Amortizing Circuit-PSI in the Multiple Sender/Receiver Setting

Aron van Baarsen[1,2] and Marc Stevens[1]

[1] CWI, Cryptology Group, Amsterdam, The Netherlands
[2] Leiden University, Mathematical Institute, Leiden, The Netherlands

Abstract. Private set intersection (PSI) is a cryptographic functionality for two parties to learn the intersection of their input sets, without leaking any other information. Circuit-PSI is a stronger PSI functionality where the parties learn only a secret-shared form of the desired intersection, thus without revealing the intersection directly. These secret shares can subsequently serve as input to a secure multiparty computation of any function on this intersection.
In this paper we consider several settings in which parties take part in multiple Circuit-PSI executions with the same input set, and aim to amortize communications and computations. To that end, we build up a new framework for Circuit-PSI around generalizations of oblivious (programmable) PRFs that are extended with offline setup phases. We present several efficient instantiations of this framework with new security proofs for this setting. As a side result, we obtain a slight improvement in communication and computation complexity over the state-of-the art Circuit-PSI protocol by Bienstock et al. (USENIX '23). Additionally, we present a novel Circuit-PSI protocol from a PRF with secret-shared outputs, which has linear communication and computation complexity in the parties' input set sizes, and incidentally, it realizes "almost malicious" security, making it the first major step in this direction since the protocol by Huang et al. (NDSS '12). Lastly, we derive the potential amortizations over multiple protocol executions, and observe that each of the presented instantiations is favorable in at least one of the multiple-execution settings.

Keywords: Circuit-PSI · Private Set Intersection · OPRF · MPC

1 Introduction

Private Set Intersection (PSI) is a specific type of multi-party computation where two parties hold private sets X and Y and wish to learn the intersection $X \cap Y$ without revealing any information about elements outside the intersection. The first PSI protocol [Mea86] makes use of an oblivious pseudorandom function (OPRF), and the state-of-the art protocols [KKRT16, PRTY19, CM20, RS21, RR22, BPSY23] still follow this same structure. An OPRF [FIPR05] is a two-party functionality that lets a receiver evaluate a pseudorandom function on a set of inputs while the sender remains oblivious to the inputs/outputs and the receiver remains oblivious to the function.

The focus of this work lies on *Circuit-PSI* protocols, realizing a more generic and richer functionality which outputs $f(X \cap Y)$ for a function f, instead of outputting the intersection directly. In particular, we consider general solutions for Circuit-PSI that output secret shares of flags indicating whether an element x_i in the receiver's set X matches an element y_j in the sender's set Y. Such a solution moreover outputs shares of auxiliary data, which, if there is a match $x_i = y_j$ with the sender's set, reconstruct to the

Research partially funded by NWO/TKI Grant 628.009.014.
E-mail: aronvanbaarsen@gmail.com (Aron van Baarsen), marc.stevens@cwi.nl (Marc Stevens)

Figure 1 Ideal Circuit-PSI functionality $\mathcal{F}_{\mathsf{Circuit\text{-}PSI}}$

Parameters: Receiver with input set $X = \{(x_i, \tilde{x}_i) \mid i \in [n]\}$ of size n, Sender with input set $Y = \{(y_j, \tilde{y_j}) \mid j \in [m]\}$ of size m. Variables x_i and y_j are identifiers chosen from the same set, the variables $\tilde{x}_i$ and $\tilde{y}_j$ are auxiliary data chosen from another set.
Functionality: Send secret shares $(a_i)_{i\in[n]}$ to receiver and $(b_i)_{i\in[n]}$ to sender, where a_i and b_i are sampled uniformly at random under the following constraint:
if $\exists j \in [m] : y_j = x_i$, then $a_i + b_i = (0, \tilde{x}_i, \tilde{y_j})$, otherwise $a_i + b_i = (1, 0, 0)$.

input values $\tilde{x}_i$ and $\tilde{y}_j$ associated to x_i and y_j, respectively. These outputs can be used as input to an MPC protocol to securely evaluate some function f on the tuples $(x_i, \tilde{x}_i) \in X$ and $(y_j, \tilde{y}_j) \in Y$ for which $x_i = y_j$, without revealing the tuples themselves. An ideal Circuit-PSI functionality is given in Figure 1. The first Circuit-PSI protocol by Huang et al. [HEK12] and later improvements thereof [PSZ14, PSSZ15, PSZ18, PSWW18, CO18, FNO19, PSTY19, MRR20] use general-purpose MPC techniques, such as garbled circuits. More recent protocols [PSTY19, RS21, CGS22, CDG+21, RR22, BPSY23] are based on a functionality called an Oblivious Programmable PRF (OPPRF) and have become very practical, taking approximately 16 seconds for sets of around a million items [RR22]. An OPPRF [KMP+17] is a two-party functionality similar to an OPRF, but where the sender can program some inputs of their choice to map to certain outputs of their choice, and the function looks pseudorandom on unprogrammed points. Again the sender remains oblivious to the receiver's inputs/outputs and the receiver remains oblivious to the function. One can construct an OPPRF from an OPRF and a recently introduced tool called an Oblivious Key-Value Store (OKVS) [GPR+21].

Most of the recent contributions in Circuit-PSI focus on improving the efficiency of *individual* protocol executions in the *semi-honest* setting. So far, potential amortizations between multiple executions have not been studied in the Circuit-PSI setting. Moreover, only the Circuit-PSI protocols based on general-purpose MPC techniques can be easily made secure against malicious adversaries, but their performance remains unsatisfactory. We consider several different settings where a sender and/or receiver take part in multiple protocol executions, over which we aim to amortize the parties' computation and communication costs. To this end, we build up a formal framework for semi-honest Circuit-PSI protocols based on OPPRFs, with stand-alone setup phases which can be run in advance and used in multiple executions through an authenticated bulletin board. This framework fits generalizations of several state-of-the-art protocols, and we present instantiations which excel in different settings. In particular, one of our instantiations provides a slight improvement over the state-of-the art Circuit-PSI protocols in the single-execution setting. We present new security proofs for these constructions in this new setting and analyze the amortizations for specific constructions. Additionally, we present a novel Circuit-PSI protocol from a shared-output PRF (SOPRF), together with instantiations in the semi-honest as well as in the malicious setting, which additionally allows re-usable input-independent setup phases. This has promising applications in the unbalanced setting where the sender's input set is significantly larger than the receiver's input set. Notably, the SOPRF-based protocol results in an "almost maliciously" secure Circuit-PSI protocol with linear communication and computation complexity in the size of the parties' input sets. The only caveat here is that a malicious receiver is able to change their output shares before the secure post-computation of the function f to be evaluated on the intersection. Still, this is a major step on the way towards efficient fully maliciously secure Circuit-PSI.

1.1 Our Contributions

In this work we make a first step in formally studying Circuit-PSI protocols that enable the reuse of data and computation between multiple executions, thereby amortizing costs over

several executions. To this end, we build up a formal framework to construct Circuit-PSI protocols from an oblivious programmable PRF (OPPRF), where we introduce setup phases which are run in advance by a single party on their private data and can be re-used in multiple executions. An OPPRF can in turn be constructed from an oblivious PRF (OPRF) and an oblivious key-value store (OKVS), for which we again present a framework with re-usable setup phases. On the lowest level, we extend several OPRF protocols with setup phases, and prove their security is preserved when the setup phase is used in multiple executions. Formally, the setup outputs are shared through an authenticated bulletin board and can be referred to and used within several executions of the online protocol phase between sender and receiver.

Vector OPRF. Our first observation is that whereas recent Circuit-PSI protocols [RS21, RR22, BPSY23] rely on an OPRF that uses the same key to evaluate each of the receiver's items, the original protocol from [PRTY19] uses an OPRF which uses a different key for each of the receiver's items. We formalize this as a *vector* OPRF (VOPRF) in Section 3.1 and add the option for re-usable sender and receiver setup phases. A direct benefit of this formalization is that it is naturally satisfied by the VOLE-based OPRF of Bui and Couteau [BC22], which is more efficient in terms of communication and computation compared to the VOLE-based OPRF of Rindal and Schoppmann [RS21] that is used in the state-of-the art Circuit-PSI protocols [RR22, BPSY23]. We additionally extend the protocol from [BC22] to satisfy malicious security and exploit the programmability of the underlying VOLE functionality to allow for a re-usable receiver setup phase in Section 3.2. In section 3.3 we revisit the maliciously secure 2Hash Diffie-Hellman-based OPRF of Jarecki et al. [JKK14, JKKX16]. We present the protocol in terms of group actions to pave the way for potential future post-quantum instantiations, and extend the protocol with a re-usable setup phase for the sender as well as for the receiver.

Vector OPPRF. Moving one level up, we generalize the ideal OPPRF functionality from [RS21] to use a different key for each of the receiver's items and extend it with the option for re-usable sender and receiver setup phases. We formalize this as a *vector* OPPRF (VOPPRF) in Section 4.1. The protocol realizing this functionality from a VOPRF and an OKVS in Section 4.2 is a natural generalization of the protocol from [RS21], but to the best of our knowledge a formal proof of this construction has not been covered before in the literature. The main idea behind this protocol is that to program (y, z), the sender encodes an OKVS $\boldsymbol{P} \leftarrow \mathsf{Encode}(y, z - F'(y))$, where F' is the underlying OPRF, and sends $\boldsymbol{P}$ to the receiver. The receiver obtains $F'(x)$ from the OPRF protocol and computes $F'(x) + \mathsf{Decode}(\boldsymbol{P}, x)$, which equals z if x is equal to some programmed value y, and a random value otherwise.

We observe that when the underlying VOPRF protocol allows the sender to use the same function during multiple executions, they can re-use their OKVS encoding during multiple executions. In this case the sender's cost during subsequent executions scales only with the size of the receiver's set. The resulting protocol is secure against malicious receivers, but only against semi-honest senders. The main way a malicious sender can deviate from the protocol is by sending a maliciously constructed encoding vector $\boldsymbol{P}$, which we discuss further in Section 4.3.

Core-PSI from Vector OPPRF. In Section 5.1 we introduce our novel Core-PSI functionality with re-usable sender and receiver setup phases. This functionality mainly differs from Circuit-PSI in the sense that the output of the protocol consists of secret shares of 0 for items in the intersection and secret shares of a random value otherwise, as opposed to secret shares of 1 for items outside of the intersection as Circuit-PSI outputs. We opted for this formalization for two main reasons. Firstly, many Circuit-PSI protocols

internally first obtain this output and then perform a final comparison, thus can be viewed as Core-PSI protocols without loss of generality: The omitted final comparison can instead be included in the post-processing function f to be MPC-evaluated over the intersection. Secondly, Core-PSI can lead to more efficient solutions than Circuit-PSI when the post-processing function does not require any direct element comparisons. For instance, checking whether at least one, or all, of the elements in a subset belong to at least one, or all, of a subset of senders' sets (see Section 8). Core-PSI therefore provides us with a more general and modular way to study Circuit-PSI protocols.

Our Core-PSI protocol in Section 5.2 follows the same structure as the Circuit-PSI protocol from [PSTY19, RS21], except that we extend it with options for sender and receiver setup phases, and replace the final secure comparison step by a rerandomization step. The main idea behind the protocol is as follows. The receiver maps their items to a Cuckoo hash table T. The sender constructs a simple hash table using the same hash functions, and programs an OPPRF to map all items in each bucket k to the same random value s_k. The receiver obtains an OPPRF evaluation for each bucket T_k, which will equal s_k if there exists a matching item in the sender's set, and equals an independent random value otherwise. Unfortunately, even when the underlying VOPPRF protocol allows a sender setup phase, the sender should not re-use this setup phase in multiple executions with corrupt receivers; since they will be able to notice collisions in the OPPRF outputs if their Cuckoo hash tables T_k, T'_k contain different items that the sender mapped to the same bucket. Just as the Circuit-PSI protocols from [PSTY19, RS21], our Core-PSI protocol is not secure against malicious parties. Malicious receivers are able to supply a maliciously constructed Cuckoo hash table, whereas malicious senders can supply a maliciously constructed simple hash table or even arbitrarily deviate from the underlying OPPRF protocol as described before. We describe this in more detail in Section 5.3.

Core-PSI from Shared-Output PRF. To tackle the above issues with respect to re-usability of the sender's setup phase and malicious adversaries, we present a novel Core-PSI protocol in Section 6. The main tool we use for this protocol is a shared-output PRF (SOPRF), which is a functionality that, on input a key K from the sender and an input x from the receiver, outputs an additive secret sharing of $F(K, x)$, where F is a PRF. With this in hand, we can obtain a relatively simple Core-PSI protocol as follows. The sender samples a key K, encodes their items $y \in Y$ in an OKVS $\boldsymbol{P} \leftarrow \mathsf{Encode}(y, F(K, y))$, and sends $\boldsymbol{P}$ to the receiver. The receiver inputs their items $x \in X$ and the sender inputs K to the SOPRF functionality, such that they obtain secret shares $r_x + s_x = F(K, x)$. The receiver can now compute $r_x - \mathsf{Decode}(\boldsymbol{P}, x)$, which together with s_x will form a secret sharing of 0 if x is equal to some $y \in Y$, and a secret sharing of a random value otherwise.

