol{y} \in \mathbb{F}_p^n$ from $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$, where for each $i \in [n]$, $y_i = (k + x_i)^g$.
3. **Simulating the transcript:** In Step 7, for each $i \in [n]$:
   (a) If $u_{n_{\mathsf{cur}}+i} = 0$, $\mathcal{S}$ samples uniform $\alpha_i' \in \mathbb{F}_p$ and sends $\mathsf{msg}_{2,i}' := \alpha_i'^{2^\lambda} \cdot (\widetilde{\mathsf{msg}_{1,i}} + w_{n_{\mathsf{cur}}+i})$;
   (b) Else if $y_i = 0$, $\mathcal{S}$ sends $\mathsf{msg}_{2,i}' := 0$ to $\mathcal{P}_c^*$;
   (c) Else, $\mathcal{S}$ executes the PPT algorithm in Figure 4 to find a z_i such that $z_i^g = y_i$, then $\mathcal{S}$ samples uniform $\alpha_i' \in \mathbb{F}_p$ and sends $\mathsf{msg}_{2,i}' := \alpha_i'^{2^\lambda} \cdot u_{n_{\mathsf{cur}}+i} \cdot z_i$ to $\mathcal{P}_c^*$.
4. $\mathcal{S}$ outputs whatever is output by $\mathcal{P}_c^*$.

We now argue why this is a valid simulator. Note that $\mathcal{P}_s$ does *not* have output except k and n, n_{batch}. As k is uniformly sampled (or a $\mathcal{P}_s$'s input in the server-specified case), all we need to show is that the transcript simulated by $\mathcal{S}$ is indistinguishable from the one in a real-world execution. This can be done by a straightforward case analysis (note, here n_{cur} is the same as n_{eval} in our protocol):

- If $u_{n_{\mathsf{cur}}+i} = 0$, in the real-world execution, $v_{n_{\mathsf{cur}}+i} = w_{n_{\mathsf{cur}}+i} + u_{n_{\mathsf{cur}}+i} \cdot k = w_{n_{\mathsf{cur}}+i}$, $\mathcal{P}_s$ would send $\alpha_i^{2^\lambda} \cdot (\widetilde{\mathsf{msg}_{1,i}} + v_{n_{\mathsf{cur}}+i}) = \alpha_i^{2^\lambda} \cdot (\widetilde{\mathsf{msg}_{1,i}} + w_{n_{\mathsf{cur}}+i})$, where α_i is uniformly sampled by (honest) $\mathcal{P}_c$. In the ideal execution, our simulator $\mathcal{S}$ (Sub-step 3a) generates an *identical* distribution.

- If $u_{n_{\mathsf{cur}}+i} \neq 0$, $\widetilde{\mathsf{msg}_{1,i}}$ can be uniquely written as $-w_{n_{\mathsf{cur}}+i} + u_{n_{\mathsf{cur}}+i} \cdot x_i$. Hence, in the real-world execution, $\mathsf{msg}_{2,i} = \alpha_i^{2^\lambda} \cdot u_{n_{\mathsf{cur}}+i} \cdot (k + x_i)$ where α_i is uniformly sampled by (honest) $\mathcal{P}_c$. On the other hand, in the ideal execution, since Step 2 extracts the same x_i, $y_i = (k + x_i)^g$. Now:
  - If $k + x_i = 0$, the real-world and ideal (Sub-step 3b) $\mathcal{P}_c^*$ both receive a 0—an *identical* distribution.
  - If $k + x_i \neq 0$, the ideal (Sub-step 3c) $\mathcal{P}_c^*$ receives $\alpha_i'^{2^\lambda} \cdot u_{n_{\mathsf{cur}}+i} \cdot z_i$ where $z_i^g = (k + x_i)^g$. This implies there exists an $a \in \mathbb{F}_p^*$, such that $z_i = a^{2^\lambda} \cdot (k + x_i)$. Hence, $\alpha_i'^{2^\lambda} \cdot u_{n_{\mathsf{cur}}+i} \cdot z_i = (a \cdot \alpha_i')^{2^\lambda} \cdot u_{n_{\mathsf{cur}}+i} \cdot (k + x_i)$ where $\alpha_i' \in \mathbb{F}_p$ is uniformly sampled by $\mathcal{S}$—an *identical* distribution to the real-world.

To conclude, the distribution seen by $\mathcal{P}_c^*$ in the real world is *identical* to the one seen by $\mathcal{P}_c^*$ in the ideal world, which further infers an *identical* distribution seen by the UC environment. The simulation is *perfect*. □

Theorem 2 (Malicious)**.** *Protocol* $\Pi_{\mathsf{2PC\text{-}Gold}}$ *(Figures 6 and 7 with gray boxes) information-theoretically UC-realizes* $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ *(Figure 5) in the* $(\mathcal{F}_{\mathsf{VOLE}}^{c,s}, \mathcal{F}_{\mathsf{VOLE}}^{s,c})$*-hybrid model (Figure 1), in the presence of static corruptions, where a corrupt* $\mathcal{P}_s$ *or* $\mathcal{P}_c$ *can be malicious.*

Proof. By constructing the simulators. When both $\mathcal{P}_s$ and $\mathcal{P}_c$ are honest, simulation is trivial and the ideal world is statistically indistinguishable from the real world: it only fails when $u_i = 0$ for one $i \in [n_{\mathsf{batch}}]$, which happens with probability $\leq \frac{n_{\mathsf{batch}}}{p}$. We now focus on the cases where either $\mathcal{P}_s$ or $\mathcal{P}_c$ is corrupted. In these cases, the simulation is perfect.

For simplicity and readability, we assume $n_{\mathsf{batch}} = n$. Supporting one n_{batch} with multiple n's (of `eval`'s) can be handled the same as our proof of Theorem 1, i.e., by $\mathcal{S}$ maintaining an n_{cur}.

Malicious $\mathcal{P}_c^*$: Intuitively, compared to the simulator used in our half-malicious protocol, the simulator $\mathcal{S}$ interacting with the malicious $\mathcal{P}_c^*$ in our malicious protocol only needs to additionally sample the transcript generated by the consistency check (Step 5) as well as the VOLE-based ZK proofs (Steps 4 and 8) as a verifier. In a nutshell, this part of the simulation is simple since $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$, knows the ZKP private coins $\Delta^{(c)}$ as well as every IT-MAC shares of $\mathcal{P}_c^*$; hence, $\mathcal{S}$ can directly compute the corresponding values $\mathcal{P}_c^*$ is expecting to receive. More precisely, we extend $\mathcal{S}$ for $\mathcal{P}_c^*$ from the half-malicious case with the following steps:

1. At Step 2, $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$, receives $\Delta^{(c)} \in \mathbb{F}_p$ from $\mathcal{P}_c^*$.
2. At Sub-steps 4a and 4b, $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$, receives $n \cdot (1 + \lambda)$ IT-MAC shares of $\mathcal{P}_c^*$ from $\mathcal{P}_c^*$.
3. At Sub-step 4c, $\mathcal{S}$ sends $\mathcal{P}_c^*$ $n \cdot \lambda$ uniformly sampled elements in $\mathbb{F}_p$. Note, this distribution in the ideal world is *identical* to the real-world one since each committed value generated in Sub-step 4b looks uniform to $\mathcal{P}_c^*$. Then, $\mathcal{S}$ adjusts each share of IT-MAC of $\mathcal{P}_c^*$ according to these uniform samples and $\Delta^{(c)}$.
4. At Sub-step 4d, $\mathcal{S}$ calls the batched LPZK simulator for verifier. This simulation is simple: $\mathcal{S}$ receives the challenges from $\mathcal{P}_c^*$, which is used to aggregate the proof; then, $\mathcal{S}$ can compute the $\mathcal{P}_c^*$'s expected value $\Pi \in \mathbb{F}_p$. Finally, $\mathcal{S}$ samples $C_1 \in \mathbb{F}_p$, sets $C_0 := \Pi - C_1 \cdot \Delta^{(c)}$, and sends C_0, C_1 to $\mathcal{P}_c^*$. This simulated distribution is *identical* to the real-world one, as in the real world and the batched LPZK check, C_1 is one-time padded by a uniform value, generated by $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$.

5. At Sub-step 5a, $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$, receives 1 IT-MAC share of $\mathcal{P}_c^*$ from $\mathcal{P}_c^*$. Then, $\mathcal{S}$ sends a uniformly sampled element in $\mathbb{F}_p$. This generates an *identical* distribution as argued in the simulator $\mathcal{S}$'s Steps 2 and 3.

6. At Sub-step 5b, $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$, receives n IT-MAC shares of $\mathcal{P}_c^*$ from $\mathcal{P}_c^*$. Then, $\mathcal{S}$ sends n uniformly sampled elements in $\mathbb{F}_p$. This generates an *identical* distribution as argued in the simulator $\mathcal{S}$'s Steps 2 and 3.

7. At Sub-step 5c, $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ for $\mathcal{P}_c^*$, receives $w_{\mathsf{sa}}, u_{\mathsf{sa}}$ from $\mathcal{P}_c^*$.

8. At Sub-step 5d, $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$, receives 1 IT-MAC share of $\mathcal{P}_c^*$ from $\mathcal{P}_c^*$. Then, $\mathcal{S}$ sends a uniformly sampled element in $\mathbb{F}_p$. This generates an *identical* distribution as argued in the simulator $\mathcal{S}$'s Steps 2 and 3.