Surprisingly, the above protocol immediately solves multiple shortcomings of the VOPPRF-based protocol. Since the receiver never learns any evaluations of the PRF F under key K, only random secret shares, the target values encoded in the OKVS $\boldsymbol{P}$ will look pseudorandom to the receiver, even when the encoding is re-used in multiple executions. Semi-honest security of our protocol is proven in Section 6.1, where we also present an efficient protocol to realize the SOPRF functionality based on evaluating the Dodis-Yampolskiy PRF [DY05] inside MPC. Additionally, the parties no longer need to perform the Cuckoo/simple hashing step, since the SOPRF functionality automatically aligns the output shares with the indexing of the receiver's set, which limits the avenues for attack for malicious parties. The main remaining obstacle towards malicious security is that the sender can supply any arbitrary vector $\boldsymbol{P}$ instead of a valid OKVS encoding, and the simulator has no way of extracting the "effective" pairs $(y, F(K, y))$ encoded in the OKVS. To overcome this, we leverage a trick introduced by Garimella et al. [GPR+21]. When the function F is modeled as a random oracle, the simulator can observe the queries (K, y) made to $F(\cdot)$ and check for which of these the relation $\mathsf{Decode}(\boldsymbol{P}, y) = F(K, y)$ is

satisfied. By imposing an upper bound on the length of the vector $\boldsymbol{P}$, we can guarantee that only a bounded number of "effective" input elements will be extracted in this way. Ultimately, we obtain a maliciously secure Core-PSI protocol with re-usable sender setup phase, whose security is proven in Section 6.2. We do have to note that the protocol does not succeed to output *authenticated* secret-shares of the intersection. More precisely, a malicious receiver is able to corrupt their output shares before moving on to a potential post-computation phase, without the sender being able to notice this.

Unbalanced Core-PSI. In the setting where there is a sender/server with a large input set Y and multiple receivers/clients with relatively small input sets X_i, we recognize that it can be very useful for the server to be able to re-use their setup phase in order to amortize communication and computation costs. However, the communication complexity of our Core-PSI protocols scales linearly in the size of the server's set, which might be prohibitive for resource-constrained clients. Inspired by the recent protocol by Hetz et al. [HSW23] for unbalanced plain PSI, we suggest to combine our Core-PSI protocols with private information retrieval (PIR) to achieve communication complexity which scales sublinearly in the server's set size. More precisely, the idea is that, instead of the server sending their OKVS encoding $\boldsymbol{P}$, the client uses a PIR scheme to query the entries of $\boldsymbol{P}$ needed to decode their values. Combined with the sparse structure of the recent OKVS construction from Bienstock et al. [BPSY23], this means that the client only needs to issue a single PIR query to decode an item from the OKVS. More details can be found in Section 7.

From Core-PSI to Circuit-PSI. To get from our Core-PSI protocols to the familiar Circuit-PSI functionality in Figure 1, one can simply re-insert the final secure comparison step at the end of the protocol (see Section 8). In this way we obtain the following incidental improvements over previous works. By instantiating our VOPPRF-based Core-PSI protocol from Section 5 with the VOLE-based VOPRF from Section 3.2 and a state-of-the art OKVS construction, one obtains a slight improvement in terms of communication as well as computation with respect to the state-of-the-art semi-honest Circuit-PSI protocols [RR22, BPSY23]. By instantiating the random oracle by a cryptographic PRF and instantiating the SOPRF functionality by a maliciously-secure MPC protocol for evaluating this PRF in our SOPRF-based Core-PSI protocol from Section 6.2, we obtain an "almost maliciously" secure Circuit-PSI protocol with linear communication and computation complexity. The only shortcoming of this protocol is that a malicious receiver is able to change their output shares without the sender being able to notice this during a potential post-computation phase. Still this provides the first major step on the way towards fully maliciously-secure Circuit-PSI since the protocol with $O(n \log n)$ complexity from Huang et al. [HEK12].

Amortization Savings. Coming back to the main objective of this work, in Section 9, we study the amortization savings that the protocols in our framework achieve in different settings with multiple Core-PSI executions:

- **1-to-N**: Single sender using the same input set with multiple receivers;
- **N-to-1**: Multiple senders with a single receiver who uses the same input set;
- **N-to-N**: Multiple senders and receivers, each re-using their input set;
- **N-query**: Multiple executions between the same sender using the same input set with the same receiver who uses typically small and distinct query sets.

From this we can conclude that each of the presented instantiations of our protocols is favorable in at least one of the above multiple-execution settings, depending on the parameters and security model required by the application setting.

1.2 Applications

The main motivating application for this work is to improve customer risk assessment procedures, which requires a lot of, often duplicate, effort from financial institutions. We envision several usecases within this domain for the different settings covered in this work:

- *1-to-N & N-query*: A large data-providing party such as a chamber of commerce, from which several organizations wish to (repeatedly) query data regarding their customers in a privacy-preserving way;
- *N-to-1 & N-to-N*: Organizations wish to improve their customer risk analysis with private data from other organizations, especially to detect important discrepancies.

Additionally, the 1-to-N and N-query settings are very relevant to the setting where the receivers' sets are significantly smaller than the sender's, e.g., for private contact discovery [KRS+19, HSW23] and password breach checking [CMdG+21, LKLM21]. In Section 7, we explore further steps that one can take next to our amortizations to obtain an efficient Circuit-PSI protocol in this unbalanced setting.

The N-to-1 and N-to-N setting could furthermore see applications in, e.g., privacy-preserving ridesharing [HOS17], private blacklist checking [MPC18] and money-laundering detection [CDG+21]. Circuit-PSI enables to strengthen privacy in all these applications by only revealing the items in the intersection that satisfy some additional predicates.

1.3 Related Work

We use the main structure of constructing a Circuit-PSI protocol using an OPPRF from [PSTY19], which was subsequently also used in [RS21, RR22, BPSY23]. The main construction of an OPPRF from an OPRF and an additional data structure is due to [KMP+17] and was subsequently also used in [RS21, GPR+21, RR22, BC22, BPSY23]. The conditions that this data structure needs to satisfy were formalized by [GPR+21] under the definition of an OKVS, and it was mentioned by [GPR+21, BC22, RR22, BPSY23] that it is possible to construct an OPPRF from any OPRF and an OKVS, but to the authors' knowledge no general proof of this claim has appeared in the literature before. Our VOLE-VOPRF is an adaptation of [BC22], our CGA-VOPRF an adaptation of [JKKX16], and our SOPRF is based on the technique of evaluating a PRF using MPC, which has been used before in, for example, [PSSW09, ARS+15, GRR+16, KLS+17, MRR20, FNO22]. To the authors' knowledge, amortizing communication and computation over multiple protocol executions has *not* been studied before in the context of Circuit-PSI.

In the context of plain PSI, amortizations have been studied before, mainly for PSI between servers with a large input set and multiple clients with relatively small input sets [KLS+17, KRS+19, HSW23]. Kiss et al. [KLS+17] introduce a framework for this setting, where the complexity of the online phase only depends on the smaller set. They adapt OPRF-based PSI protocols based on various existing OPRFs [CT10, Mea86, NR97, HL08, PSSW09]. OPRF protocols that fit the framework of [KLS+17] coincide with OPRF protocols allowing sender's key reuse with multiple receivers ($\mathsf{mr} = \mathsf{yes}$), see Appendix E how these fit in our framework. Kales et al. [KRS+19] and Hetz et al. [HSW23] introduce further improvements to [KLS+17]. Notably, Hetz et al. [HSW23] use OPRF-based PSI in combination with PIR, which we extend to the Circuit-PSI setting in Section 7. The combination of PIR and PSI was earlier proposed by Demmler et al. [DRRT18]. All these works only consider PSI, and results do not directly apply to the Circuit-PSI setting.

Furthermore, there are several unbalanced PSI protocols [CLR17, CHLR18, CMdG+21] and unbalanced Circuit-PSI protocols [LPR+21, SJ23] based on fully homomorphic encryption (FHE), which achieve sublinear communication complexity in the large set size, but suffer from high computation overhead for the sender. Concurrently, Hao et al. [HLP+23]

constructed an unbalanced Circuit-PSI protocol from a primitive they call oblivious key-value retrieval. The idea is to let the receiver in the VOPPRF protocol (see Figure 9) decode the OKVS encoding using PIR queries rather than the sender sending the OKVS encoding to the receiver. This is comparable to the approach we propose in Section 7. The main difference is that we suggest to use the OKVS construction of Bienstock et al. [BPSY23], which can be decoded using a single PIR query, as opposed to using a garbled Cuckoo table-like OKVS (such as [RR22]) as [HLP+23] suggests.

Independently from our work, Qiu et al. [QYYZ22] noted that the programmability of the PCG for VOLE correlations from [BCG+19b] can be exploited to improve upon the communication and computation complexity of the central party in the maliciously secure plain multi-party PSI protocol from [GPR+21], using the OPRF from [RS21]. They only consider the setting where the central party directly learns the intersection, and their results do not apply to the Circuit-PSI setting.

Finally, all previous works on Circuit-PSI target semi-honest security. The protocol by Huang et al. [HEK12] can achieve malicious security by using an actively-secure MPC protocol to compute the circuit, but this results in a protocol with $O(n \log n)$ complexity in the input set size n. Our protocol in Section 6.2 manages to achieve "almost malicious" security while still having linear communication and computation complexity. It does unfortunately not achieve fully malicious security since a corrupted receiver can change their output shares before the post-computation phase without the sender being able to notice this, but provides a major step towards a fully secure solution with linear complexity.

1.4 Acknowledgments

The authors would like to thank Brett Hemenway Falk and Daniel Noble for helpful discussions about shared-output PRFs. In particular, we would like to thank Daniel Noble for pointing us in the direction of the Dodis-Yampolskiy PRF being especially suited for evaluation inside MPC.

2 Preliminaries

We denote $\kappa \in \mathbb{N}$ for the computational security parameter and $\lambda \in \mathbb{N}$ for the statistical security parameter. We denote $\mathcal{D} \approx_c \mathcal{D}'$ for the computational indistinguishability of distribution ensembles $\mathcal{D} := \{\mathcal{D}_\kappa\}_{\kappa\in\mathbb{N}}$, $\mathcal{D}' := \{\mathcal{D}'_\kappa\}_{\kappa\in\mathbb{N}}$ and $\mathcal{D} \approx_s \mathcal{D}'$ for their statistical indistinguishability. For a ring R and the R-module R^n, $n \in \mathbb{N}$, we denote $\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \sum_{i=1}^{n} x_i y_i$ for the "inner product" of $\boldsymbol{x}, \boldsymbol{y} \in R^n$. We extend this componentwise to products $R \times S$ for rings R and S.

2.1 Security Model

All of our protocols and functionalities are parameterized by multiple sending parties $\mathcal{S}_j$, $j \in [N_\mathcal{S}]$ and multiple receiving parties $\mathcal{R}_i$, $i \in [N_\mathcal{R}]$, of which each pair can decide to run an execution of the protocol between themselves. We use sid to indicate the *session identifier*, which is a unique string for each protocol execution that takes place between a sender and a receiver. Each sender $\mathcal{S}_j$ holds a unique *sender identifier* $\mathsf{id}_{\mathcal{S}_j}$ and each receiver $\mathcal{R}_i$ holds a unique *receiver identifier* $\mathsf{id}_{\mathcal{R}_i}$. We moreover use σ to indicate the *sender setup phase identifier* and ρ to indicate the *receiver setup phase identifier*.

For ease of exposition, we consider the corruption model of an adversary $\mathcal{A}$ which can maliciously corrupt either several receiving parties $\mathcal{R}_i$, or several sending parties $\mathcal{S}_i$, but never simultaneously corrupts sending and receiving parties. By combining the simulator for a subset of corrupt senders and the simulator for a subset of corrupt receivers, one can obtain a simulator for the setting where both sending and receiving parties are corrupted.

Whether a protocol is secure against semi-honest or malicious adversaries is indicated in the corresponding theorem statement. Security is proven via the standard simulation-based paradigm (see e.g., [Lin17]). For our VOPRF (see Section 3) and VOPPRF (see Section 4) protocols either all sending parties or all receiving parties can simultaneously be corrupted. For our VOPPRF-based Core-PSI protocol (see Section 5), all sending parties can simultaneously be corrupted, but at most one receiving party can be corrupted at the same time. For our SOPRF-based Core-PSI protocol (see Section 6) again either all sending parties or all receiving parties can simultaneously be corrupted.

2.2 Oblivious Key-Value Store (OKVS)

An oblivious key value store (OKVS) [GPR+21] can be used to "program" certain points of the sender's choosing into the OPRF, resulting in a functionality called an oblivious programmable PRF (OPPRF), which is a frequent building block of modern Circuit-PSI protocols. The concept of an oblivious key-value store (OKVS) was introduced by Garimella et al. [GPR+21] as a generalization of the polynomial interpolation techniques that are used in many PSI protocols [FNP04, FIPR05, KS05, DMRY09, HN10, MPP10, Haz15, HV17, FHNP16, CDJ16, KMP+17, GN19, GS19, PRTY19, KRTW19, PSTY19, CDG+21].

Recent OKVS constructions [RR22, BPSY23] have become very efficient, taking $O(n\lambda)$ time to encode n key-value pairs, $O(\lambda)$ time to decode a single key, and having constant expansion factor $\eta := q(n)/n$. For our OPPRF construction, we require an OKVS with some extra properties as defined below, and note that all of the most efficient OKVS constructions [RR22, BPSY23] satisfy these properties.

Definition 1. A *linear double oblivious key value store* $\mathsf{OKVS} = (\mathsf{Encode}, \mathsf{Decode})$ is defined with respect to a key space $\mathcal{K}$, value space $\mathcal{V}$, statistical security parameter $\lambda \in \mathbb{N}$, randomness space $\{0,1\}^\kappa$ and expansion function $q : \mathbb{N} \to \mathbb{N}$, as follows:

- Encode : takes as input a set of key-value pairs $L \in (\mathcal{K} \times \mathcal{V})^n$ and randomness $\theta \in \{0,1\}^\kappa$, and outputs a vector $\boldsymbol{P} \in \mathcal{V}^{q(n)}$ or an error indicator $\perp$.
- Decode : takes as input a vector $\boldsymbol{P} \in \mathcal{V}^m$, a key $k \in \mathcal{K}$ and randomness $\theta \in \{0,1\}^\kappa$, and outputs a value $v \in \mathcal{V}$.

That satisfy:

- **Correctness**: For all $L \in (\mathcal{K} \times \mathcal{V})^n, \theta \in \{0,1\}^\kappa$ with n distinct keys: $\mathsf{Encode}(L, \theta) = \boldsymbol{P} \neq \perp \implies \forall (k, v) \in L$: $\mathsf{Decode}(\boldsymbol{P}, k; \theta) = v$.
- **Low error probability**: For all $L \in (\mathcal{K} \times \mathcal{V})^n$ with n distinct keys: $\Pr_{\theta \xleftarrow{\$} \{0,1\}^\kappa}[\mathsf{Encode}(L; \theta) = \perp] \leq 2^{-\lambda}$.
- **Double Obliviousness:** For any $\{k_1, \ldots, k_n\} \subset \mathcal{K}$ of n distinct keys and any $\theta \in \{0,1\}^\kappa$, then for $v_1, \ldots, v_n \xleftarrow{\$} \mathcal{V}$, if Encode does not output $\perp$: $\{P \leftarrow \mathsf{Encode}(\{(k_i, v_i)_{i \in [n]}\}; \theta)\} \approx_s \{P \xleftarrow{\$} \mathcal{V}^{q(n)}\}$.
- **Linearity:** There exists a public function family $\{d_{q(n)} \mid n \in \mathbb{N}\}$, with $d_{q(n)} : \mathcal{K} \times \{0,1\}^\kappa \to \mathcal{V}^{q(n)}$ such that for all $n \in \mathbb{N}$, $\boldsymbol{P} \in \mathcal{V}^{q(n)}$, $k \in \mathcal{K}$ and $\theta \in \{0,1\}^\kappa$ it holds that: $\mathsf{Decode}(\boldsymbol{P}, k; \theta) := \langle \boldsymbol{P}, d_{q(n)}(k; \theta) \rangle$.

In a protocol where a malicious party supplies an OKVS encoding $\boldsymbol{P}$, the simulator often needs to be able to extract the items that are encoded in $\boldsymbol{P}$. One way to do this is to require the party to encode pairs $(k_i, H(k_i))$ for keys k_i. When $H(\cdot)$ is modeled as a random oracle, the simulator can observe the queries $(k, H(k))$ to $H(\cdot)$ and test which of these satisfy $\mathsf{Decode}(\boldsymbol{P}, k) = H(k)$. In such a scenario, we would like to be able to upper bound the number of items a malicious party can fit in the OKVS encoding, which is formalized through the following *overfitting game* [GPR+21].

Definition 2. Let $(\mathsf{Encode}, \mathsf{Decode})$ be an OKVS with parameters chosen to support n items and let $\mathcal{A}$ be a PPT adversary. Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a public function and $(\boldsymbol{P}, \theta) \leftarrow \mathcal{A}^H(1^\kappa)$ (i.e., under the condition that $|\boldsymbol{P}| = q(n)$). Define the set

$$X := \{k \in \mathcal{K} \mid \mathcal{A} \text{ queried } H \text{ at } k \text{ and } \mathsf{Decode}(\boldsymbol{P}, k; \theta) = H(k)\}.$$

Then $\mathcal{A}$ wins the (n, n')-OKVS overfitting game if $|X| > n'$. We say the (n, n')-OKVS overfitting problem is hard if no PPT $\mathcal{A}$ wins the game with non-negligible probability.

Pinkas et al. [PRTY20, App. A] showed that when H is modeled as a random oracle with output length ℓ, then the probability of an adversary making Q queries to H and winning the (n, n')-OKVS overfitting game is unconditionally upper bounded by $\binom{Q}{n'} \cdot 2^{(q(n)-n')\ell}$.

2.3 Cuckoo Hashing

To construct a Circuit-PSI protocol, an OPPRF is often used in combination with a data structure called a *Cuckoo hash table*. In a Cuckoo hash table, t hash functions $h_1, \ldots, h_t$ are used to map n items into a table T consisting of $m = \epsilon n$ bins (with $\epsilon > 1$), such that each bin contains at most one item and each item x is placed at precisely one of the locations $h_1(x), \ldots, h_t(x)$ [PR01]. The insertion procedure works roughly as follows: each item x is first attempted to be placed at location $h_1(x)$; if $T[h_1(x)]$ already contains an item y, then y is evicted and attempted to be placed at a different location $h_i(y) \neq h_1(x)$; this procedure is repeated until no more evictions are necessary or until a threshold number of relocations has been performed, in which case the item is placed in a special *stash*. We refer to the latter event as failure to place the given element. We follow recent Circuit-PSI protocols [PSTY19, RS21] by using the extrapolated experimental analysis of Pinkas et al. [PSZ18] to argue that when $t > 2$ hash functions together with an appropriate number of bins are used, a failure probability of $< 2^{-40}$ can be guaranteed, which makes it unnecessary to use a stash. Their analysis showed that $\epsilon = 1.27, 1.09, 1.05$ is sufficient for $t = 3, 4, 5$, respectively [PSZ18].

2.4 Cryptographic Group Actions

In this section we briefly recap the concept of a hard homogeneous space (HHS) from Couveignes [Cou06]. We make use of this framework for our generalization of the 2HashDH-OPRF of Jarecki et al. [JKK14, JKKX16] to the group action setting in Sec. 3.3.

Definition 3. Let $\mathbb{H}$ be a finite abelian group and U a set. We say that $\mathbb{H}$ *acts regularly* on U or that U is a *principal homogeneous space* for $\mathbb{H}$ if there is a map $\star : \mathbb{H} \times U \to U$ such that:

- **Identity:** Let $1_\mathbb{H} \in \mathbb{H}$ be the identity. Then for any $x \in U$, we have $1_\mathbb{H} \star x = x$.
- **Compatibility:** For any $g, h \in \mathbb{H}$ and any $x \in U$, we have $(gh) \star x = g \star (h \star x)$.
- **Regularity:** For any $x, y \in U$ there exists a *unique* $g \in \mathbb{H}$ such that $g \star x = y$.

We denote the group action by the tuple $(\mathbb{H}, U, \star)$, note that in particular $|\mathbb{H}| = |U|$.

We present a slightly adapted version of the definition of a hard homogeneous space from Couveignes [Cou06], inspired by the framework of [AFMP20], in which we additionally require the existence of an efficiently computable hash function $H : \{0,1\}^* \to U$ for which it is hard to find a path between hashed points.

Definition 4. Let $(\mathbb{H}, U, \star)$ be a principal homogeneous space. We say that U is a *hashable hard homogeneous space* for $\mathbb{H}$ if one can efficiently:

- Decide validity and equality of representations of elements of $\mathbb{H}$.

Figure 2 Ideal Authenticated Bulletin Board functionality $\mathcal{F}_{\mathsf{BB}}$

For a set of parties $\mathcal{P}$, the bulletin board starts with an empty set of messages $\mathcal{M}$.

- On receiving message $(\mathtt{post}, \mathsf{sid}, \mathsf{id}_m, m)$ from party $P_i \in \mathcal{P}$:
  If $\nexists m' : (\mathsf{id}_{P_i}, \mathsf{sid}, \mathsf{id}_m, m') \in \mathcal{M}$ then append $(\mathsf{id}_{P_i}, \mathsf{sid}, \mathsf{id}_m, m)$ to $\mathcal{M}$ and send $(\mathtt{posted}, P_i, \mathsf{sid}, \mathsf{id}_m, m)$ to the adversary.
- On receiving a message $(\mathtt{read}, \mathsf{sid})$ from a party $P_i \in \mathcal{P}$ return $\mathcal{M}$.

- Compute gh and g^{-1} for any $g, h \in \mathbb{H}$.
- Sample elements $g \xleftarrow{\$} \mathbb{H}$ statistically negligibly close to uniform.
- Decide validity and equality of representations of elements of U.
- Compute $g \star x$ for any $g \in \mathbb{H}$ and any $x \in U$.
- Compute $H(s) \in U$ for any $s \in \{0,1\}^*$.

Additionally, it should be intractable to solve the following problems:

- **Vectorization:** Given $x, y \in U$, find $g \in \mathbb{H}$ such that $g \star x = y$.
- **Parallelization:** Given $x, x', y \in U$ such that $x' = g \star x$, find $y' = g \star y$.
- **Hashing path:** Given $s, t \in \{0,1\}^*$, find $g \in \mathbb{H}$ such that $g \star H(s) = H(t)$.

It is clear that the parallelization problem reduces to the vectorization problem. It has been shown by Montgomery and Zhandry [MZ22] that the problems are in fact *quantumly* equivalent. Moreover, note that the vectorization problem and the hashing path problem are *classically* equivalent when H is a random oracle.

Two of the main examples of principal homogeneous spaces in cryptography are prime order cyclic groups and supersingular elliptic curves, whose suitability is further discussed in Appendix C.

2.5 Bulletin Board

Our Core-PSI framework introduces setup phases whose outputs can be obtained by other parties on demand. In practice one imagines solutions could use various means like CDNs, or broadcasts. In our protocols we use an idealized functionality in the form of publicly readable bulletin boards, where messages are authenticated and cannot be erased after being posted. As these can be realized in a similar way as for [CGJ+17], like Google's Certificate Transparency Project [Goo13], Cloudflare's Nimbus [Clo18], or a blockchain-based ledger [BGM16]. Alternatively, one could choose to use a public authenticated broadcast channel indeed reducing communication complexity. There are several possible acceptable formalizations, we reuse the one from [BDD20]. See Figure 2 for the ideal functionality $\mathcal{F}_{\mathsf{BB}}$.

2.6 Arithmetic Black Box

The arithmetic black box (ABB) model was introduced by Damgård and Nielsen [DN03] as a tool to prove the security of MPC protocols for general *reactive* functionalities, in which the parties might receive some intermediate results of the computation and can use these in the rest of the computation. The ABB basically serves as a general-purpose computer for secure computation, to which the parties can supply inputs and ask to perform any feasible computational task, as detailed in Figure 3.

We will often omit the arguments $\mathtt{var}$ and $\mathtt{out}$ if they are clear from the context. Sometimes, when it is not clear from the context, we will denote $[\![x]\!]_q$ for a sharing over a field $\mathbb{F}_q$. Finally, we will use the shorthand $[\![x]\!]_{q,1} := \mathtt{out}_1$, $[\![x]\!]_{q,2} := \mathtt{out}_2$ for the output of $\mathbf{OutputShares}([\![x]\!]_q, \mathtt{out}_1, \mathtt{out}_2)$.

Figure 3 Arithmetic Black Box Functionality $\mathcal{F}_{\mathsf{ABB}}$

Parameters: Inputs are elements of some input space $\mathcal{X}$ and outputs are elements of a finite ring R. Parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ each holding an identifier $\mathsf{id}_{\mathcal{P}_i}$. The value $[\![x]\!]$ is an identifier for the value x that is stored by the functionality.

Functionalities:

- **Input**$(x, \mathsf{id}_{\mathcal{P}}, \mathtt{var})$: Receive x from party $\mathsf{id}_{\mathcal{P}}$ and store it as $[\![\mathtt{var}]\!]$.
- **Add**$([\![x]\!], [\![y]\!], \mathtt{out})$: Compute $z := x + y$ and store z in $[\![\mathtt{out}]\!]$.
- **Multiply**$([\![x]\!], [\![y]\!], \mathtt{out})$: Compute $z := x \cdot y$ and store z in $[\![\mathtt{out}]\!]$.
- **Random**$([\![\mathtt{out}]\!])$: Sample $r \xleftarrow{\$} R$ and store r in $[\![\mathtt{out}]\!]$.
- **Eval**$^F([\![x]\!], [\![k]\!], \mathtt{out})$: Compute $z := F(k, x)$ and store z in $[\![\mathtt{out}]\!]$.
  % F is either a public oracle or a public function parameter to the functionality
- **Rerand**$([\![(x_1, x_2)]\!], (\mathtt{out}_1, \mathtt{out}_2))$: Sample $r_1, r_2 \xleftarrow{\$} R$, compute $z_1 := r_1 \cdot x_1$ and $z_2 := x_2 + r_2 \cdot x_1$, and store (z_1, z_2) in $[\![(\mathtt{out}_1, \mathtt{out}_2)]\!]$.
- **Equality**$([\![(x_1, x_2)]\!], (\mathtt{out}_1, \mathtt{out}_2))$: If $x_1 = 0$, put $z_1 := 0$, and $z_1 := 1$ otherwise. Put $z_2 := z_1 \cdot x_2$ and store (z_1, z_2) in $[\![(\mathtt{out}_1, \mathtt{out}_2)]\!]$.
- **OutputShares**$([\![x]\!], \mathtt{out}_1, \mathtt{out}_2)$: Sample uniform shares $\mathtt{out}_1, \mathtt{out}_2 \xleftarrow{\$} R$ such that $\mathtt{out}_1 + \mathtt{out}_2 = x$ and output one of these to each of the parties supplying the input.
- **Output**$([\![x]\!], \mathtt{var})$: Output the value x to all parties as $\mathtt{var}$.