9. At Sub-step 5e, $\mathcal{S}$ receives $\chi, \widetilde{u_{\mathsf{poly}}}, \widetilde{w_{\mathsf{poly}}}$ from $\mathcal{P}_c^*$, then checks if $\widetilde{u_{\mathsf{poly}}} \stackrel{?}{=} u_{\mathsf{sa}} + \sum_{i=1}^n u_i \cdot \chi^i$ and $\widetilde{w_{\mathsf{poly}}} \stackrel{?}{=} w_{\mathsf{sa}} + \sum_{i=1}^n w_i \cdot \chi^i$. (Note, this can be done since $\mathcal{S}$, by emulating the hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_c^*$ know all $\boldsymbol{u}, \boldsymbol{w}, u_{\mathsf{sa}}, w_{\mathsf{sa}}$.) If check fails, $\mathcal{S}$ sends `abort` to $\mathcal{P}_c^*$ and outputs whatever outputted by $\mathcal{P}_c^*$. Crucially, the real-world honest $\mathcal{P}_s$ would also abort w.h.p. This is because, if $\mathcal{P}_s$ does *not* abort, this implies

$$\widetilde{w_{\mathsf{poly}}} + \widetilde{u_{\mathsf{poly}}} \cdot \Delta^{(s)} = v_{\mathsf{poly}} = (w_{\mathsf{sa}} + \sum_{i=1}^n w_i \cdot \chi^i) + (u_{\mathsf{sa}} + \sum_{i=1}^n w_i \cdot \chi^i) \cdot \Delta^{(s)}$$

   Then, if $\widetilde{w_{\mathsf{poly}}} \neq w_{\mathsf{sa}} + \sum_{i=1}^n w_i \cdot \chi^i$ or $\widetilde{u_{\mathsf{poly}}} \neq u_{\mathsf{sa}} + \sum_{i=1}^n u_i \cdot \chi^i$, since $\Delta^{(s)}$ is uniformly sampled in $\mathbb{F}_p$, the above equality can only happen with up to negligible probability $\frac{1}{p}$, based on Corollary 1. Now, conditioned on the check is passed, $\mathcal{S}$ can compute the equation $[v_{\mathsf{sa}}]_{\Delta^{(c)}} + \sum_{i=1}^n \chi^i \cdot [v_i]_{\Delta^{(c)}} - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot [\Delta^{(s)}]_{\Delta^{(c)}}$ over $\mathcal{P}_c^*$'s shares and sends it to $\mathcal{P}_c^*$, which is *identical* to the real world as it *must* be an IT-MAC of 0.

10. At Step 8, $\mathcal{S}$ calls the batched LPZK simulator for verifier, same as the simulator $\mathcal{S}$'s Step 4.

To conclude, the distribution seen by $\mathcal{P}_c^*$ in the real world is *statistically close* to the one seen by $\mathcal{P}_c^*$ in the ideal world, which induces a *statistically indistinguishable* distribution seen by the UC environment. In particular, the statistical distance is $\leq \frac{1}{p}$—a negligible probability.

Malicious $\mathcal{P}_s^*$: The simulator $\mathcal{S}$ for malicious $\mathcal{P}_s^*$ needs to simulate (1) the transcripts seen by $\mathcal{P}_s^*$ and (2) the honest client's output. In a nutshell, $\mathcal{S}$ achieves this by relying on the soundness of the VOLE-based ZK (i.e., the batched LPZK) and our consistency check. In detail, the simulator $\mathcal{S}$, emulating $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ and $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ for $\mathcal{P}_s^*$, operates as follows:

1. At Step 1, $\mathcal{S}$ receives the OPRF key $k = \Delta^{(s)}$ from $\mathcal{P}_s^*$. Note, this step reflects the fact that $\mathcal{P}_s^*$ can select her own OPRF key, captured by $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$; see Figure 1. Then, $\mathcal{S}$ sends k to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ to set the ideal-world OPRF key.

2. At Step 3, $\mathcal{S}$ receives $\boldsymbol{v} \in \mathbb{F}^n$ from $\mathcal{P}_s^*$.

3. At Step 4, $\mathcal{S}$ performs as follows:

   (a) At Sub-step 4a, $\mathcal{S}$ receives n IT-MAC $[\boldsymbol{\alpha}]_{\Delta^{(c)}}$ shares of $\mathcal{P}_s^*$ from $\mathcal{P}_s^*$, including $\boldsymbol{\alpha} \in \mathbb{F}_p^n$.

(b) At Sub-step 4b, $\mathcal{S}$ receives $n \cdot \lambda$ IT-MAC $[\delta_{i,j}]_{\Delta^{(c)}}$ (for $i \in [n], j \in [\lambda]$) shares of $\mathcal{P}_s^*$ from $\mathcal{P}_s^*$, including $\{\delta_{i,j}\}_{i\in[n],j\in[\lambda]}$ where each $\delta_{i,j} \in \mathbb{F}_p$.

(c) At Sub-step 4c, for each $i \in [n], j \in [\lambda]$, $\mathcal{S}$ receives $\eta_{i,j} \in \mathbb{F}_p$ from $\mathcal{P}_s^*$ and set $\eta_{i,j} := \eta_{i,j} + \delta_{i,j}$.

(d) At Sub-step 4d, $\mathcal{S}$ mimics the verifier behaviors in the batched LPZK check. I.e., $\mathcal{S}$ sends some uniform elements to $\mathcal{P}_s^*$.

Now, $\mathcal{S}$ checks if the following condition is satisfied:

$$\exists i \in [n], j \in [\lambda], \eta_{i,j} \neq \alpha_i^{2^j}$$

If so, $\mathcal{S}$ sends (`abort`) to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ and then outputs whatever outputted by $\mathcal{P}_s^*$. Crucially, when the above condition is satisfied, relying on the soundness of the batched LPZK technique, the real-world honest $\mathcal{P}_c$ would also abort except for up to probability $\frac{\lambda n+1}{p}$. Subsequently, consider only the case where the above condition is unsatisfied, i.e., we assume $\forall i \in [n], j \in [\lambda], \eta_{i,j} = \alpha_i^{2^j}$.

4. At Step 5, $\mathcal{S}$ performs as follows:

(a) At Sub-step 5a, $\mathcal{S}$ receives the IT-MAC $[\beta]_{\Delta^{(c)}}$ share of $\mathcal{P}_s^*$ from $\mathcal{P}_s^*$, and a value in $\mathbb{F}_p$ from $\mathcal{P}_s^*$, which is used to obtain $[\widetilde{k}]_{\Delta^{(c)}}$. I.e., $\widetilde{k} \in \mathbb{F}_p$ is selected by $\mathcal{P}_s^*$, which does *not* have to be equal to k.

(b) Similarly, at Sub-step 5b, $\mathcal{S}$ receives the IT-MAC $[\widetilde{v}_1]_{\Delta^{(c)}}, \ldots, [\widetilde{v}_n]_{\Delta^{(c)}}$ shares of $\mathcal{P}_s^*$ from $\mathcal{P}_s^*$.

(c) At Sub-step 5c, $\mathcal{S}$ receives $v_{\mathsf{sa}} \in \mathbb{F}_p$ from $\mathcal{P}_s^*$.

(d) At Sub-step 5d, $\mathcal{S}$ receives the IT-MAC $[\widetilde{v}_{\mathsf{sa}}]_{\Delta^{(c)}}$ share of $\mathcal{P}_s^*$ from $\mathcal{P}_s^*$.

(e) At Sub-step 5e, $\mathcal{S}$ samples $\chi \xleftarrow{\$} \mathbb{F}_p, u_{\mathsf{poly}} \xleftarrow{\$} \mathbb{F}_p$ and computes $w_{\mathsf{poly}} := v_{\mathsf{sa}} + \sum_{i=1}^n v_i \cdot \chi^i - u_{\mathsf{poly}} \cdot k$. (Note, $v_{i\in[n]}$ and v_{sa}, specified by $\mathcal{P}_s^*$, are known by $\mathcal{S}$.) Then, $\mathcal{S}$ sends $\chi, u_{\mathsf{poly}}, w_{\mathsf{poly}}$ to $\mathcal{P}_s^*$. We remark that the distribution generated here is *identical* to the one for this step used in the real-world execution. This is because u_{poly} in the real-world execution is one-time padded by a uniform sampled $u_{\mathsf{sa}} \in \mathbb{F}_p$.

Now, $\mathcal{S}$ checks if the following condition is satisfied:

$$\widetilde{v}_{\mathsf{sa}} + \sum_{i=1}^{n} \chi^i \cdot \widetilde{v}_i - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot \widetilde{k} \neq 0 \vee \widetilde{k} \neq k \vee \widetilde{v}_{\mathsf{sa}} \neq v_{\mathsf{sa}} \vee \exists i \in [n], \widetilde{v}_i \neq v_i$$

If so, $\mathcal{S}$ sends (`abort`) to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ and then outputs whatever outputted by $\mathcal{P}_s^*$. Crucially, we now argue that when the above condition is satisfied, the real-world honest $\mathcal{P}_c$ would also abort w.h.p. as follows:

- If $\widetilde{v}_{\mathsf{sa}} + \sum_{i=1}^n \chi^i \cdot \widetilde{v}_i - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot \widetilde{k} \neq 0$, based on the binding property of IT-MAC, except for probability $\frac{1}{p}$, it cannot be opened to 0, implying real-world honest $\mathcal{P}_c$'s abort.
- If $\widetilde{v}_{\mathsf{sa}} + \sum_{i=1}^n \chi^i \cdot \widetilde{v}_i - w_{\mathsf{poly}} - u_{\mathsf{poly}} \cdot \widetilde{k} = 0$, by plugging in $w_{\mathsf{poly}} = v_{\mathsf{sa}} + \sum_{i=1}^n v_i \cdot \chi^i - u_{\mathsf{poly}} \cdot k$ and in the real-world, $u_{\mathsf{poly}} = u_{\mathsf{sa}} + \sum_{i=1}^n u_i \cdot \chi^i$, we have:

$$(\widetilde{v}_{\mathsf{sa}} - v_{\mathsf{sa}}) + u_{\mathsf{sa}} \cdot (k - \widetilde{k}) + \sum_{i=1}^{n} \chi^i \cdot [(\widetilde{v}_i - v_i) + u_i \cdot (k - \widetilde{k})] = 0$$

Now, if $\widetilde{k} \neq k \vee \widetilde{v}_{\mathsf{sa}} \neq v_{\mathsf{sa}} \vee \exists i \in [n], \widetilde{v}_i \neq v_i$, since u_{sa} and $u_{i\in[n]}$ each is uniformly sampled in $\mathbb{F}_p$ *and* has full entropy when $\widetilde{\boldsymbol{v}}, \widetilde{v}_{\mathsf{sa}}, \boldsymbol{v}, v_{\mathsf{sa}}, \widetilde{k}, k$ are selected by $\mathcal{P}_s^*$, except for up to probability $\frac{1}{p}$, the vector:

$$(\widetilde{v}_{\mathsf{sa}} - v_{\mathsf{sa}}) + u_{\mathsf{sa}} \cdot (k - \widetilde{k}), \{(\widetilde{v}_i - v_i) + u_i \cdot (k - \widetilde{k})\}_{i\in[n]}$$

must be a non-zero vector. Since χ is uniformly sampled from $\mathbb{F}_p$ after $\widetilde{\boldsymbol{v}}, \widetilde{v}_{\mathsf{sa}}, \boldsymbol{v}, v_{\mathsf{sa}}, \widetilde{k}, k$ were selected by $\mathcal{P}_s^*$, based on Corollary 1, the equality would only hold (i.e., $\mathcal{P}_c$ not abort) with probability at most $\frac{n}{p}$.

To sum up, the real-world honest $\mathcal{P}_c$ would also abort except for up to probability $\frac{n+2}{p}$. Subsequently, consider only the case where the condition is unsatisfied, which implies that $\forall i \in [n], \widetilde{v}_i = v_i$.

5. At Step 6, $\mathcal{S}$ receives $(\texttt{eval}, \boldsymbol{\ell})$ from $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ where $\boldsymbol{\ell}$ is the leaky array defined as $\ell_i = (k + x_i) \stackrel{?}{=} 0$ for each $i \in [n]$. Now, for each $i \in [n]$:
   - If $\ell_i = 1$, $\mathcal{S}$ sends $\mathsf{msg}'_{1,i} := -v_i$ to $\mathcal{P}_s^*$. Note, this transcript is *identical* to the real-world one as in the real-world execution $\mathsf{msg}_{1,i} + v_i = u_i \cdot (k + x_i) = 0$.
   - If $\ell_i = 0$, $\mathcal{S}$ samples and sends $\mathsf{msg}'_{1,i} \stackrel{\$}{\leftarrow} \mathbb{F}_p$ to $\mathcal{P}_s^*$. Note, this transcript is *identical* to the real-world one as in the real-world execution: (1) u_i is uniformly sampled by $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ and crucially remains full entropy to $\mathcal{P}_s^*$ at this point, and (2) $\mathsf{msg}_{1,i} + v_i = u_i \cdot (k + x_i)$.
6. At Step 7, for each $i \in [n]$, $\mathcal{S}$ receives $\widetilde{\mathsf{msg}_{2,i}} \in \mathbb{F}_p$. If $\exists i \in [n], \widetilde{\mathsf{msg}_{2,i}} = 0$, $\mathcal{S}$ sends $(\texttt{abort})$ to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ and then outputs whatever outputted by $\mathcal{P}_s^*$; obviously, the real-world $\mathcal{P}_c$ would also abort in this case.
7. At Step 8, $\mathcal{S}$ mimics the verifier behaviors in the batched LPZK check. I.e., $\mathcal{S}$ sends some uniform elements to $\mathcal{P}_s^*$.
8. Finally, if $\exists i \in [n]$, $\alpha_i^{2^\lambda} \cdot (\mathsf{msg}'_{1,i} + v_i) \neq \widetilde{\mathsf{msg}_{2,i}}$, $\mathcal{S}$ sends $(\texttt{abort})$ to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ and then outputs whatever outputted by $\mathcal{P}_s^*$. Otherwise, $\mathcal{S}$ sends $(\texttt{continue})$ to $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ and then outputs whatever outputted by $\mathcal{P}_s^*$. Clearly, based on the soundness of the batch LPZK technique, in the real-world execution, if $\exists i \in [n]$, $\alpha_i^{2^\lambda} \cdot (\mathsf{msg}_{1,i} + v_i) \neq \mathsf{msg}'_{2,i}$, the real-world honest $\mathcal{P}_c$ would also abort except for up to probability $\frac{n+1}{p}$; otherwise, $\mathcal{P}_c$ in the real-world would output *correct* OPRF evaluations, same as the ideal-world.

To conclude, the distribution seen by $\mathcal{P}_s^*$ in the real world is *statistically close* to the one seen by $\mathcal{P}_s^*$ in the ideal world, *and* the honest $\mathcal{P}_c$ outputs *identical* distribution is the two worlds. This induces a *statistically indistinguishable* distribution seen by the UC environment. In particular, the statistical distance is $\leq \frac{n(\lambda+2)+4}{p}$—a negligible probability. □

5.3 Some Details

Cost Analysis. We tally the cost of our half-malicious and malicious protocols in Figures 6 and 7 (for simplicity, we assume $n_{\mathsf{batch}} = n$):

- Step 3 and Sub-steps 4a and 4b require $n(\lambda+2)$ VOLE correlations over $\mathbb{F}_p$.
- Sub-step 4c requires $\mathcal{P}_s$ to sends $n\lambda$ field elements, and each party to perform $\mathcal{O}(n\lambda)$ field operations.
- Sub-step 4d requires 1 VOLE correlation over $\mathbb{F}_p$, $\mathcal{P}_c$ to send 1 field element, $\mathcal{P}_s$ to send 2 field elements, and each party to perform $\mathcal{O}(n\lambda)$ field operations.
- Sub-step 5a requires 1 VOLE correlation over $\mathbb{F}_p$, $\mathcal{P}_s$ to send 1 field element, and each party to perform 1 field operation.
- Sub-step 5b requires n VOLE correlations over $\mathbb{F}_p$, $\mathcal{P}_s$ to send n field elements, and each party to perform n field operations.
- Sub-step 5c requires 1 VOLE correlation over $\mathbb{F}_p$.
- Sub-step 5d requires 1 VOLE correlation over $\mathbb{F}_p$, $\mathcal{P}_s$ to send 1 field element, and each party to perform 1 field operation.
- Sub-step 5e requires $\mathcal{P}_c$ to send 3 field elements, $\mathcal{P}_s$ to send 1 field element, and each party to perform $\mathcal{O}(n)$ field operations.
- Step 6 requires $\mathcal{P}_c$ to send n field elements.
- Step 7 requires $\mathcal{P}_s$ to send n field elements. In our half-malicious protocol, it requires $\mathcal{P}_s$ to sample n field elements and perform $\mathcal{O}(n\lambda)$ field operations (in the offline phase). Then, in both protocols, it requires $\mathcal{P}_s$ to perform $\mathcal{O}(n)$ field operations.
- Step 8 requires 1 VOLE correlation over $\mathbb{F}_p$, $\mathcal{P}_c$ to send 1 field element, $\mathcal{P}_s$ to send 2 field elements, and each party to perform $\mathcal{O}(n)$ field operations.
- Step 9 requires $\mathcal{P}_c$ to perform $\mathcal{O}(n\lambda)$ field operations.

To sum up, in the VOLE-hybrid model, our protocols cost[10]:

- **Half-malicious** one needs n VOLE correlations with:
  - $\mathcal{P}_s$**'s Comp.:** $\mathcal{O}(n\lambda)$ (resp. $\mathcal{O}(n)$) field operations in offline (resp. online) phase.
  - $\mathcal{P}_c$**'s Comp.:** $\mathcal{O}(n\lambda)$ field operations in online phase.
  - **Comm.:** $2n$ field elements in online phase.
  - **Round:** 2. Note that the online phase only involves the generation of VOLE correlations.
- **Malicious** one needs $n\lambda+3n+5$ VOLE correlations with:
  - $\mathcal{P}_s$**'s Comp.:** $\mathcal{O}(n\lambda)$ (resp. $\mathcal{O}(n)$) field operations in offline (resp. online) phase.
  - $\mathcal{P}_c$**'s Comp.:** $\mathcal{O}(n\lambda)$ field operations in both offline and online phases.
  - **Comm.:** $n\lambda+n+9$ (resp. $2n+3$) field elements in offline (resp. online) phase.

[10]For computation, we count the field operations, which are the dominating costs.

– **Round:** 5 (as all checks can be executed in parallel). It can be compressed to 3 using the Fiat-Shamir transformation [FS87], which is PQ secure [YZ21].