3 Vector Oblivious PRF (VOPRF)

In this section we present our ideal functionality for a vector oblivious pseudorandom function (VOPRF). Subsequently we present two protocols and prove that they realize this ideal functionality: a Vector Oblivious Linear Evaluation (VOLE) based construction, which allows a receiver setup phase (Section 3.2), and a Cryptographic Group Action (CGA) based construction, which allows a sender and a receiver setup phase (Section 3.3). We discuss some other OPRF constructions and the setup phases we expect them to allow in Appendix E.

3.1 Ideal Functionality

Our ideal VOPRF functionality $\mathcal{F}_{\mathsf{voprf}}$ in Figure 4 can be seen as an extension of the OPRF functionality of Rindal and Schoppmann [RS21]. The main difference with their functionality is that our ideal functionality uses an independently random function to evaluate each entry of the receiver's input vector; hence the name *vector* OPRF. We additionally expand upon the functionality from [RS21] by allowing the sender and receiver to run a setup phase, which they can securely re-use during subsequent executions. If a sender setup phase is allowed ($\mathsf{mr} = \mathsf{yes}$), the sender can enforce that a certain random function is used, while in protocols that do not allow a sender setup phase ($\mathsf{mr} = \mathsf{no}$), the sender has no control which function is being used.

Our VOLE-based protocol in Section 3.2 realizes the $\mathcal{F}_{\mathsf{voprf}}$ functionality against malicious parties with $\mathsf{mr} = \mathsf{no}$. The CGA-based protocol in Section 3.3 realizes a slightly different functionality $\mathcal{F}^*_{\mathsf{voprf}}$, as detailed in Appendix A, against malicious parties with $\mathsf{mr} = \mathsf{yes}$. Here a malicious receiver gets to change their input vector on the fly, but can still only receive a single evaluation from each function. In particular, this implies that it realizes the functionality $\mathcal{F}_{\mathsf{voprf}}$ against semi-honest parties with $\mathsf{mr} = \mathsf{yes}$.

3.2 Vector Oblivious Linear Evaluation VOPRF (VOLE-VOPRF)

Our first OPRF construction is a maliciously secure adaptation of the semi-honest subfield VOLE-based OPRF of Bui and Couteau [BC22]. We essentially need only 3κ extra bits

Figure 4 Ideal VOPRF functionality $\mathcal{F}_{\mathsf{voprf}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding a unique identifier $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on an input space $\mathcal{X}$ and a finite ring R as output space. PRF is an internal random oracle for random functions $\mathcal{F} = \{f : \mathcal{X} \to R\}$. $\mathcal{L}^{\mathcal{R}_i}$ is a list of $\mathcal{R}_i$'s setup phases, initialized empty.
Functionality:

- On query $(\texttt{setup}, \rho, X)$ from $\mathcal{R}_i$:
  - If $\exists X : (\rho, X) \in \mathcal{L}^{\mathcal{R}_i}$, do nothing.
  - Else, add (ρ, X) to $\mathcal{L}^{\mathcal{R}_i}$ and send $(\mathsf{id}_{\mathcal{R}_i}, \rho, |X|)$ to $\mathcal{A}$.
- If $\mathsf{mr} = \mathsf{yes}$ then on query $(\texttt{setup}, \sigma, m)$ from $\mathcal{S}_j$, return $(\mathcal{O}^{\mathsf{PRF}(\mathsf{id}_{\mathcal{S}_j}, \sigma, m, k)})_{k=1}^{m}$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \texttt{evaluate}, \rho)$ from $\mathcal{R}_i$:
  - If $\exists X : (\rho, X) \in \mathcal{L}^{\mathcal{R}_i}$, send $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho, |X|)$ to $\mathcal{S}_j$.
  - Else, send abort to $\mathcal{R}_i$ and $\mathcal{S}_j$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \texttt{evaluate}, \sigma)$ from $\mathcal{S}_j$:
  - If $\mathsf{mr} = \mathsf{no}$, sample $F \xleftarrow{\$} \mathcal{F}^{|X|}$.
  - If $\mathsf{mr} = \mathsf{yes}$, set $F := (\mathsf{PRF}(\mathsf{id}_{\mathcal{S}_j}, \sigma, |X|, k))_{k=1}^{|X|}$.
  - Output $(\mathcal{O}^{F_k})_{k=1}^{|X|}$ to $\mathcal{S}_j$ and $(F_k(X_k))_{k=1}^{|X|}$ to $\mathcal{R}_i$.

in communication to achieve malicious security. Their construction can in turn be seen as a generalization of the OT-based OPRF from [KKRT16] replacing OT by sVOLE. On the other hand, our construction is similar to the malicious VOLE-based OPRF of Rindal and Schoppmann [RS21] and its sVOLE-based generalization of Rindal and Raghuraman [RR22]. The main difference is that in our construction the receiver does not first encode their input set into an OKVS, but rather directly sends a masked version of their input set to the sender. This saves computation and roughly saves a multiplicative factor η in communication, where $\eta \approx q(m)/m$ is the expansion factor of the OKVS construction. Our protocol $\Pi_{\mathsf{vole\text{-}voprf}}$ is detailed in Figure 6 and security is proven in Theorem 1.

Remark 1. In order to instantiate our construction, one needs a maliciously secure (programmable) sVOLE protocol Π_{psvole}, realizing the ideal functionality $\mathcal{F}_{\mathsf{psvole}}$ from Figure 5. The semi-honest VOLE generator from Boyle et al. [BCGI18] has been shown to satisfy the programmability condition [BCG+19b, Prop. 59]. Note that this is sufficient for our semi-honestly secure Core-PSI protocol presented in Section 5. We expect that through careful analysis it can be shown that the maliciously secure sVOLE protocol from Boyle et al. [BCG+19a] also satisfies the programmability condition for honest receivers. Note that we do not need to enforce consistency with respect to the receiver's input for maliciously corrupted receivers. We leave it to future work to prove this formally for the protocol of Boyle et al. [BCG+19a] and its various optimizations [WYKW21, CRR21, RRT23][1]. Note that a similar condition is also used in the malicious setting by Qiu et al. [QYYZ22].

Theorem 1. *If $|\mathbb{B}| \geq 2^{\ell}$ with $\ell := \lambda + \log_2 m$ and $|\mathbb{F}| \geq 2^{\kappa}$, the protocol $\Pi_{\mathsf{vole\text{-}voprf}}$ securely realizes the functionality $\mathcal{F}_{\mathsf{voprf}}$ with $\mathsf{mr} = \mathsf{no}$ against malicious adversaries in the random oracle, $\mathcal{F}_{\mathsf{BB}}$-, $\mathcal{F}_{\mathsf{psvole}}$-hybrid model (where H, H' are modeled as random oracles).*

Proof. Executions with the same sender $\mathcal{S}$ and different receivers $\mathcal{R}_i$ are independent, so it suffices to construct a simulator that interacts with a single corrupted receiver $\mathcal{R}$. Moreover, we only have to simulate an interaction with a single honest sender $\mathcal{S}$ since interactions with different honest senders are independent. If the receiver behaves semi-honestly, the

[1] It has recently been shown that Silver codes [CRR21] are insecure and should not be used [RRT23]

Figure 5 Ideal psVOLE functionality $\mathcal{F}_{\mathsf{psvole}}$ ([RS21, Fig. 2], [BCG+19b, Def. 40])

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique a identifier $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on a finite extension field $\mathbb{F}$ and a base field $\mathbb{B}$. $\mathcal{L}^{\mathcal{R}_i}$ is a list of $\mathcal{R}_i$'s setup phases, initialize $\mathcal{L}^{\mathcal{R}_i} := \emptyset$.
Functionality:

- On query $(\texttt{setup}, \rho, m)$ from $\mathcal{R}_i$, if $\exists \boldsymbol{A}^\rho : (\rho, \boldsymbol{A}^\rho) \in \mathcal{L}^{\mathcal{R}_i}$ do nothing; Else, sample $\boldsymbol{A}^\rho \xleftarrow{\$} \mathbb{B}^m$, add $(\rho, \boldsymbol{A}^\rho)$ to $\mathcal{L}^{\mathcal{R}_i}$ and return $\boldsymbol{A}^\rho$ to $\mathcal{R}_i$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \texttt{evaluate}, \rho)$ from $\mathcal{R}_i$: If $\exists \boldsymbol{A}^\rho : (\rho, \boldsymbol{A}^\rho) \in \mathcal{L}^{\mathcal{R}_i}$ with $\boldsymbol{A}^\rho \in \mathbb{B}^m$ then send $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho)$ to $\mathcal{S}_j$; else send abort to $\mathcal{R}_i$ and $\mathcal{S}_j$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \texttt{evaluate})$ from $\mathcal{S}_j$:
  - The functionality samples $\Delta \xleftarrow{\$} \mathbb{F}$, $\boldsymbol{B} \xleftarrow{\$} \mathbb{F}^m$ and sets $\boldsymbol{A} := \boldsymbol{A}^\rho$, $\boldsymbol{C} := \Delta \cdot \boldsymbol{A} + \boldsymbol{B}$.
  - If $\mathcal{S}_j$ is corrupted, query $\Delta', \boldsymbol{B}' \leftarrow \mathcal{A}(\mathsf{sid}, \Delta, \boldsymbol{B})$ and set $\Delta := \Delta'$, $\boldsymbol{B} := \boldsymbol{B}'$, $\boldsymbol{C} := \Delta' \cdot \boldsymbol{A} + \boldsymbol{B}'$.
  - If $\mathcal{R}_i$ is corrupted, query $\boldsymbol{A}', \boldsymbol{C}' \leftarrow \mathcal{A}(\mathsf{sid}, \boldsymbol{A}, \boldsymbol{C})$ and set $\boldsymbol{A} := \boldsymbol{A}'$, $\boldsymbol{C} := \boldsymbol{C}'$, $\boldsymbol{B} := \boldsymbol{C}' - \Delta \cdot \boldsymbol{A}'$.
  - Output Δ, $\boldsymbol{B}$ to $\mathcal{S}_j$ and output $\boldsymbol{A}$, $\boldsymbol{C}$ to $\mathcal{R}_i$.

simulator can act as the ideal $\mathcal{F}_{\mathsf{psvole}}$ functionality to answer the receiver's `setup` and `evaluate` queries. Otherwise, the simulator receives ρ and $\boldsymbol{A}' \in \mathbb{B}^m$, $\boldsymbol{C} \in \mathbb{F}^m$ from the receiver's `evaluate` message to $\mathcal{F}_{\mathsf{psvole}}$ and retrieves $\boldsymbol{Z}^\rho$ from the simulated bulletin board to compute the receiver's effective input set $\boldsymbol{X} := \boldsymbol{Z}^\rho - \boldsymbol{A}'$. The simulator then forwards $\boldsymbol{X}$ to $\mathcal{F}_{\mathsf{voprf}}$ and can use the resulting $F_i(x_i)$ to answer queries to H of the form (i, C_i, x_i, w). The simulator can sample $w^{\mathcal{S}} \xleftarrow{\$} \{0,1\}^\kappa$ and send it to the receiver. If the sender poses a query of the form (i, y, x, w) to H before receiving $w^{\mathcal{S}}$, the simulator aborts, which happens with probability negligible in κ. Consider a hybrid which behaves as a real protocol execution but aborts when the receiver poses a query of the form $(i, \Delta \cdot (x_i - x) + C_i, w)$ for $x \in \mathbb{B}$, $x \neq x_i$. This hybrid aborts with negligible probability in κ since $\Delta \xleftarrow{\$} \mathbb{F}$ from the receiver's point of view, with $|\mathbb{F}| \geq 2^\kappa$. The simulated execution and this hybrid are computationally indistinguishable by our above discussion.