Optimization via Generalized LPZK. The communication bottleneck of our malicious protocol lies in Sub-step 4c where $\mathcal{P}_s$ needs to commit to $n_{\mathsf{batch}} \cdot \lambda$ field elements representing each intermediate result of computing $\alpha_{i\in[n_{\mathsf{batch}}]}^{2^\lambda}$. We can exploit the generalized LPZK technique [YSWW21] (see Section 2.7) to optimize this step. In particular, w.l.o.g., let ϕ be some constant that divides λ. We can raise each element to the power of 2^ϕ rather than 2. This will reduce the communication cost of this step to $\frac{n_{\mathsf{batch}} \cdot \lambda}{\phi}$. Note that this will also increase the coefficients $\mathcal{P}_s$ needs to send in Sub-step 4d to 2^ϕ. Hence, ϕ can only be a small constant. In conclusion, for any constant ϕ divides λ, we can improve the offline communication cost to $\frac{n_{\mathsf{batch}} \cdot \lambda}{\phi} + n_{\mathsf{batch}} + 7 + 2^\phi$ field elements, with reduced VOLE correlations required.

Using the generalized LPZK will also increase the computation *concretely*—the hidden constant will increase by a factor of $\approx 2^\phi$ in asymptotic. Moreover, while it saves the required VOLE correlations for committing to the intermediate results by $\phi\times$, it requires $2^\phi - 2$ more VOLE correlations (see [HHK+24]) to randomize the coefficients sent in ZKP. Finally, the statistical advantage needs to be adjusted.

Key Selection. The protocols in Figures 6 and 7 assume a uniformly sampled PRF key $k := \Delta^{(s)}$. In some applications, $\mathcal{P}_s$ may instead want to specify this key k^*. This can be done by $\mathcal{P}_s$ sending $k^* - \Delta^{(s)}$ at the very beginning of the online phase, followed by $\mathcal{P}_c$ adjusting his $\boldsymbol{w}$. Crucially, $k^* - \Delta^{(s)}$ is one-time padded by uniformly sampled $\Delta^{(s)}$ *and* the simulator can trivially extract k^*.

Note, this does not affect the generation of $[\boldsymbol{v}]_{\Delta^{(c)}}$ in the offline phase of our malicious protocol since this generation (and check) can be based on $[\Delta^{(s)}]_{\Delta^{(c)}}$ (see Sub-step 5a).

Of course, this increases the communication complexity of both protocols by 1 $\mathbb{F}_p$ element (in the online phase).

ZKP over the Committed Key. In our malicious protocol, Sub-step 5a generates $[\Delta^{(s)}]_{\Delta^{(c)}}$ and Step 5 ensures the unforgability of $\Delta^{(s)}$. This is an IT-MAC over the PRF key. (Note, this is true even in the case where $\mathcal{P}_s$ selects her own PRF key since $[k^*]_{\Delta^{(c)}} = [\Delta^{(s)}]_{\Delta^{(c)}} + (k^* - \Delta^{(s)})$.) Thus, $\mathcal{P}_s$ can execute the standard VOLE-based ZKP using this IT-MAC to demonstrate any properties over the used PRF key. In particular, for a standard fan-in 2 arithmetic circuit $\mathcal{C} : \mathbb{F}_p^{n_{in}} \mapsto \mathbb{F}_p^*$ with $n_\times$ multiplication gates. $\mathcal{P}_s$ can perform a ZKP to show $\mathcal{P}_c$ that she knows $\boldsymbol{w} \in \mathbb{F}_p^{n_{in}-1}$ s.t. $\mathcal{C}(k, \boldsymbol{w}) = \mathbf{0}$ with a communication complexity of $n_{in} + n_\times + \mathcal{O}(1)$ field elements and a computation complexity of $\mathcal{O}(n_\times)$ field operations. For example, $\mathcal{P}_s$ can prove (1) the leading two bits of the key are zero to get rid of the one-bit leakage in $\mathcal{F}_{\mathsf{2PC\text{-}Gold}}$ (Figure 5) as discussed in Section 3.4; and (2) the key binds to a public verification key as discussed in Section 6.1.

6 Verifiability and Strong UC Security

6.1 Verifiability: Ensuring a Consistent Key

In this section, we discuss how to add different forms of verifiability to our protocols, which ensures the same key is used across multiple sessions and/or clients.

Verifiability is important, sometimes essential, in many applications. Naturally, this requires the client to keep verification information across OPRF invocations.

Private Verifiability. We show how to achieve *private verifiability*, namely, the verification information is specific to a client. This client wants to participate in an OPRF protocol multiple times and ensure that the server, which can be malicious, only uses a single key.

Note, since the PRF key is embedded in the VOLE correlations generated by $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$, if $\mathcal{P}_c$ can save all correlations (i.e., malicious batched offline phases), it already achieves private verifiability—the ZKP with a wrong key fails w.h.p.

We highlight that $\mathcal{P}_c$ can also achieve this by saving only one *fresh* correlation related to k. Specifically, consider $\mathcal{P}_c$ saves u and w, whereas $\mathcal{P}_s$ has k and $v = uk + w$. Now, during the new invocation, we can start a new $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$ where $\mathcal{P}_s$ inputs her key k' (via sending $k' - \Delta_{\mathsf{new}}^{(s)}$) and the client verifies that $k' = k$. For this, let $v' = u'k' + w'$ be a correlation generated by this new $\mathcal{F}_{\mathsf{VOLE}}^{c,s}$. Parties proceed:

1. $\mathcal{P}_c$ sends $\alpha := u - u'$.
2. $\mathcal{P}_s$ commits $\beta := v - v' - \alpha \cdot k$ via a commitment scheme.
3. $\mathcal{P}_c$ sends $\gamma := w - w'$. If $\gamma \neq \beta$, $\mathcal{P}_s$ aborts.
4. $\mathcal{P}_s$ opens β, If $\beta \neq \gamma$, $\mathcal{P}_c$ aborts.

Here, the commitment scheme prevents a malicious $\mathcal{P}_c$ to learn k. Additionally, since u and u' are uniformly sampled, $w - w' = v - v' - (uk - u'k')$ is uniformly random to $\mathcal{P}_s$ given $v, v', u - u'$ when $k \neq k'$. After verification, these two involved correlations are discarded. Note that the $\mathcal{F}_{\mathsf{VOLE}}^{s,c}$ to support ZK (and Steps 4 and 5 in Figure 6) must also be re-executed. These steps need to be executed *before* $k' - \Delta_{\mathsf{new}}^{(s)}$ is sent. This ensures that $\Delta_{\mathsf{new}}^{(s)}$ remains full entropy to the UC environment to protect against a malicious client.

We note that the UC environment can learn k from the honest $\mathcal{P}_s$ output, so a malicious $\mathcal{P}_c$ can easily pass the equality check in Step 3 with maliciously-chosen $\alpha^* = u - u' + \delta$ and $\gamma^* = w - w' - \delta k$ for some $\delta \neq 0$. While this implies the extraction of k by the UC simulator, we still need to simulate the honest $\mathcal{P}_s$'s abort. This can be done by adding one extra instruction in the ideal functionality, defined as follows:

> "**Global-key query.** If $\mathcal{P}_c$ is corrupted, receive $(\texttt{guess}, k')$ from $\mathcal{S}$: if $k = k'$, send `success` to $\mathcal{S}$ and ignore any subsequent query; otherwise, send `abort` to both parties."

This global-key query is harmless as the $\mathcal{P}_c$ with PRF evaluation can always verify his guess on the key locally.

Finally, we remark that the above technique to achieve private verifiability with a single correlation can be used to remove the predetermined n_{batch} limit: it allows the execution of a new offline phase with a new n_{batch} and ensures the same key is reused. Actually, we can directly adopt the commit-verify-open idea to the consistency check for $[v]_{\Delta^{(c)}}$ (particularly the proof of a zero IT-MAC; see Section 3.2) in a way that it can be executed even after the key k (or $\Delta^{(s)}$) is revealed to the UC environment.

Public Verifiability. We show how to achieve *public verifiability*, namely, the verification information is public (denoted as VK) and can be used by multiple clients. VK can be viewed as a public key of the server, hiding and binding to her PRF key k, and it needs to be obtained by the clients

from a reliable source (e.g., authenticated by the server, via certificates, etc.). This corresponds to the notion of *Verifiable* OPRF (VOPRF) in the literature.

We achieve this inspired by [BDFH24, Lemma 7 and 8], which focuses on building (V)OPRFs from Legendre PRFs. We first port the core Lemma into our Gold setting as follows:

Lemma 3. *For a prime p and any positive integer g that divides $p-1$, for any $k \neq k' \in \mathbb{F}_p$, there are at most $g-1$ different $x \in \mathbb{F}_p$ such that $(k+x)^g = (k'+x)^g$.*

Proof. The equation is of degree-$(g-1)$ in x. □

Lemma 4. *For a prime p and any positive integer g that divides $p-1$, for m uniformly sampled elements $\ell_1, \ldots, \ell_m \xleftarrow{\$} \mathbb{F}_p$, the following statement holds:*

$$\Pr\left[\exists k \neq k', \forall i, (k+\ell_i)^g = (k'+\ell_i)^g\right] \leq \frac{(p-1)(g-1)^m}{2p^{m-1}}$$

Proof. By applying the union bound and Lemma 3:

$$\begin{aligned}
&\Pr\left[\exists K \neq K', \forall i \in [m], (K+\ell_i)^g = (K'+\ell_i)^g\right] \\
\leq &\sum_{0 \leq K < K' < p} \Pr\left[\forall i \in [m], (K+\ell_i)^g = (K'+\ell_i)^g\right] && \text{(Union bound)} \\
= &\sum_{0 \leq K < K' < p} \prod_{i \in [m]} \Pr\left[(K+\ell_i)^g = (K'+\ell_i)^g\right] && \text{(Independent } \boldsymbol{\ell}\text{)} \\
\leq &\sum_{0 \leq K < K' < p} \prod_{i \in [m]} \frac{g-1}{p} && \text{(Lemma 3)} \\
= &\frac{(p-1)(g-1)^m}{2p^{m-1}}
\end{aligned}$$

□

In our approach, let $\ell_1, \ldots, \ell_m \in \mathbb{F}_p$ be public parameters which are assumed to be uniform; $\mathcal{P}_s$ publishes

$$\mathsf{VK}_m := (k+\ell_1)^g, \ldots, (k+\ell_m)^g \tag{1}$$

as the public key. Based on Lemma 4, this achieves statistical binding for large enough m. Concretely, since $\log p \approx 3\lambda$ and $\log g \approx 2\lambda$, we can set $m = 7$. For a uniformly sampled k, this also achieves computational hiding based on the hardness of the DSPRS problem (see Assumption 1). We only consider the uniform key since this is typical of how the OPRF key is chosen in the first place.

To validate, we exploit $[k]_{\Delta^{(c)}}$, the IT-MAC of k, generated intermediately in our malicious 2PC-Gold (i.e. Sub-step 5a in Figure 6). With it, $\mathcal{P}_s$ can prove that for each $i \in [m]$, $(k+\ell_i)^g$ meets the i-th entry of VK_m where ℓ_i is public. Note, instead of raising to the power g directly, the random root technique (see Section 3.1) can be applied for better efficiency: $\mathcal{P}_s$ can let $\mathcal{P}_c$ get $r_i^{2^\lambda} \cdot (k+\ell_i)$ for $r_i \xleftarrow{\$} \mathbb{F}_p$. In detail, parties generate $[r_i^{2^\lambda}]_{\Delta^{(c)}}$, $i \in [m]$, which can be merged into Step 4; then $\mathcal{P}_s$ sends $r_i^{2^\lambda} \cdot (k+\ell_i)$ and proves each multiplication is done correctly, which can be merged into Step 8. This results in $m\frac{\lambda}{\phi} + m$ additional $\mathbb{F}_p$ elements of communication (independently of n) and $m\frac{\lambda}{\phi} + m$

additional VOLE correlations, in the VOLE-hybrid model. See Section 7.4 for concrete cost, including VOLE.

Practically, $\boldsymbol{\ell}$ can be generated by applying a public hash function H over $[m]$. In this case, it is essential to hash the inputs to Gold with an independent hash function (this is H_1 in the $\mathsf{O\text{-}Gold}$ function discussed below). Otherwise, VK_m would reveal m Gold evaluations.

Stateless Verifiability. Either private or public verifiability discussed above requires the client to be stateful or be able to receive authenticated VK. However, in some applications, the client is stateless and has no means to guarantee an authenticated VK. Interestingly, in applications such as Password-Protected Secret Sharing (PPSS) [JKKX16], a weaker form of verification suffices: if a malicious $\mathcal{P}_s$ changes the key with $\mathcal{P}_c$ using the same input, the output is different. $\mathsf{2PC\text{-}Gold}$ does not ensure this property, particularly due to the collision issues discussed in Section 6.1. The same is the case for the function $\mathsf{O\text{-}Gold}$ that we defined (see section 1.1) as the 2Hash mode of Gold, namely,

$$\mathsf{O\text{-}Gold}_k(x) := \mathsf{H}_2(x, \mathsf{Gold}_k(\mathsf{H}_1(x))).$$

However, the above property can be provided by including the verification value VK_6 in the computation of $\mathsf{O\text{-}Gold}$ as

$$\mathsf{H}_2(x, \mathsf{Gold}_k(\mathsf{H}_1(x)), \mathsf{Gold}_k(\ell_1), \ldots, \mathsf{Gold}_k(\ell_6)) \tag{2}$$

where $\ell_1, \ldots, \ell_6 \xleftarrow{\$} \mathbb{F}_p$ are public parameters. This value VK_6 and its binding property (i.e., Lemma 4) ensures two keys produce two different public keys, ultimately resulting in two different OPRF outputs. Note that we only need VK_6 rather than VK_7 since $\mathsf{H}_1(x)$ acts as an extra point. Looking ahead, this modified $\mathsf{O\text{-}Gold}$ implements a strong UC OPRF [BDFH24], which also explains why it supports PPSS [JKKX16].

6.2 Avoiding Collisions in Gold: Towards Strong UC Security

In this section, we discuss collisions in the Gold function and their effect in defining a Gold-based UC OPRF in the strong sense (e.g., [JKK14, JKKX16, BDFH24]). By the end of this section, we show how to build such an OPRF ($\mathsf{UC\text{-}Gold}$).

Collisions in Gold—pairs $(k, x) \neq (k', x')$ s.t. $\mathsf{Gold}_k(x) = \mathsf{Gold}_{k'}(x')$—are trivial to find. However, when considering the function $\mathsf{O\text{-}Gold}_k(x) := \mathsf{H}_2(x, \mathsf{Gold}_k(\mathsf{H}_1(x)))$ where H_1 and H_2 are modeled as random oracles, the latter with 2λ bits of output, finding collisions with $x \neq x'$ is infeasible. Yet, $\mathsf{O\text{-}Gold}$ inherits one form of Gold collisions, namely, colliding pairs $(k, x), (k', x)$ for $k \neq k'$. To see that such pairs exist (and can be computed), consider a procedure similar to the algorithm in Figure 4 that on input k, x finds X such that $X^g = (k + x)^g$ and then outputs $k' := X - x$.

While collisions of the form $(k, x), (k', x)$ do not violate the standard security of PRFs, OPRF applications often require stronger properties. In particular, strong UC formulations of OPRF model these functions as random oracles with outputs independent for any two pairs $(k, x) \neq (k', x')$. Therefore, collisions of the form $(k, x), (k', x)$ are not allowed (some OPRF applications, e.g., [JKKX16], are actually insecure in the presence of such collisions).

Hence, to achieve such strong UC OPRFs, we need to eliminate these collisions. Thanks to the fact (Lemma 4) that $\mathsf{Gold}_k(\ell_1), \ldots, \mathsf{Gold}_k(\ell_6)$ act as a commitment to a single k, the function defined in Equation (2) achieves this.

This function can be proven UC secure using the formalism and methodology from [BDFH24]. Informally, they show that the 2Hash mode applied to a keyed function F, namely, $\mathsf{H}_2(x, F_k(\mathsf{H}_1(x)))$,

results in a strong UC-secure OPRF (in their corresponding OPRF functionality) provided the following properties hold for F: (1) F has a secure UC two-party computation between a server with input a key k and a client with input x (similar to the leakage-free version of 2PC-Gold functionality from Figure 5); (2) F is one-more unpredictable; and (3) collisions $F_k(x) = F_{k'}(x)$ are hard to find for any x and any $k \neq k'$.

For our case, we consider the function $F_k(x)$ defined as the concatenation of the 7 values:

$$\mathsf{Gold}_k(\mathsf{H}_1(x)), \mathsf{Gold}_k(\ell_1), \ldots, \mathsf{Gold}_k(\ell_6).$$

This F inherits from Gold its UC 2PC security (Section 5.1) and one-more unpredictability (implied by the hardness of the DSPRS problem; see Assumption 1). In addition, it is collision-resistant (in the sense of condition (3) above) based on Lemma 4. Thus, if we consider F in 2Hash mode, we obtain exactly the function defined in Equation (2), and the results from [BDFH24, Theorem 1] imply this function is a UC OPRF in the formalization of [BDFH24]. For concreteness and future use, we define:

$$\mathsf{UC\text{-}Gold}_k(x) := \mathsf{H}_2\big(x, \mathsf{Gold}_k(\mathsf{H}_1(x)), \mathsf{Gold}_k(\mathsf{H}_0(1)), \ldots, \mathsf{Gold}_k(\mathsf{H}_0(6))\big)$$

where $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$ are hash functions modeled as independent random oracles with ranges $\mathbb{F}_p$, $\mathbb{F}_p$ and $\{0,1\}^{2\lambda}$, resp.

7 Implementation and Benchmark

We implemented our half-malicious and malicious 2PC-Gold with both non-batched and batched variants (Figures 6 and 7) using C++. In particular, since the performance difference is nuanced, we consider batched variants with a single batch, i.e., $n_{\mathsf{batch}} = n$. In this section, we discuss this implementation and provide a comprehensive benchmark.

We only implemented 2PC-Gold since it is sufficient for many applications and reflects our performance. Note, 2PC-Gold and O-Gold have similar concrete performance, and we estimate additional costs needed for UC-Gold in Section 7.4. One can easily extend our compact and modular implementation to O-Gold and UC-Gold.

We only consider the case where the key is uniformly sampled since extra costs to support a server-specified key are negligible, as discussed in Section 7.4.

In this paper, we use the convention: 1KB$= 2^{10}$B.

Open-Source Implementation. Our implementation is public and available at

https://github.com/gconeice/PR-OPRF

7.1 Setup

Security Level. Our implementation considers $\lambda = 128$, aiming at the NIST Security Strength Category 1 for post-quantum cryptography. In particular, we use 128-bit OTs, and AES-128 to implement, e.g., the PRNG. We chose this based on baselines in the literature. We note that the famous Grover algorithm [Gro96] can indeed provide a square-root quantum attack over AES-128, but this is not considered as a practical attack (see, e.g., [GLRS16, Zal99]). We can upgrade our implementation to be based on 256-bit OTs and AES-256 with some engineering efforts, and our estimated performance overhead is only $\leq 2\times$.