Since a receiver $\mathcal{R}$ can re-use their setup phase (ρ, m) with multiple corrupted senders $\mathcal{S}_i$, we construct our simulator out of subroutines each interacting with a corrupted sender $\mathcal{S}_i$. Moreover, we only have to simulate an interaction with a single honest receiver $\mathcal{R}$ since interactions with different honest receivers are independent. The simulator receives the honest receiver's setup phase identifiers (ρ, m) from the ideal functionality $\mathcal{F}_{\mathsf{voprf}}$ and can sample $\boldsymbol{Z}^\rho \xleftarrow{\$} \mathbb{B}^m$ and add these to the simulated bulletin board. If the sender behaves semi-honestly, the simulator can act as the ideal $\mathcal{F}_{\mathsf{psvole}}$ functionality to answer the sender's `evaluate` query. Otherwise, the simulator receives $\Delta_i \in \mathbb{F}$, $\boldsymbol{B}^i \in \mathbb{F}^m$ from the sender's `evaluate` message to $\mathcal{F}_{\mathsf{psvole}}$. The simulator can sample $c_i^{\mathcal{R}} \xleftarrow{\$} \{0,1\}^\kappa$, and, after receiving $w_i^{\mathcal{S}}$, sample $w_i^{\mathcal{R}} \xleftarrow{\$} \{0,1\}^\kappa$ and program $H'(w_i^{\mathcal{R}}) := c_i^{\mathcal{R}}$. If the sender queried $w_i^{\mathcal{R}}$ to H' before, the simulator aborts, which happens with probability negligible in κ. The simulator can answer queries to H of the form $(j, \Delta_i \cdot (Z_j^{\rho_i} - x) + B_j^i, x, w_i)$ by $F_j^i(x)$ using the $\mathcal{O}^{F_j^i}$ oracles obtained from $\mathcal{F}_{\mathsf{voprf}}$. If the sender poses a query of the form $(j, \Delta_i \cdot (Z_j^{\rho_i} - x) + B_j^i, x, w_i)$ to H before receiving $w_i^{\mathcal{R}}$, the simulator aborts, which again happens with probability negligible in κ. It follows that the simulated execution and the real protocol execution are computationally indistinguishable. □

Figure 6 psVOLE-based VOPRF protocol $\Pi_{\mathsf{vole\text{-}voprf}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding a unique identifier $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on a finite extension field $\mathbb{F}$, input space the base field $\mathbb{B}$ and an output space $\mathcal{Y}$. $H : \{0,1\}^* \to \mathcal{Y}$ and $H' : \{0,1\}^* \to \{0,1\}^\kappa$ are random oracles, $\kappa \in \mathbb{N}$ the computational security parameter.

Subprotocol $\mathsf{Setup}_{\mathcal{R}}$ **with** $\mathcal{R}_i$**-input** $(\rho, \boldsymbol{X})$**:**

- $\mathcal{R}_i$ sends $(\mathtt{setup}, \rho, m)$ to $\mathcal{F}_{\mathsf{psvole}}$, with $m := |\boldsymbol{X}|$, and receives $\boldsymbol{A}^\rho \in \mathbb{B}^m$.
- $\mathcal{R}_i$ computes $\boldsymbol{Z}^\rho := \boldsymbol{X} + \boldsymbol{A}^\rho$ and sends $(\mathtt{post}, \mathsf{sid}, \mathsf{id}_\rho, (\rho, \boldsymbol{Z}^\rho))$ to $\mathcal{F}_{\mathsf{ABB}}$.

Subprotocol $\mathsf{Evaluate}$ **with** $\mathcal{R}_i$**-input** $(\mathsf{id}_{\mathcal{S}_j}, \rho)$ **and** $\mathcal{S}_j$**-input** $\mathsf{id}_{\mathcal{R}_i}$**:**

- $\mathcal{R}_i$ sends $(\mathsf{id}_{\mathcal{S}_j}, \mathsf{sid}, \mathtt{evaluate}, \rho)$ to $\mathcal{F}_{\mathsf{psvole}}$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ abort if they receive abort from $\mathcal{F}_{\mathsf{psvole}}$.
- Otherwise, $\mathcal{S}_j$ receives $(\mathsf{id}_{\mathcal{R}_i}, \mathsf{sid}, \rho)$ and sends $(\mathsf{id}_{\mathcal{R}_i}, \mathsf{sid}, \mathtt{evaluate})$ to $\mathcal{F}_{\mathsf{psvole}}$.
- $\mathcal{S}_j$ receives $\Delta \in \mathbb{F}$, $\boldsymbol{B} \in \mathbb{F}^m$ and $\mathcal{R}_i$ receives $\boldsymbol{A}^\rho \in \mathbb{B}^m$, $\boldsymbol{C} \in \mathbb{F}^m$ satisfying the relation $\boldsymbol{C} = \Delta \cdot \boldsymbol{A}^\rho + \boldsymbol{B}$.
- $\mathcal{S}_j$ sends $(\mathtt{read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ABB}}$ and searches $\mathcal{M}$ for the first message $(\mathsf{id}_{\mathcal{R}_i}, \mathsf{sid}, \mathsf{id}_\rho, (\rho, \boldsymbol{Z}^\rho))$ with $\boldsymbol{Z}^\rho \in \mathbb{B}^m$ or aborts if it does not exist.
- $\mathcal{R}_i$ samples $w^{\mathcal{R}} \xleftarrow{\$} \{0,1\}^\kappa$ and sends $c^{\mathcal{R}} := H'(w^{\mathcal{R}})$ to $\mathcal{S}_j$.
- $\mathcal{S}_j$ aborts if $c^{\mathcal{R}} \notin \{0,1\}^\kappa$, samples $w^{\mathcal{S}} \xleftarrow{\$} \{0,1\}^\kappa$ and sends $w^{\mathcal{S}}$ to $\mathcal{R}_i$.
- $\mathcal{R}_i$ aborts if $w^{\mathcal{S}} \notin \{0,1\}^\kappa$, sends $w^{\mathcal{R}}$ to $\mathcal{S}_j$, and defines $w := w^{\mathcal{R}} + w^{\mathcal{S}}$.
- $\mathcal{R}_i$ outputs $(F_{\Delta,K_i}(x_i))_{i=1}^m$ with $F_{\Delta,K_i}(x_i) = H(i, C_i, x_i, w)$.
- $\mathcal{S}_j$ aborts if $w^{\mathcal{R}} \notin \{0,1\}^\kappa$ or $c^{\mathcal{R}} \neq H'(w^{\mathcal{R}})$, and defines $w := w^{\mathcal{R}} + w^{\mathcal{S}}$.
- $\mathcal{S}_j$ defines the keys $(\Delta, \boldsymbol{K} := \Delta \cdot \boldsymbol{Z}^\rho + \boldsymbol{B})$.
- $\mathcal{S}_j$ outputs $(\mathcal{O}^{F_{\Delta,K_i}})_{i=1}^m$ with $F_{\Delta,K_i}(x) := H(i, K_i - \Delta \cdot x, x, w)$.

3.3 Cryptographic Group Action VOPRF (CGA-VOPRF)

Jarecki et al. provide a construction of an OPRF protocol for the 2HashDH PRF $F_k(x) := H'(H(x)^k, x)$ in a cyclic group $\mathbb{G}$ of prime order p, where $H : \mathcal{X} \to \mathbb{G}$ and $H' : \mathbb{G} \times \mathcal{X} \to \mathcal{Y}$ are random oracles, and prove that this construction is secure in the universal composability (UC) framework under a variant of the one-more gap computational Diffie-Hellman (OM-CDH) assumption [JKK14, JKKX16]. We generalize this protocol to the group action setting formalized in Section 2.4, add setup phases for the sender and receiver, and prove that our protocol achieves malicious security in the standard simulation-based security framework under the natural generalization of the OM-CDH assumption. In particular, we prove that the sender (resp. receiver) setup phase can securely be re-used with multiple receiving (resp. sending) parties[2]. Our protocol $\Pi_{\mathsf{cga\text{-}voprf}}$ is detailed in Figure 7 and security is proven in Theorem 2 under the assumption in Definition 5.

Definition 5 (OM-CDH-GA Assumption)**.** Let $(\mathcal{U}_\kappa)_{\kappa\in\mathbb{N}}$ be a family of principal homogeneous spaces and let $m, n \in \mathbb{N}$. We say that the (m,n)-one more computational Diffie-Hellman group action ((m,n)-OM-CDH-GA) assumption holds with respect to $(\mathcal{U}_\kappa)_{\kappa\in\mathbb{N}}$, if for any $\kappa \in \mathbb{N}$, any $(\mathbb{H}, U, \star) \leftarrow \mathcal{U}_\kappa$, and any PPT adversary $\mathcal{A}$, the following probability is negligible in κ:

$$\Pr\left[\{(a_{j_i}, K \star a_{j_i}) : i \in [m+1]\} \leftarrow \mathcal{A}^{K\star(\cdot),\mathsf{DDH}(\cdot)}(1^\kappa, E, K \star E, a_1, \ldots, a_n)\right],$$

where the probability is taken over $K \xleftarrow{\$} \mathbb{H}$ and $a_1, \ldots, a_n \xleftarrow{\$} U$, $E \in U$ is an arbitrary point and the $j_i \in [n]$ are distinct for $i = 1, \ldots, m+1$. Moreover, m is the number of

[2]Note that this re-usability is not implied by UC security

Figure 7 Cryptographic Group Action-based VOPRF protocol $\Pi_{\mathsf{cga\text{-}voprf}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on an input space $\mathcal{X}$, an output space $\mathcal{Y}$ and a principal homogeneous space $(\mathbb{H}, U, \star) \leftarrow \mathcal{U}_\kappa$ for security parameter $\kappa \in \mathbb{N}$. $H : \mathcal{X} \to U$ and $H' : \{0,1\}^* \to \mathcal{Y}$ are random oracles. $\mathcal{L}^{\mathcal{S}_j}$ is a list of $\mathcal{S}_j$'s setup phases, and $\mathcal{L}^{\mathcal{R}_i}$ is a list of $\mathcal{R}_i$'s setup phases. Initialize $\mathcal{L}^{\mathcal{S}_j}, \mathcal{L}^{\mathcal{R}_i} := \emptyset$.

Subprotocol $\mathsf{Setup}_{\mathcal{R}}$ with $\mathcal{R}_i$-input (ρ, X):

- If $\exists X', (r_k^\rho)_{k=1}^{|X'|}$: $(\rho, X', (r_k^\rho)_{k=1}^{|X'|}) \in \mathcal{L}^{\mathcal{R}_i}$ then do nothing.
- Else, $\mathcal{R}_i$ samples $r_1^\rho, \ldots, r_{|X|}^\rho \xleftarrow{\$} \mathbb{H}$, computes $a_k^\rho := r_k^\rho \star H(x_k)$, updates $\mathcal{L}^{\mathcal{R}_i} \leftarrow \mathcal{L}^{\mathcal{R}_i} \cup \{(\rho, X, (r_k^\rho)_{k=1}^{|X|})\}$ and sends $(\mathtt{post}, \mathsf{sid}, \mathsf{id}_\rho, (\rho, (a_k^\rho)_{k=1}^{|X|}))$ to $\mathcal{F}_{\mathsf{ABB}}$.

Subprotocol $\mathsf{Setup}_{\mathcal{S}}$ with $\mathcal{S}_j$-input (σ, m):

- If $\exists (\tilde{K}_k^\sigma)_{k=1}^m$: $(\sigma, m, (\tilde{K}_k^\sigma)_{k=1}^m) \in \mathcal{L}^{\mathcal{S}_j}$ put $K_k^\sigma := \tilde{K}_k^\sigma$ for each $k \in [m]$.
- Else, $\mathcal{S}_j$ samples $K_1^\sigma, \ldots, K_m^\sigma \xleftarrow{\$} \mathbb{H}$ and updates $\mathcal{L}^{\mathcal{S}_j} \leftarrow \mathcal{L}^{\mathcal{S}_j} \cup \{(\sigma, m, (K_k^\sigma)_{k=1}^m)\}$.
- Return $(\mathcal{O}^{F_{K_k^\sigma}})_{k=1}^m$ to $\mathcal{S}_j$ with $F_{K_k^\sigma}(x) := H'(\mathsf{id}_{\mathcal{S}_j}, \sigma, m, k, K_k^\sigma \star H(x), x)$.

Subprotocol $\mathsf{Evaluate}$ with $\mathcal{R}_i$-input $(\mathsf{id}_{\mathcal{S}_j}, \rho)$ and $\mathcal{S}_j$-input $(\mathsf{id}_{\mathcal{R}_i}, \sigma)$:

- $\mathcal{R}_i$ sends ρ to $\mathcal{S}_j$.
- $\mathcal{R}_i$ retrieves $(X, (r_k^\rho)_{k=1}^{|X|})$ for which $(\rho, X, (r_k^\rho)_{k=1}^{|X|}) \in \mathcal{L}^{\mathcal{R}_i}$ or aborts if this fails.
- $\mathcal{S}_j$ sends $(\mathtt{read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ABB}}$ and searches $\mathcal{M}$ for the first message $(\mathsf{id}_{\mathcal{R}_i}, \mathsf{sid}, \mathsf{id}_\rho, (\rho, (a_k^\rho)_{k=1}^{|X|}))$ with $a_k^\rho \in U$, or aborts if it does not exist.
- $\mathcal{S}_j$ sends σ to $\mathcal{R}_i$.
- $\mathcal{S}_j$ calls $\mathsf{Setup}_{\mathcal{S}}$ with input $(\sigma, |X|)$ and obtains $(K_k^\sigma)_{k=1}^{|X|}$.
- $\mathcal{S}_j$ computes $b_k^{\sigma,\rho} := K_k^\sigma \star a_k^\rho$ for each $k \in [|X|]$, and sends $(b_k^{\sigma,\rho})_{k=1}^{|X|}$ to $\mathcal{R}_i$.
- $\mathcal{R}_i$ aborts if $b_k^{\sigma,\rho} \notin U$ for any $k \in [m]$.
- $\mathcal{R}_i$ outputs $(F_{K_k^\sigma}(x_k))_{k=1}^{|X|}$ with $F_{K_k^\sigma}(x_k) := H'(\mathsf{id}_{\mathcal{S}_j}, \sigma, |X|, k, (r_k^\rho)^{-1} \star b_k^{\sigma,\rho}, x_k)$.
- $\mathcal{S}_j$ outputs $(\mathcal{O}^{F_{K_k^\sigma}})_{k=1}^{|X|}$ with $F_{K_k^\sigma}(x) := H'(\mathsf{id}_{\mathcal{S}_j}, \sigma, |X|, k, K_k^\sigma \star H(x), x)$.

queries $\mathcal{A}$ can make to the $K \star (\cdot)$ oracle, i.e., the oracle that returns $K \star b$ on input $b \in U$, and $\mathsf{DDH}(\cdot)$ is a decisional Diffie-Hellman oracle, i.e., which on input $(u, K \star u, b, c)$ returns 1 if $c = K \star b$ and 0 otherwise.