Prime p. Our implementation sets $p = 2^{128} \cdot g + 1$ as a 384-bit prime to produce 128-bit OPRF outputs. We choose $g = 2^{256} - 33375$, which is the largest 256-bit prime, ensuring p is also a prime. Observe that for our selected p, hash functions or PRNGs generating outputs in $\{0,1\}^{384}$ suffice to produce $\mathbb{F}_p$ elements.[11] We remark that g does not need to be a prime, but we chose a prime g as a conservative option. We use the `GMP` library [Gt20] for $\mathbb{F}_p$ operations.

Functionality $\mathcal{F}_{\mathsf{VOLE}}$. Recall that our protocols are designed in the VOLE-hybrid (Figure 1) model. To generate these VOLE correlations, we deploy the following malicious-secure VOLE protocols:

- **Non-batched variant:** Our non-batched (i.e., single-input) variant, either half-malicious or malicious, only requires a small number of VOLE correlations. Hence, we implemented the VOLE protocol in [BDFH24], which relies on [Roy22, BBD+23]. In short, in this protocol, besides a one-time setup to generate random OTs and some GGM trees, each VOLE correlation requires $\approx \frac{\log p + 2\sigma}{t}$ elements of $\mathbb{F}_p$, where $\sigma = 64$ is the statistical security parameter and $t = 8$ is a communication-computation trade-off parameter—a per-correlation cost of $\approx$ 3KB.

- **Batched variant:** Our n-batched (particularly for a sufficiently large n) variant, either half-malicious or malicious, requires a large number of VOLE correlations. Hence, we deployed the [WYKW21]'s VOLE protocol to extend a small amount of VOLE correlations into a large amount of VOLE correlations with sublinear communication (in length) based on OTs and the post-quantum LPN assumption. That is, amortized communication is almost free. [WYKW21]'s implementation is open-sourced in the `EMP-Toolkit` [WMK16], and we adopted it with adjustments toward $\mathbb{F}_p$ and PQ OTs.

OTs. We utilize the `libOTe` library [RR] to implement OTs. These (random) OTs are generated using the state-of-the-art malicious-secure post-quantum OT extension technique [Roy22] relied on base (random) OTs, which we generate using the malicious-secure OT protocol in [MR19]. In particular, [MR19]'s OT protocol can be instantiated under either a classical (i.e., Diffie-Hellman-type with curve25519) or post-quantum (i.e., lattice-type with Kyber512) assumption. Switching between these instantiations yields classical or post-quantum security to the full 2PC-Gold. We consider both in our benchmark.

Hardware. Our experiments were executed on two AWS EC2 `m5.large` machines[12] that respectively implemented $\mathcal{P}_s$ and $\mathcal{P}_c$. Each party ran single-threaded. We configured different network settings via Linux `tc` command:

- **WAN-like:** 25Mbps with a 30ms round-trip latency.

- **LAN-like:** 1Gbps with a 2ms round-trip latency.

These configurations are selected to match prior work.

Metrics. We report the following metrics:

- **Communication:** the total communication.

- **Computation:** the total execution time of $\mathcal{P}_s$ and $\mathcal{P}_c$ respectively. Note that the end-to-end (E2E) execution time is exactly the $\mathcal{P}_c$'s execution time.

Security	Metric	$\phi =$ 1	2	4	8
Classical	Comm. (KB)	488	302	242	914
	WAN, E2E Time (ms)	1447	1120	966	2229
	LAN, E2E Time (ms)	879	606	508	1509
PQ	Comm. (KB)	1216	1030	970	1642
	WAN, E2E Time (ms)	1682	1333	1206	2423
	LAN, E2E Time (ms)	887	607	510	1503

Table 1: Performance of our non-batched malicious protocol as a function of the optimization parameter ϕ. Note that $\phi = 1$ is essentially the case of no optimization. We highlight the column $\phi = 4$, which provides the lowest cost.

Variant	Security $\mathcal{P}_s$-$\mathcal{P}_c$	Comm.	WAN, Time $\mathcal{P}_s$	WAN, Time $\mathcal{P}_c$	LAN, Time $\mathcal{P}_s$	LAN, Time $\mathcal{P}_c$
Non-batched (single-input)	◐-●	46KB	338ms	368ms	158ms	160ms
	●-●	242KB	936ms	966ms	506ms	508ms
Batched (amortized)	◐-●	99B	61µs	87µs	30µs	56µs
	●-●	1.9KB	1.6ms	1.6ms	1ms	1ms

(a) Classical

Variant	Security $\mathcal{P}_s$-$\mathcal{P}_c$	Comm.	WAN, Time $\mathcal{P}_s$	WAN, Time $\mathcal{P}_c$	LAN, Time $\mathcal{P}_s$	LAN, Time $\mathcal{P}_c$
Non-batched (single-input)	◐-●	774KB	538ms	568ms	161ms	163ms
	●-●	970KB	1.1s	1.1s	507ms	510ms
Batched (amortized)	◐-●	100B	61µs	87µs	30µs	57µs
	●-●	1.9KB	1.6ms	1.6ms	1ms	1ms

(b) Post-quantum

Table 2: Performance of our protocols with classical and post-quantum OTs. The overall security is inherited from the OT. ◐ denotes the semi-honest security whereas ● denotes the malicious security. The costs of batched protocols correspond to the costs of a single OPRF evaluation when amortized as part of an n-batched evaluation with $n \approx 10^7$ for half-malicious and $n \approx 3 \times 10^5$ for malicious (smaller n are also possible, see text and Table 4).

7.2 Performance of Our Protocols

Optimization Parameter ϕ. Recall that our malicious 2PC-Gold can leverage the generalized LPZK technique [YSWW21] (Section 2.7) to balance communication and computation. We set $\phi = 4$.

We performed experiments over our non-batched malicious 2PC-Gold with either classical or post-quantum security to select this optimization parameter ϕ. Table 1 tabulates the results.

According to these results, we set $\phi = 4$. We remark that while our batched protocols may enjoy a larger ϕ since the communication overhead induced by 2^ϕ coefficients in ZKP can be amortized, our experiments showed that this is not the case because of 2^ϕ field operations in computation.

Overall Performance. Table 2a (resp. Table 2b) tabulates the results with classical (resp. post-quantum) OTs.

For the batched test cases, we set the evaluation number n large enough to enable VOLE extension and use up the extended VOLE correlations, seeking the best amortization. In particular, $n \approx 10^7$ (resp. $\approx 3 \times 10^5$) for half-malicious (resp. malicious) protocols.[13] This is purely for benchmarking. While this $n \approx 10^7$ (or $\approx 3 \times 10^5$) is determined by the LPN parameter in [WMK16], one can adjust

[11] A random integer in $[2^{384}]$ is $< p$ with overwhelming probability.

[12] Intel Xeon Platinum 8259CL @ 2.50GHz, 2 vCPUs, 8GiB Memory

[13] Note, $\mathcal{P}_s$ and $\mathcal{P}_c$ each only needs to save the base VOLE correlations to support, e.g., day-to-day invocations. For example, for $n = 10^7$, about 5000 base correlations suffice, requiring about 1MB physical memory.

Security $\mathcal{P}_s$-$\mathcal{P}_c$		Offline Comm. (B) OT $\mathcal{P}_s \to \mathcal{P}_c$	OT $\mathcal{P}_c \to \mathcal{P}_s$	VOLE $\mathcal{P}_s \to \mathcal{P}_c$	VOLE $\mathcal{P}_c \to \mathcal{P}_s$	Π_{offline} $\mathcal{P}_s \to \mathcal{P}_c$	Π_{offline} $\mathcal{P}_c \to \mathcal{P}_s$	Online Comm. (B) Π_{online} $\mathcal{P}_s \to \mathcal{P}_c$	Π_{online} $\mathcal{P}_c \to \mathcal{P}_s$
◐-●	Classical	13,805	8,368	16	25,552	–	–	48	48
	PQ	439,757	327,800						
●-●	Classical	14,917	22,109	179,792	28,592	2,496	192	144	48
	PQ	440,869	341,541						

Table 3: Fine-grained communication analysis of our non-batched protocols. ◐ denotes the semi-honest security whereas ● denotes the malicious security. Note that the difference between classical and PQ instantiations lies only in OTs.

Variant	Security $\mathcal{P}_s$-$\mathcal{P}_c$	Comm.	WAN, Time $\mathcal{P}_s$	WAN, Time $\mathcal{P}_c$	LAN, Time $\mathcal{P}_s$	LAN, Time $\mathcal{P}_c$
Batched	◐-●	175B	181µs	216µs	120µs	146µs
(amortized)	●-●	17.1KB	24.4ms	24.4ms	19.4ms	19.4ms

Table 4: Performance of our PQ protocols with moderate size n. ◐ denotes the semi-honest security whereas ● denotes the malicious security. The costs of batched protocols correspond to the costs of a single OPRF evaluation when amortized as part of an n-batched evaluation with $n = 10^5$ for half-malicious and $n = 10^3$ for malicious.

their VOLE extension parameters (e.g., using the LPN estimator [LWYY24]) to smaller values (e.g., 10^5) with *little effect* on the amortized cost.

The performance of our batched variant is almost identical between classical and post-quantum instantiations. This is because only $\lambda = 128$ base OTs need to be generated accordingly, and the (amortized) cost difference is negligible.

Fine-Grained Analysis. We performed fine-grained analysis over the communication of our non-batched protocols, tabulated in Table 3. Here, Π_{offline} (resp. Π_{online}) denotes the offline (resp. online) phase of our protocols (see Figures 6 and 7) in the VOLE-hybrid model; the communication of OTs includes the generation of base OTs and OT extension.