The protocol $\Pi_{\mathsf{cga\text{-}voprf}}$ actually realizes the functionality $\mathcal{F}^*_{\mathsf{voprf}}$, detailed in Figure 15, against malicious adversaries. In this functionality, a corrupted receiver has the possibility to adaptively query inputs to F_k for $k \in [|X|]$, but can only pose a single query for each k. In the semi-honest setting, we only need to be able to answer queries of the form $F_k(X_k)$, where X_k is the k-th element of the receivers input set X; so this implies that $\Pi_{\mathsf{cga\text{-}voprf}}$ realizes the functionality $\mathcal{F}_{\mathsf{voprf}}$, detailed in Figure 4, against semi-honest adversaries.

Theorem 2 (Adapted from [JKKX16, Thm. 1])**.** *The protocol $\Pi_{\mathsf{cga\text{-}voprf}}$ securely realizes the functionality $\mathcal{F}^*_{\mathsf{voprf}}$ with $\mathsf{mr} = \mathsf{yes}$ against malicious adversaries in the random oracle, $\mathcal{F}_{\mathsf{BB}}$-hybrid model under the $(N_{\mathcal{R}}, Q)$-OM-CDH-GA assumption, where $N_{\mathcal{R}}$ is the number of receiving parties, Q is an upper bound on the total number of H and H' queries the receiving parties $(\mathcal{R}_i)_{i \in [N_{\mathcal{R}}]}$ make and H, H' are modeled as random oracles.*

Proof. Since a sender $\mathcal{S}$ can re-use their setup phase (σ, m) with multiple corrupted receivers $\mathcal{R}_i$, we construct a simulator out of subroutines each interacting with a corrupted receiver $\mathcal{R}_i$. Moreover, since executions with different honest senders are independent, it is sufficient

to simulate an interaction with a single honest sender $\mathcal{S}$. The simulator receives the honest sender's relevant setup phase identifiers σ_i as the sender sends their `evaluate` message to the ideal functionality $\mathcal{F}^*_{\mathsf{voprf}}$ and can subsequently sample corresponding keys $K_j^{\sigma_i} \xleftarrow{\$} \mathbb{H}$. Using these keys, the simulator can compute the elements $b_k^{\sigma_i,\rho_i} := K_j^{\sigma_i} \star a_j^{\rho_i}$ to send to $\mathcal{R}_i$, where the simulator retrieves the elements $a_j^{\rho_i}$ from the simulated bulletin board. The simulator can moreover answer queries to H' of the form $(\mathsf{id}_{\mathcal{S}}, \sigma_i, m_i, j, K_j^{\sigma_i} \star H(x), x)$ by querying $F_j^{\sigma_i}(x)$ from $\mathcal{F}^*_{\mathsf{voprf}}$. The latter fails if the simulator queries a particular function $F_j^{\sigma_i}$ once more than the number of executions the setup phase σ_i was used by the honest sender $\mathcal{S}$. We denote this failure event by FAIL and show in Theorem 7 that it occurs with at most negligible probability if the $(N_{\mathcal{R}}, Q)$-OM-CDH-GA assumption holds, where $N_{\mathcal{R}}$ is the total number of receiving parties and Q is an upper bound on the number of H and H' the receiving parties make in total. If FAIL does not occur, the simulated execution is perfectly indistinguishable from the real protocol execution.

Again the receiver $\mathcal{R}$ can re-use their setup phase (σ, m) with multiple corrupted senders $\mathcal{S}_i$ so we construct a simulator out of subroutines each interacting with a corrupted sender $\mathcal{S}_i$. Moreover, since interactions with different honest receivers are independent, it is sufficient to simulate an interaction with a single honest receiver $\mathcal{R}$. The simulator obtains the receiver's setup phase identifiers (ρ, m) from the ideal functionality $\mathcal{F}^*_{\mathsf{voprf}}$, samples $\alpha_j^{\rho} \xleftarrow{\$} \mathbb{H}$ and adds the elements $a_j^{\rho} := \alpha_j^{\rho} \star E$ to the simulated bulletin board, where $E \in U$ is an arbitrary base point picked by the simulator. Upon receiving the elements $b_j^{\rho_i,\bar{\sigma}_i}$ from $\mathcal{S}_i$, the simulator can compute identifiers for the sender's keys as $\bar{\sigma}_j^i := (\alpha_j^{\rho_i})^{-1} \star b_j^{\rho_i,\bar{\sigma}_i}$ and send these to $\mathcal{F}^*_{\mathsf{voprf}}$ to obtain the functions $F_j^{\bar{\sigma}_i}$. The simulator answers queries x to H by sampling $h_x \xleftarrow{\$} \mathbb{H}$ and returning $h_x \star E$. It can then use h_x to compute an identifier $(\sigma, (h_x)^{-1} \star u)$ for the key corresponding to a query $(\mathsf{id}_{\mathcal{S}_i}, \sigma, m, j, u, x)$ to H', and answer these queries by posing `setup` queries to $\mathcal{F}^*_{\mathsf{voprf}}$. The resulting distributions are perfectly indistinguishable. □

4 Vector Oblivious Programmable PRF (VOPPRF)

In this section we present our ideal functionality for a vector oblivious programmable pseudorandom function (VOPPRF) that allows options for sender and receiver setup phases, and present a protocol realizing this functionality using a VOPRF and an OKVS.

4.1 Ideal Functionality

Our ideal VOPPRF functionality $\mathcal{F}_{\mathsf{vopprf}}$ in Figure 8 naturally extends the functionality $\mathcal{F}_{\mathsf{voprf}}$ from Section 3 with the possibility for the sender to "program" points of their choosing in the random function. Compared to the ideal OPPRF functionality from Rindal and Schoppmann [RS21], our functionality uses a different programmed random function for each entry of the receiver's input vector, and we again add possibilities for sender and receiver setup phases which they can re-use during subsequent executions.

4.2 VOPPRF from VOPRF + OKVS

We present a VOPPRF protocol from a VOPRF and an OKVS as a natural generalization of the protocol from Rindal and Schoppmann [RS21], who construct an OPPRF from an OPRF and their XoPaXoS solver, by replacing XoPaXoS with a *linear* and *doubly oblivious* OKVS. This generalization was also mentioned by [GPR+21, RR22], but to the best of our knowledge a formal proof has not been written down anywhere. Note that the currently most efficient OKVS constructions [RR22, BPSY23] are both linear as well as doubly oblivious. In case the underlying VOPRF protocol supports multiple receivers ($\mathsf{mr} = \mathsf{yes}$), we add the option for the sender to post their OKVS encoding on a public

Figure 8 Ideal VOPPRF functionality $\mathcal{F}_{\mathsf{vopprf}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on an input space $\mathcal{X}$ and a finite ring R as output space. Let PPRF_L be an internal random oracle for programmed random functions $\mathcal{F}_L := \{f : \mathcal{X} \to R \mid \forall (x,y) \in L : f(x) = y\}$. $\mathcal{L}^{\mathcal{R}_i}$ is a list of $\mathcal{R}_i$'s setup phases and $\mathcal{L}^{\mathcal{S}_j}$ is a list of $\mathcal{S}_j$'s setup phases. Initialize $\mathcal{L}^{\mathcal{S}_j}, \mathcal{L}^{\mathcal{R}_i} := \emptyset$. $M : \mathbb{N} \to \mathbb{N}$ is an upper bound.

Functionality:

- On query $(\mathtt{setup}, \rho, X)$ from $\mathcal{R}_i$, if $\exists X : (\rho, X) \in \mathcal{L}^{\mathcal{R}_i}$ do nothing. Else, add (ρ, X) to $\mathcal{L}^{\mathcal{R}_i}$, and send $(\mathsf{id}_{\mathcal{R}}, \rho, |X|)$ to $\mathcal{A}$.
- If $\mathsf{mr} = \mathsf{yes}$ then on query $(\mathtt{setup}, \sigma, (L_k)_{k=1}^m)$ from $\mathcal{S}_j$:
  - If $\exists m', (L'_k)_{k=1}^{m'} : (\sigma, (L'_k)_{k=1}^{m'}) \in \mathcal{L}^{\mathcal{S}_j}$ then do nothing.
  - Else:
    - Add $(\sigma, (L_k)_{k=1}^m)$ to $\mathcal{L}^{\mathcal{S}_j}$.
    - Send $(\mathsf{id}_{\mathcal{S}}, \sigma, m, \sum_k |L_k|)$ to $\mathcal{A}$.
    - Return $(\mathcal{O}^{\mathsf{PPRF}_{L_k}(\mathsf{id}_{\mathcal{S}_j}, \sigma, k)})_{k=1}^m$ to $\mathcal{S}_j$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \mathtt{evaluate}, \rho)$ from $\mathcal{R}_i$, if $\exists X : (\rho, X) \in \mathcal{L}^{\mathcal{R}_i}$ send $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho, |X|)$ to $\mathcal{S}_j$. Else, send abort to $\mathcal{R}_i$ and $\mathcal{S}_j$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \mathtt{evaluate}, \sigma, (L_k)_{k=1}^{|X|})$ from $\mathcal{S}_j$:
  - Set $N := \sum_{k=1}^m |L_k|$ and abort if $N > M(m)$.
  - If $\mathsf{mr} = \mathsf{no}$, sample $F \xleftarrow{\$} \prod_{k=1}^m \mathcal{F}_{L_k}$.
  - If $\mathsf{mr} = \mathsf{yes}$, then set $F := (\mathsf{PPRF}_{L_k}(\mathsf{id}_{\mathcal{S}_j}, \sigma, k))_{k=1}^{|X|}$; if $(\sigma, (L_k)_{k=1}^{|X|}) \notin \mathcal{L}^{\mathcal{S}}$ then send abort to $\mathcal{R}_i$ and $\mathcal{S}_j$.
  - Output $(\mathcal{O}^{F_k})_{k=1}^{|X|}$ to $\mathcal{S}_j$ and σ, N, $(F_k(X_k))_{k=1}^{|X|}$ to $\mathcal{R}_i$.

bulletin board and effectively re-use the same programmed function with multiple receivers. Our protocol Π_{vopprf} is detailed in Figure 9 and security is proven in Theorem 3.

Theorem 3. *Let* $\mathsf{OKVS} = (\mathsf{Encode}, \mathsf{Decode})$ *be a linear doubly oblivious OKVS. Then the protocol* Π_{vopprf} *securely realizes the functionality* $\mathcal{F}_{\mathsf{vopprf}}$ *against semi-honest senders and malicious receivers in the* $\mathcal{F}_{\mathsf{BB}}$*-,* $\mathcal{F}_{\mathsf{voprf}}$*-hybrid model, when the flag* $\mathsf{mr} \in \{\mathsf{yes}, \mathsf{no}\}$ *has the same value for* $\mathcal{F}_{\mathsf{vopprf}}$*,* Π_{vopprf} *and* $\mathcal{F}_{\mathsf{voprf}}$*.*

Proof. First consider the setting of semi-honest corrupted senders $\mathcal{S}_i$. The simulator receives as input σ_i, $(L_k^i)_{k=1}^{m_i}$, ρ_i, $|X^i|$, $(\mathcal{O}^{F_k^i})_{k=1}^{|X^i|}$ and can handle the sender's setup and evaluate queries as the ideal functionality $\mathcal{F}_{\mathsf{voprf}}$ does to obtain $(\mathcal{O}^{F_k^{i'}})_{k=1}^{m_i}$, and compute θ_i, $\boldsymbol{P}^i$ as the sender does. On programmed points $(y, z) \in L_k^i$, the ideal functionality output satisfies $F_k^i(y) = z = F_k^{i'}(y) + \langle \boldsymbol{P}^i, d_{|\boldsymbol{P}^i|}(y; \theta) \rangle$. On unprogrammed points we can simulate the OPRF oracles $(\mathcal{O}^{F_k^{i'}})_{k=1}^{m_i}$ as $F_k^{i'}(x) := F_k^i(x) - \langle \boldsymbol{P}^i, d_{|\boldsymbol{P}^i|}(x; \theta) \rangle$, which is perfectly indistinguishable from the ideal $\mathcal{F}_{\mathsf{voprf}}$ output since $F_k^i(x)$ is an independent uniformly random value on unprogrammed points.