We make the following remarks:

- The communication difference between classical and PQ instantiations only lies in the cost of OTs. In particular, the communication cost of OTs mainly depends on the generation of base OTs.
- For either classical or PQ instantiation, the communication of OTs between the half-malicious and malicious protocols is small. This might be counter-intuitive since our malicious protocols require OTs in both directions. However, note that we only need to generate base OTs in one direction, and the extended OTs can be used as base OTs in the opposite direction.
- The communication of $\Pi_{\mathsf{offline}}+\Pi_{\mathsf{online}}$ in malicious protocols (2.8KB) does not match the amortized communication cost of the batched variants (1.9KB). This is mainly because in the batched variants, the 16 coefficients of the generalized LPZK to prepare $[\alpha^{2^\lambda}]_{\Delta^{(c)}}$ can be amortized over n evaluations, inducing a $16 \cdot 384$ bits difference.
- The online phase of our malicious protocol incurs 96 additional Bytes (i.e., 2 $\mathbb{F}_p$ elements), compared to the online phase of our half-malicious protocol. This reflects the VOLE-based ZK

over a single multiplication. In the batched variant, the additive overhead (i.e., independent of n) is 3 $\mathbb{F}_p$ elements, where one extra element is used to aggregate the n-multiplication check.

- Almost all communication of our protocols is used to generate VOLE correlations. As our protocols are black-box in VOLE, it is valuable to study how to more efficiently generate a small amount of VOLE correlations, even classically. Indeed, even improving the cost of generating (base) PQ OTs may significantly increase the performance of our non-batched PQ performance.

  Our half-malicious non-batched protocol needs only a single VOLE correlation, whereas our malicious non-batched protocol requires a total of 54 VOLE correlations (2 in one direction and 52 in the other direction).

Selecting n for Cost Amortization. In our batched setting, we need a large enough n to effectively amortize the communication cost. For example, in our malicious PQ batched setting, the amortized cost comes from two components: (1) a per OPRF evaluation cost that we can estimate to be about 1.9KB using Table 2b (where $n \approx 3 \times 10^5$); (2) the generation of around $1,800$ base VOLE correlations (in each direction, depending *solely* on the LPN parameters), necessary to initiate VOLE extensions using the *primal LPN*. We can use Table 3 to estimate the latter as follows:

$$\overbrace{(430+333)\text{KB}}^{\text{Base PQ OTs in Table 3}} + \overbrace{(175+28)\text{KB}}^{\text{VOLE setup in Table 3}} + \overbrace{3600 \cdot 3\text{KB}}^{3,600 \text{ more VOLEs}}$$

which is approximately $12,000$KB. We can then approximate the amortized cost for any n to: $1.9\text{KB} + 12,000\text{KB}/n$. For example, for $n = 10^3$, the cost is about 14KB; for $n = 10^5$, it is about 2KB.

We instantiated the LPN-based VOLE extension with a smaller n to verify the above estimation; see Table 4.

An even smaller cost of (2) can be achieved by using VOLE extension protocols that rely on the *dual LPN* [BCGI18], which requires fewer base VOLE correlations.

7.3 Comparison with Prior Work

We compare our protocols concretely with prior post-quantum OPRFs. Most of them do not have a (public) implementation, but we try our best to compare with them:

- **Isogeny-based:** The state-of-the-art *semi-honest* (both client and server are semi-honest) OPRF based on isogenies is OPUS [HHM+24]. OPUS focuses on the non-batched setting, and it is unclear how it can be optimized/amortized in the batched setting. When $\lambda = 128$, the communication cost of OPUS is $\approx$ 24KB. This outperforms our half-malicious *non-batched* PQ protocol. (Note that in our half-malicious model, the client is fully malicious, and only the server is semi-honest.)

  However, OPUS requires 258 rounds while ours requires 3 rounds in the VOLE-hybrid model (with Fiat-Shamir), and the generation of VOLE correlations can be finished in $\leq$5 rounds. More importantly, OPUS is extremely computationally intensive. We tested the open-sourced OPUS implementation on our machines, and it required over 13s E2E for each evaluation in the WAN. Hence, ours is over 20$\times$ better in terms of E2E time and has stronger security. Our improvement is over 50$\times$ in LAN and over 100000$\times$ in the batched setting.

  The state-of-the-art malicious isogeny-based OPRF is [Bas23], which requires 8.7MB communication. Our communication is over 9$\times$ smaller. Note, [Bas23]'s protocol is a theoretical result, and no implementation is available. Similar to OPUS, it is very heavy in computation.

- **Lattice-based:** The state-of-the-art malicious OPRF based on lattices is [AG24, ESTX24], which requires ≈200KB in the batched amortized setting. Ours is ≈100× better. Both [AG24] and [ESTX24] need 2 rounds.

  The state-of-the-art semi-honest OPRF based on lattices is [HKL+25], which requires 23KB in the batched amortized setting. Ours is ≈235× better. [HKL+25] needs 6 rounds.

  On the other hand, [ESTX24] requires ≈400KB in the non-batched malicious setting, which is ≈2× better than our corresponding protocol in communication. However, [ESTX24] relies on SNARG, which is computationally expensive.

- **"Crypto-Dark-Matter" [BIP+18]:** The state-of-the-art semi-honest OPRF based on the "Crypto-Dark-Matter" is [APRR24], which requires 119 Bytes of communication in the batched amortized setting. Ours achieves 100 Bytes, and in addition, we provide security against a malicious client. [APRR24] needs 2 rounds in the VOLE-hybrid model, same as ours. On the other hand, in general, "Crypto-Dark-Matter" PRF constructions have a lower computational overhead. It should also be noted that the parameters used in [APRR24] are overly aggressive, as analyzed by [AR24].

- **Legendre-based:** The state-of-the-art malicious OPRF based on regular Legendre PRFs is [BDFH24] which focuses on non-batched setting and estimates a 911KB communication (without implementation). This protocol needs 9 rounds [ESTX24].

  In comparison, our *real-world tested* communication is 970KB. However, [BDFH24] estimates a 296KB cost of OTs, relying on [BMM22] to generate base OTs. By using these OTs, our communication can get down to 502KB—1.8× better. More significantly, our per-evaluation communication reduces to just 1.9KB when amortized over batched evaluations. In contrast, [BDFH24]'s protocol does not support batched amortization. This is inherent in their approach because they use (non-standard) VOLE to commit the evaluation input x by revealing $x - \Delta$, where Δ is the scalar from the VOLE. Thus, for each new x, new VOLE correlations (with a new Δ) must be generated. See also Section 1.2.

  Note that our protocols rely solely on standard VOLE correlations in a black-box manner. [BDFH24]'s protocol requires a customized VOLE functionality. Hence, our protocols are more friendly to future improvements of VOLE.

7.4 Projected Overhead for Additional Properties

In this section, we discuss the overhead that will occur if additional properties are added to our implementation. We focus solely on the non-batched setting, as this overhead would be amortized in the batched setting.

Server-specified Key. This only incurs an additional $\mathbb{F}_p$ element (48 Bytes) to be transferred in the online phase.

Randomized $\Delta^{(s)}$. This only incurs an additional $\mathbb{F}_p$ element (48 Bytes) to be transferred in the offline phase. Recall that this can eliminate the one-bit leakage if we only need a uniformly sampled key; see Section 3.4.

Avoiding Leakage for a Server-specified Key. In the case of a server-specified key, we can eliminate the one-bit leakage by using a key with two leading zeros, as discussed in Section 3.4. This can be done by requiring (malicious) $\mathcal{P}_s$ to bit-decompose $[k]_{\Delta^{(c)}}$. To improve efficiency, $\mathcal{P}_s$ can

decompose k into 95 4-bit trunks and 2 1-bit trunks. Hence, it requires 97 VOLE correlations (each cost $\approx$ 3KB) and 97 derandomization over correlations (each cost 48 Bytes, i.e., sending $z - u$ to convert a random IT-MAC $[u]_{\Delta^{(c)}}$ to $[z]_{\Delta^{(c)}}$). In total, this gives a $\approx$ 296KB overhead. We also need to ensure that $\mathcal{P}_s$ uses values within the ranges $[0, 16)$ (and $[0, 2)$). This can be *freely* incorporated with the power-raising check as degree-16 (and 2) polynomials.

Public Verifiability. Recall that Gold over 7 public random inputs can be used as the public verification information (i.e., VK_7 in Equation (1)) to achieve public verifiability. To ensure the key used is indeed in line with published VK_7, each public input (i.e., $\ell_{i\in[7]}$) requires $\frac{\lambda}{\phi} + 1 = 33$ more VOLE correlations (each cost $\approx$ 3KB) and $\frac{\lambda}{\phi} = 32$ derandomization over the intermediate result (each cost 48 Bytes) and 1 field element for sending $r_i^{2^\lambda} \cdot (k + \ell_i)$; see Section 6.1. Hence, the total overhead for achieving public verifiability is $\approx$ 703KB.

UC-Gold. To achieve UC-Gold, we need to verify VK_6 (which does not need to be public now) and also deploy leakage-free 2PC-Gold; see Section 6.1. Hence, the total overhead for achieving UC-Gold is $\approx$ 899KB ($\approx$ 603KB for verification of VK_6 and $\approx$ 296KB for verification that the key has two leading zeros to remove leakage). This overhead is independent of n and can be amortized.

Acknowledgement

We thank our S&P'25 reviewers for their thoughtful feedback. We also thank Ward Beullens, Julia Hesse, Yuval Ishai, Peter Rindal, and Lawrence Roy for insightful discussions.

References

[ADDG24] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus - oblivious PRFs from shallow PRFs and TFHE. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 447–476. Springer, Cham, May 2024.

[ADDS21] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 261–289. Springer, Cham, May 2021.