Now consider the setting of maliciously corrupted receivers $\mathcal{R}_i$. The simulator can extract the receiver $\mathcal{R}_i$'s inputs ρ_i, X^i from their `setup` and `evaluate` queries to $\mathcal{F}_{\mathsf{voprf}}$ and can forward these to $\mathcal{F}_{\mathsf{vopprf}}$. Moreover, the simulator receives the sender's setup phase identifiers σ, N, m as the honest sender sends their `setup` queries to $\mathcal{F}_{\mathsf{voprf}}$ and can simulate the sender's setup phases by sampling $\theta \xleftarrow{\$} \{0,1\}^\kappa$, $\boldsymbol{P} \xleftarrow{\$} R^N$ and posting it on the bulletin board held by $\mathcal{F}_{\mathsf{BB}}$. Note that this is statistically indistinguishable from the real-protocol

Figure 9 VOPRF-based VOPPRF protocol Π_{voppprf}

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on an input space $\mathcal{X}$ and a finite ring R as output space. $(\mathsf{Encode}, \mathsf{Decode})$ is a linear OKVS with key space $\mathcal{X}$, value space R and associated decoding function family $\{d_{q(n)} \mid n \in \mathbb{N}\}$. $\mathcal{L}^{\mathcal{S}}$ is a list of $\mathcal{S}$'s setup phases. Initialize $\mathcal{L}^{\mathcal{S}} := \emptyset$. $M : \mathbb{N} \to \mathbb{N}$ and $q : \mathbb{N} \to \mathbb{N}$ are functions.

Subprotocol $\mathsf{Setup}_{\mathcal{R}}$ with $\mathcal{R}$-input (ρ, X):

- $\mathcal{R}_j$ sends $(\mathtt{setup}, \rho, X)$ to $\mathcal{F}_{\mathsf{voprf}}$.

If $\mathsf{mr} = \mathsf{yes}$: Subprotocol $\mathsf{Setup}_{\mathcal{S}}$ with $\mathcal{S}_j$-input $(\sigma, (L_k)_{k=1}^m)$:

- If $\exists m', \boldsymbol{P}', (L'_k)_{k=1}^{m'}, \theta'$: $(\sigma, (L'_k)_{k=1}^{m'}, \boldsymbol{P}', \theta') \in \mathcal{L}^{\mathcal{S}_j}$ then $\mathcal{S}_j$ does nothing.
- Else:
  - $\mathcal{S}_j$ sends $(\mathtt{setup}, \sigma, m)$ to $\mathcal{F}_{\mathsf{voprf}}$ and receives $(\mathcal{O}^{F'_{\sigma,k}})_{k=1}^m$.
  - $\mathcal{S}_j$ samples $\theta \xleftarrow{\$} \{0,1\}^\kappa$ and computes $\boldsymbol{P} \leftarrow \mathsf{Encode}\Big(\{(y, z - F'_{\sigma,k}(y)) \mid (y,z) \in L_k, k \in [m]\}; \theta\Big)$.
  - $\mathcal{S}_j$ updates $\mathcal{L}^{\mathcal{S}_j} \leftarrow \mathcal{L}^{\mathcal{S}_j} \cup \{(\sigma, (L_k)_{k=1}^m, \boldsymbol{P}, \theta)\}$.
  - $\mathcal{S}_j$ sends $(\mathtt{post}, \mathsf{sid}, \mathsf{id}_\sigma, (\sigma, m, \boldsymbol{P}, \theta))$ to $\mathcal{F}_{\mathsf{BB}}$.
  - Return $(\mathcal{O}^{F_{\sigma,k}})_{k=1}^m$ to $\mathcal{S}_j$ with $F_{\sigma,k}(x) := F'_{\sigma,k}(x) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(x;\theta)\rangle$.

Subprotocol $\mathsf{Evaluate}$ with $\mathcal{R}_i$-input $(\mathsf{id}_{\mathcal{S}_j}, \rho)$ and $\mathcal{S}_j$-input $(\mathsf{id}_{\mathcal{R}_i}, \sigma, (L_k)_{k=1}^m)$:

- $\mathcal{R}_i$ sends $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \mathtt{evaluate}, \rho)$ to $\mathcal{F}_{\mathsf{voprf}}$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ abort if they receive abort from $\mathcal{F}_{\mathsf{voprf}}$.
- Otherwise, $\mathcal{S}_j$ gets $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho, |X|)$, aborts if $|X| \neq m$, and sends $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \mathtt{evaluate}, \sigma)$ to $\mathcal{F}_{\mathsf{voprf}}$.
- $\mathcal{S}_j$ receives $(\mathcal{O}^{F'_k})_{k=1}^{|X|}$ and $\mathcal{R}$ receives $(F'_k(X_k))_{k=1}^{|X|}$ from $\mathcal{F}_{\mathsf{voprf}}$.
- If $\mathsf{mr} = \mathsf{yes}$:
  - $\mathcal{S}_j$ sends σ to $\mathcal{R}_i$.
  - $\mathcal{S}_j$ retrieves $(\boldsymbol{P}, \theta)$ for which $(\sigma, |X|, (L_k)_{k=1}^m, \boldsymbol{P}, \theta) \in \mathcal{L}^{\mathcal{S}_j}$, or aborts.
  - $\mathcal{R}_i$ sends $(\mathtt{read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BB}}$ and searches $\mathcal{M}$ for the first message $(\mathsf{id}_{\mathcal{S}_j}, \mathsf{sid}, \mathsf{id}_\sigma, (\sigma, |X|, \boldsymbol{P}, \theta))$ with $\boldsymbol{P} \in R^*$ and $\theta \in \{0,1\}^\kappa$, or aborts.
- If $\mathsf{mr} = \mathsf{no}$:
  - $\mathcal{S}_j$ samples $\theta \xleftarrow{\$} \{0,1\}^\kappa$.
  - $\mathcal{S}_j$ computes $\boldsymbol{P} \leftarrow \mathsf{Encode}\left(\{(y, z - F'_k(y)) \mid (y,z) \in L_k, k \in [m]\}; \theta\right)$.
  - $\mathcal{S}_j$ sends $(\boldsymbol{P}, \theta)$ to $\mathcal{R}_i$.
- $\mathcal{R}_i$ aborts if $\theta \notin \{0,1\}^\kappa$, $|\boldsymbol{P}| \notin \mathrm{im}(q)$ or $|\boldsymbol{P}| > q(M(m))$.
- $\mathcal{R}_i$ outputs $|\boldsymbol{P}|, (F_k(X_k))_{k=1}^{|X|}$ and $\mathcal{S}_j$ outputs $(\mathcal{O}^{F_k})_{k=1}^{|X|}$, where $F_k(x) := F'_k(x) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(x;\theta)\rangle$.

OKVS encoding since we require OKVS to be doubly oblivious and the sender encodes distinct keys with independent uniformly random values in the OKVS. The simulator receives $\sigma_i, N_i, (F_k^i(X_k^i))_{i=1}^{|X^i|}$ as the honest sender sends their $\mathtt{evaluate}$ query to $\mathcal{F}_{\mathsf{voppprf}}$, and can retrieve the corresponding θ_i, $\boldsymbol{P}^i$. Hence we can answer $\mathcal{R}_i$'s $\mathtt{evaluate}$ query to $\mathcal{F}_{\mathsf{voprf}}$ by putting $F_k^{i'}(X_k^i) := F_k^i(X_k^i) - \langle \boldsymbol{P}^i, d_{|\boldsymbol{P}^i|}(X_k^i;\theta)\rangle$. These values have statistical distance $\leq 2^{-\lambda}$ with the real protocol values coming from the ideal $\mathcal{F}_{\mathsf{voprf}}$ functionality, which can be seen from the following two cases:

(a) If $X_k^i = y_{k,j}^i$ for some programmed point $(y_{k,j}^i, z_{k,j}^i) \in L_k^i$, this implies that

$F_k^{i'}(X_k^i) := z_{k,j}^i - \langle \boldsymbol{P}^i, d_{|\boldsymbol{P}^i|}(y_{k,j}^i;\theta)\rangle$.The values $\langle \boldsymbol{P}^i, d_{|\boldsymbol{P}^i|}(y_{k,j}^i;\theta)\rangle$ are independent uniformly random for $k \in [m_i]$ and $j \in [|L_k^i|]$ since $\boldsymbol{P}^i$ is uniformly random and independent from L_k^i and $d_{|\boldsymbol{P}^i|}(y_{k,j}^i;\theta)$ are linearly independent vectors with probability $\geq 1 - 2^{-\lambda}$ over the choice of $\theta \xleftarrow{\$} \{0,1\}^\kappa$ by the correctness of the OKVS.

(b) On unprogrammed points the values $F_k^i(X_k^i)$ are uniformly random and independent from $\boldsymbol{P}^i$, resulting in independent uniform values $F_k^{i'}(X_k^i)$. □

4.3 Malicious Adversaries

Recall that the protocol Π_{vopprf} presented in Section 4.2 is only secure against malicious receivers, but not against malicious senders. The main obstacle is that a malicious sender, after receiving a random function F', is allowed to provide a vector $\boldsymbol{P}$ of bounded length and associated encoding randomness $\theta \in \{0,1\}^\kappa$ such that the programmed function is defined as $F(x) := F'(x) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(x;\theta)\rangle$. Here $\{d_N\}$ is a function family parameter to the functionality, corresponding to a chosen linear OKVS scheme, i.e., it maps x to a vector of length $N = |\boldsymbol{P}|$ with parameter θ (see Sec. 2.2). This adversarial behavior does not pose any issues with respect to the privacy of the receiver's input set X, but hurts the correctness of the protocol. The main issue here is that the simulator has no way to extract the points (y, z), on which F is meant to be programmed, from the vector $\boldsymbol{P}$. What a malicious sender can achieve in this way is highly dependent on the larger context within which the VOPPRF protocol is used as a subprototocol. We discuss this in the context of our Core-PSI protocol in Section 5.3.

Note that if we want to instantiate the protocol Π_{vopprf} using the group action OPRF from Section 3.3 as the underlying VOPRF, we need to replace the calls to the ideal $\mathcal{F}_{\mathsf{voprf}}$ functionality by calls to the ideal $\mathcal{F}^*_{\mathsf{voprf}}$ functionality. If the receiver is malicious, this means they have the ability to adaptively change their input vector, but can still only receive a single evaluation from each programmed function. We will refer to this slightly different ideal functionality as $\mathcal{F}^*_{\mathsf{vopprf}}$.

5 Core-PSI from VOPPRF

In this section we introduce the first variant of our new Core-PSI functionality and provide a protocol that realizes the functionality from a VOPPRF in the semi-honest setting. Core-PSI can be seen as a generalization of Circuit-PSI, where the final comparison step is omitted and the possibility for sender and receiver setup phases is added. In section 6 we present another variant of Core-PSI and a protocol realizing it from a shared-output PRF (SOPRF). In Section 8 we briefly discuss how one can obtain Circuit-PSI and other functionalities from Core-PSI through post-processing.

5.1 Ideal Functionality

Our ideal Core-PSI functionality $\mathcal{F}_{\mathsf{Core\text{-}PSI}}$ is given in Figure 10. In essence, the functionality is similar to the ideal Circuit-PSI functionality from [RS21], with the main difference being that the Core-PSI functionality returns arithmetic shares of matching values in case there is a match and of random values in case there is no match, whereas the Circuit-PSI functionality returns shares of zero in case there is no match. The intuition behind this difference is that we replaced the secure comparison step that is usually done at the end of Circuit-PSI protocols by a (cheaper) rerandomization step, and formalized the ideal functionality that the adapted protocol realizes. We chose this solution as the secure comparison step is a general post-processing solution for Circuit-PSI applications, but not the only solution, nor the best in some applications. This allowed us to study the core of PSI and potential amortizations, while allowing freedom in post-processing for different

Figure 10 Ideal Core-PSI functionality $\mathcal{F}_{\mathsf{Core\text{-}PSI}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on a key alphabet $\mathcal{X}$, a finite ring S as value alphabet and a finite ring R. A function $\mathsf{Reorder} : \mathcal{X}^n \to (\pi : [n] \to [m])$ which maps a set $X \subset \mathcal{X}$ to an injective function. $\mathcal{L}^{\mathcal{R}_i}$ is a list of receiver setup phases, $\mathcal{L}^{\mathcal{S}_j}$ is a list of sender setup phases, and $\mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}$ is a list of (sender setup phase, receiver id) tuples. Initialize $\mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}, \mathcal{L}^{\mathcal{S}_j}, \mathcal{L}^{\mathcal{R}_i} := \emptyset$.

Functionality:

- On query $(\texttt{setup}, \rho, m, X)$ from $\mathcal{R}_i$:
  - If $\exists \pi : (\rho, m, \pi) \in \mathcal{L}^{\mathcal{R}_i}$ then do nothing; else:
  - Set $\pi \leftarrow \mathsf{Reorder}(X)$.
  - Add (ρ, m, π) to $\mathcal{L}^{\mathcal{R}_i}$ and send $(\mathsf{id}_{\mathcal{R}}, \rho, m)$ to $\mathcal{A}$.
- If $\mathsf{mr} = \mathsf{yes}$, then on query $(\texttt{setup}, \sigma, m, Y)$ from $\mathcal{S}_j$:
  - If $\exists m', Y' : (\sigma, m', Y') \in \mathcal{L}^{\mathcal{S}_j}$ then do nothing; Else:
  - Add (σ, m, Y) to $\mathcal{L}^{\mathcal{S}_j}$ and send $(\mathsf{id}_{\mathcal{S}_j}, \sigma, m, |Y|)$ to $\mathcal{A}$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \texttt{evaluate}, \rho)$ from $\mathcal{R}_i$:
  - If $\exists \pi' : (\rho, m, \pi') \in \mathcal{L}^{\mathcal{R}_i}$ then let $\pi := \pi'$. Else, send abort to $\mathcal{R}_i$ and $\mathcal{S}_j$.
  - Send $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho, m)$ to $\mathcal{S}_j$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \texttt{evaluate}, \sigma, m, Y)$ from $\mathcal{S}_j$:
  - If $\mathsf{mr} = \mathsf{yes}$, abort if $(\sigma, m, Y) \notin \mathcal{L}^{\mathcal{S}_j}$ or $(\mathsf{id}_{\mathcal{R}_i}, \sigma) \in \mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}$.
    Else, add $(\mathsf{id}_{\mathcal{R}_i}, \sigma)$ to $\mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}$.
  - Sample $r_k, s_k \in R \times S$ uniformly such that:
    - $r_k + s_k = (0, \tilde{y})$ if there exists $x_\nu \in X$, $y \in Y$ with $x_\nu = y$, $\pi(\nu) = k$,
    - $r_k, s_k \xleftarrow{\$} R \times S$ otherwise
  - Output σ, $|Y|$, $(r_k)_{k=1}^m$ to $\mathcal{R}_i$ and $(s_k)_{k=1}^m$ to $\mathcal{S}_j$.

kinds of applications. We discuss different post-processing options in Section 8. To enable amortizations, the Core-PSI functionality additionally allows sender and receiver setup phases similar to the VOPPRF functionality.