[AG24] Martin R. Albrecht and Kamil Doruk Gür. Verifiable oblivious pseudorandom functions from lattices: Practical-ish and thresholdisable. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 205–237. Springer, Singapore, December 2024.

[APRR24] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. Improved alternating-moduli PRFs and post-quantum signatures. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 274–308. Springer, Cham, August 2024.

[AR24] Irati Manterola Ayala and Håvard Raddum. Zeroed out: Cryptanalysis of weak PRFs in alternating moduli. Cryptology ePrint Archive, Report 2024/2055, 2024.

[Bas23] Andrea Basso. A post-quantum round-optimal oblivious PRF from isogenies. Cryptology ePrint Archive, Report 2023/225, 2023.

[BBD+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

[BBUV20] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. Cryptanalysis of the Legendre PRF and generalizations. *IACR Trans. Symm. Cryptol.*, 2020(1):313–330, 2020.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Berlin, Heidelberg, December 2011.

[BDFH24] Ward Beullens, Lucas Dodgson, Sebastian Faller, and Julia Hesse. The 2Hash OPRF framework and efficient post-quantum instantiations. Cryptology ePrint Archive, Report 2024/450, 2024.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011.

[BDSW23] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. Sok: vector ole-based zero-knowledge protocols. *Des. Codes Cryptogr.*, 91(11):3527–3561, 2023.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, Berlin, Heidelberg, August 1995.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291. Springer, Berlin, Heidelberg, August 1994.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos

Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 699–729. Springer, Cham, November 2018.

[BKM+21] Andrea Basso, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Antonio Sanso. Cryptanalysis of an oblivious PRF from supersingular isogenies. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 160–184. Springer, Cham, December 2021.

[BKW20] Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 520–550. Springer, Cham, December 2020.

[BMM22] Saikrishna Badrinarayanan, Daniel Masny, and Pratyay Mukherjee. Efficient and tight oblivious transfer from PKE with tight multi-user security. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22International Conference on Applied Cryptography and Network Security*, volume 13269 of *LNCS*, pages 626–642. Springer, Cham, June 2022.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CHL22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious pseudorandom functions. In *2022 IEEE European Symposium on Security and Privacy*, pages 625–646. IEEE Computer Society Press, June 2022.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Cham, August 2020.

[CW24a] Henry Corrigan-Gibbs and David J. Wu. Legendre sequences are pseudorandom under the quadratic-residuosity assumption. Cryptology ePrint Archive, Report 2024/1252, 2024.

[CW24b] Henry Corrigan-Gibbs and David J. Wu. The one-wayness of jacobi signatures. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part V*, volume 14924 of *LNCS*, pages 3–13. Springer, Cham, August 2024.

[Dam90] Ivan Damgård. On the randomness of Legendre and Jacobi sequences. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 163–172. Springer, New York, August 1990.

[Dav31] Harold Davenport. On the distribution of quadratic residues (mod p). *Journal of the London Mathematical Society*, 1(1):49–54, 1931.

[DCZ+24] Shen Dong, Hongrui Cui, Kaiyi Zhang, Kang Yang, and Yu Yu. A simple post-quantum oblivious transfer protocol from mod-LWR. Cryptology ePrint Archive, Paper 2024/1116, 2024.

[DE52] Harold Davenport and Paul Erdös. The distribution of quadratic and higher residues. *Publ. Math. Debrecen*, 2(3–4):252–65, 1952.

[DGH+21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 517–547, Virtual Event, August 2021. Springer, Cham.

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

[DHS98] Cunsheng Ding, Tor Hesseseth, and Weijuan Shan. On the linear complexity of legendre sequences. *IEEE Transactions on Information Theory*, 44(3):1276–1278, 1998.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *2nd Conference on Information-Theoretic Cryptography*, 2021.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Berlin, Heidelberg, January 2005.

[ESTX24] Muhammed F. Esgin, Ron Steinfeld, Erkan Tairi, and Jie Xu. LeOPaRd: Towards practical post-quantum oblivious PRFs via interactive lattice problems. Cryptology ePrint Archive, Paper 2024/1615, 2024.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Berlin, Heidelberg, February 2005.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[FOO23] Sebastian H. Faller, Astrid Ottenhues, and Johannes Ottenhues. Composable oblivious pseudo-random functions via garbled circuits. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *LATINCRYPT 2023*, volume 14168 of *LNCS*, pages 249–270. Springer, Cham, October 2023.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

[FS21] Paul Frixons and André Schrottenloher. Quantum security of the legendre PRF. Cryptology ePrint Archive, Report 2021/149, 2021.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to aes: quantum resource estimates. In *International Workshop on Post-Quantum Cryptography*, pages 29–43. Springer, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[GMS14] Katalin Gyarmati, Christian Mauduit, and András Sárközy. The cross-correlation measure for families of binary sequences. In Gerhard Larcher, Friedrich Pillichshammer, Arne Winterhof, and Chaoping Xing, editors, *Applied Algebra and Number Theory*, number Theory, pages 126–143. Cambridge University Press, 2014.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 430–443. ACM Press, October 2016.

[Gt20] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.2.1 edition, 2020. https://gmplib.org.

[HHK+24] Carmit Hazay, David Heath, Vladimir Kolesnikov, Muthuramakrishnan Venkitasubramaniam, and Yibin Yang. LogRobin++: Optimizing proofs of disjunctive statements in VOLE-based ZK. Cryptology ePrint Archive, Paper 2024/1427, 2024.

[HHM+24] Lena Heimberger, Tobias Hennerbichler, Fredrik Meisingseth, Sebastian Ramacher, and Christian Rechberger. OPRFs from isogenies: Designs and analysis. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas, editors, *ASIACCS 24*. ACM Press, July 2024.

[HKL+25] Lena Heimberger, Daniel Kales, Riccardo Lolato, Omid Mir, Sebastian Ramacher, and Christian Rechberger. Leap: A fast, lattice-based OPRF with application to private set intersection. Cryptology ePrint Archive, Paper 2025/333, 2025.

[HY24] Carmit Hazay and Yibin Yang. Toward malicious constant-rate 2PC via arithmetic garbling. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 401–431. Springer, Cham, May 2024.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Berlin, Heidelberg, December 2014.

[JKKX16] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *2016 IEEE European Symposium on Security and Privacy*, pages 276–291. IEEE Computer Society Press, March 2016.

[JKR19] Stanislaw Jarecki, Hugo Krawczyk, and Jason K. Resch. Updatable oblivious key management for storage systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 379–393. ACM Press, November 2019.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Cham, April / May 2018.

[KCM24] Novak Kaluderovic, Nan Cheng, and Katerina Mitrokotsa. A post-quantum distributed OPRF from the legendre PRF. Cryptology ePrint Archive, Report 2024/544, 2024.

[Kho19] Dmitry Khovratovich. Key recovery attacks on the Legendre PRFs within the birthday bound. Cryptology ePrint Archive, Report 2019/862, 2019.

[KKK20a] Novak Kaluđerović, Thorsten Kleinjung, and Dušan Kostić. Cryptanalysis of the generalised legendre pseudorandom function. *Open Book Series*, 4(1):267–282, 2020.

[KKK20b] Novak Kaluđerović, Thorsten Kleinjung, and Dusan Kostic. Improved key recovery on the legendre PRF. Cryptology ePrint Archive, Report 2020/098, 2020.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

[KLOR20] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 308–336. Springer, Cham, August 2020.

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 149–179. Springer, Cham, May 2024.

[MM22] Luciano Maino and Chloe Martindale. An attack on SIDH with arbitrary starting curve. Cryptology ePrint Archive, Report 2022/1026, 2022.

[MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, November 2019.

[MS97] Christian Mauduit and András Sárközy. On finite pseudorandom binary sequences i: Measure of pseudorandomness, the legendre symbol. *Acta Arithmetica*, 82(4):365–377, 1997.

[MZ22] Alexander May and Floyd Zweydinger. Legendre PRF (multiple) key attacks and the power of preprocessing. In *CSF 2022 Computer Security Foundations Symposium*, pages 428–438. IEEE Computer Society Press, August 2022.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Berlin, Heidelberg, August 2012.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

[Per92] Rene Peralta. On the distribution of quadratic residues and nonresidues modulo a prime number. *Mathematics of Computation*, 58(197):433–440, 1992.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over gf (p) and its cryptographic significance (corresp.). *IEEE Transactions on information Theory*, 24(1):106–110, 1978.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.

[Roy22] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Cham, August 2022.

[RR] Peter Rindal and Lance Roy. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[RS04] Alexander Russell and Igor E Shparlinski. Classical and quantum function reconstruction via character evaluation. *Journal of Complexity*, 20(2-3):404–422, 2004.

[Sch80] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[SHB23] István András Seres, Máté Horváth, and Péter Burcsi. The legendre pseudorandom function as a multivariate quadratic cryptosystem: security and applications. *Applicable Algebra in Engineering, Communication and Computing*, pages 1–31, 2023.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[SKR+24] Yunqing Sun, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Xiao Wang. Actively secure private set intersection in the client-server setting. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1478–1492. ACM Press, October 2024.

[Tót07] Viktória Tóth. Collision and avalanche effect in families of pseudorandom binary sequences. *Periodica Mathematica Hungarica*, 55:185–196, 2007.

[vDH00] Wim van Dam and Sean Hallgren. Efficient quantum algorithms for shifted quadratic character problems. *arXiv preprint quant-ph/0011067*, 2000.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021.

[YZ21] Takashi Yamakawa and Mark Zhandry. Classical vs quantum random oracles. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 568–597. Springer, Cham, October 2021.

[Zal99] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 1999.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.