5.2 Protocol

Our Core-PSI protocol follows the same blueprint as the Circuit-PSI protocol from Pinkas et al. [PSTY19] and several follow-up works [RS21, CGS22, CDG+21, RR22, BPSY23]. The main difference is that we replace the final secure comparison step by a (cheaper) rerandomization step and add options for receiver and sender setup phases. The protocol $\Pi_{\mathsf{Core\text{-}PSI}}$ is detailed in Figure 11 and proven secure in Theorem 4.

Remark 2. One important caveat in terms of reusability is that, when $\mathsf{mr} = \mathsf{yes}$, a sender should *neither* use the same setup phase (σ, Y) with multiple corrupted receivers, *nor* use it multiple times with the same receiver. Namely, if the sender maps two different items $y_1, y_2 \in Y$ to the same bucket k, then the OPPRF outputs $F_k(y_1) = (0, \tilde{y}_1) - s'_k$ and $F_k(y_2) = (0, \tilde{y}_2) - s'_k$ share the same first component. Hence if a receiver with $y_1, y_2 \in X$ maps y_1 in one execution and y_2 in another execution to bucket k of their Cuckoo hash table, they learn from the collision in the first part of the OPPRF outputs that both y_1 and y_2 are in the sender's set Y with overwhelming probability.

Figure 11 VOPPRF-based Core-PSI protocol $\Pi_{\mathsf{Core\text{-}PSI}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on a key alphabet $\mathcal{X}$, a finite ring S as value alphabet and a finite ring R. $(h_v^m)_{m\in\mathbb{N}}$ is a family of public hash functions $h_v^m : \mathcal{X} \to [m]$ for each $v \in [t]$. $\mathcal{L}^{\mathcal{R}_i}$ is a list of receiver setup phases, $\mathcal{L}^{\mathcal{S}_j}$ is a list of sender setup phases, and $\mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}$ is a list of (sender setup phase, receiver id) tuples. Initialize $\mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}, \mathcal{L}^{\mathcal{S}_j}, \mathcal{L}^{\mathcal{R}_i} := \emptyset$.

Subprotocol $\mathsf{Setup}_{\mathcal{R}}$ **with** $\mathcal{R}_i$**-input** (ρ, m, X)**:**

- If $\exists T'_x : (\rho, m, X, T'_x) \in \mathcal{L}^{\mathcal{R}_i}$, then put $T_x := T'_x$.
- Else, $\mathcal{R}_i$ constructs a Cuckoo hash table T_x of X of size m using $h_1, \ldots, h_t$. That is, for each $x \in X$ there exists a unique $v \in [t]$ such that $(x, h_v(x)) = T_x[h_v(x)]$ and each bin contains at most one tuple, and pads the empty bins $k \in [m]$ with dummy values (x'_k, k), where $x'_k \xleftarrow{\$} \mathcal{X}$.
  $\mathcal{R}_i$ updates $\mathcal{L}^{\mathcal{R}_i} \leftarrow \mathcal{L}^{\mathcal{R}_i} \cup \{(\rho, m, X, T_x)\}$.
- $\mathcal{R}_i$ queries $(\mathtt{setup}, \rho, T_x)$ to $\mathcal{F}_{\mathsf{voprf}}$.

If $\mathsf{mr} = \mathsf{yes}$**: Subprotocol** $\mathsf{Setup}_{\mathcal{S}}$ **with** $\mathcal{S}_j$**-input** (σ, m, Y)

- If $\exists m', Y', (\tilde{s}_k)_{k=1}^{m'}, (L'_k)_{k=1}^{m'}$: $(\sigma, m', Y', (\tilde{s}_k)_{k=1}^{m'}, (L'_k)_{k=1}^{m'}) \in \mathcal{L}^{\mathcal{S}_j}$, $\mathcal{S}_j$ does nothing.
- Else:
  - $\mathcal{S}_j$ samples $(s'_k)_{k=1}^m \xleftarrow{\$} (R \times S)^m$.
  - $\mathcal{S}_j$ defines $L_k := \{((y,k), (0,\tilde{y}) - s'_k) : y \in Y, v \in [t],\ h_v(y) = k\}$.
  - $\mathcal{S}_j$ updates $\mathcal{L}^{\mathcal{S}_j} \leftarrow \mathcal{L}^{\mathcal{S}_j} \cup \{(\sigma, m, Y, (s'_k)_{k=1}^m, (L_k)_{k=1}^m)\}$.
  - $\mathcal{S}_j$ queries $(\mathtt{setup}, \sigma, (L_k)_{k=1}^m)$ to $\mathcal{F}_{\mathsf{voprf}}$ and receives $(\mathcal{O}^{F_k^\sigma})_{k=1}^m$.

Subprotocol $\mathsf{Evaluate}$ **with** $\mathcal{R}_i$**-input** $(\mathsf{id}_{\mathcal{S}_j}, \rho)$ **and** $\mathcal{S}_j$**-input** $(\mathsf{id}_{\mathcal{R}_i}, \sigma, Y)$**:**

- $\mathcal{R}_i$ sends $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \mathtt{evaluate}, \rho)$ to $\mathcal{F}_{\mathsf{voprf}}$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ abort if they receive abort from $\mathcal{F}_{\mathsf{voprf}}$.
- Otherwise, $\mathcal{S}_j$ gets $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho, m)$.
- If $\mathsf{mr} = \mathsf{yes}$:
  - $\mathcal{S}_j$ retrieves $(s'_k)_{k=1}^m, (L_k)_{k=1}^m$ s.t. $(\sigma, m, Y, (s'_k)_{k=1}^m, (L_k)_{k=1}^m) \in \mathcal{L}^{\mathcal{S}_j}$ or aborts;
  - $\mathcal{S}_j$ aborts if $(\mathsf{id}_{\mathcal{R}_i}, \sigma) \in \mathcal{L}_{\mathcal{R}}^{\mathcal{S}_j}$.
- If $\mathsf{mr} = \mathsf{no}$:
  - $\mathcal{S}_j$ samples $(s'_k)_{k=1}^m \xleftarrow{\$} (R \times S)^m$.
  - $\mathcal{S}_j$ defines $L_k := \{((y,k), (0,\tilde{y}) - s'_k) : y \in Y, v \in [t],\ h_v(y) = k\}$.
  - $\mathcal{S}_j$ sends $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \mathtt{evaluate}, \sigma, (L_k)_{k=1}^m)$ to $\mathcal{F}_{\mathsf{voprf}}$.
- $\mathcal{S}_j$ receives $(\mathcal{O}^{F_k})_{k=1}^m$ and $\mathcal{R}_i$ receives $\sigma, N, (F_k(T_x[k]))_{k=1}^m$ from $\mathcal{F}_{\mathsf{voprf}}$.
- $\mathcal{R}_i$ sends $\mathbf{Input}(F_k(T_x[k]), \mathsf{id}_{\mathcal{R}_i}, x'_k)$ and $\mathcal{S}_j$ sends $\mathbf{Input}(s'_k, \mathsf{id}_{\mathcal{S}_i}, s'_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{Add}([\![x'_k]\!], [\![s'_k]\!], b_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{Rerand}([\![b_k]\!], c_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{OutputShares}([\![c_k]\!], r_k, s_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ outputs $(r_k)_{k=1}^{|X|}$ and $\mathcal{S}_j$ outputs $(s_k)_{k=1}^{|X|}$.

Theorem 4. *The protocol* $\Pi_{\mathsf{Core\text{-}PSI}}$ *securely realizes the functionality* $\mathcal{F}_{\mathsf{Core\text{-}PSI}}$ *against semi-honest adversaries in the* $\mathcal{F}_{\mathsf{ABB}}$*-,* $\mathcal{F}_{\mathsf{voprf}}$*-hybrid model with VOPPRF output length* $\ell \geq \lambda + \log_2 m + \log_2 N_c$*, where* m *is the receiver's maximum input set size and* N_c *is*

the number of corrupted parties. More precisely, all sending parties $\mathcal{S}_j$ are allowed to be corrupted and, if $\mathsf{mr} = \mathsf{yes}$ *at most one receiving party $\mathcal{R}_i$ is allowed to be corrupted and multiple executions with the same receiving party $\mathcal{R}_i$ are not allowed if the same sender setup phase (σ, Y) is used.*

Remark 3. The result of Theorem 4 holds up analogously if we replace the ideal functionality $\mathcal{F}_{\mathsf{vopprf}}$ by $\mathcal{F}^*_{\mathsf{vopprf}}$ (see Section 4.3). Adaptively querying entries to the OPPRF functionality does not give a corrupted receiver any extra power since the functions as well as the programmed values are independent for different entries, and the receiver only gets to query each function a single time. Here we again assume that the sender does not reuse their setup phase multiple times with corrupt parties.

Proof. First consider the setting of corrupted senders $\mathcal{S}_i$. The simulator, on input σ_i, Y^i, ρ_i, m_i, $(s^i_k)_{k=1}^{m_i}$, can handle the sender's setup and evaluate queries as the $\mathcal{F}_{\mathsf{vopprf}}$ functionality does. The simulator furthermore samples random identifiers to answer queries to $\mathcal{F}_{\mathsf{ABB}}$, and returns $(s^i_k)_{k=1}^{m_i}$ upon the **OutputShares** query to $\mathcal{F}_{\mathsf{ABB}}$

Now consider the setting of corrupted receivers $\mathcal{R}_i$. The simulator, on input ρ_i, X^i, σ_i, $|Y_i|$, $(r^i_k)_{k=1}^{m_i}$, can sample random answers to $\mathcal{R}'_i s$ evaluate queries to $\mathcal{F}_{\mathsf{vopprf}}$. The queries to $\mathcal{F}_{\mathsf{ABB}}$ can be answered with random identifiers, and the simulator returns $(r^i_k)_{k=1}^{m_i}$ upon the **OutputShares** query to $\mathcal{F}_{\mathsf{ABB}}$. Note that if $\mathsf{mr} = \mathsf{yes}$, only one receiver is allowed to be corrupted if the same sender setup phase is used; so the real-protocol OPPRF outputs are indeed independent and uniformly random for different $i \in [N_c]$.

Furthermore, if there exists a $y \in Y^i$ such that $y = T^i_x[k]$ and a $v \in [t]$ such that $h_v(y) = k$, then the real protocol output shares $(r^i_k)_{k=1}^{m_i}$, $(s^i_k)_{k=1}^{m_i}$ satisfy $r^i_k + s^i_k = (0, \tilde{y})$, with s^i_k sampled uniformly at random. Now $T^i_x[k]$ either holds $x \in X^i$ with $h_{v_x}(x) = k$ for some $v_x \in [t]$, or holds a dummy item $x' \xleftarrow{\$} \mathcal{X}$. The first case implies that all matches $x \in X^i$, $y \in Y^i$ with $x = y$ result in $r^i_k + s^i_k = (0, \tilde{y})$ since the Cuckoo hash procedure gives an injective mapping $x \mapsto h_{v_x}(x) =: k$. In the second case this results in a false positive only if $F_k(x') = (0, \tilde{y}) - s^{i'}_k$, for which the probability is upper bounded by $m_i/2^\ell$ over all buckets $k \in [m_i]$, where $\ell := \log_2 R$ is the output length of the OPPRF. In all other cases, r^i_k and s^i_k are sampled independently and uniformly random. Hence for $\ell \geq \lambda + \log_2 m_i$, the real and simulated transcripts have statistical distance upper bounded by $2^{-\lambda}$. To guarantee indistinguishability for the joint distribution over all N_c corrupted parties we put $m := \max_i\{m_i\}$ and $\ell \geq \lambda + \log_2 m + \log_2 N_c$. □

5.3 Malicious Adversaries

In the malicious setting, the protocol $\Pi_{\mathsf{Core\text{-}PSI}}$ exhibits several vulnerabilities, which we will detail below. In Section 6.2 we will explain how our Core-PSI protocol from a shared-output PRF is able to overcome almost all of these vulnerabilities.

If the receiver is malicious, they are able to slightly enlarge their input set by providing a malformed Cuckoo hash table T. However, this only influences the outcome of the protocol if an item x at $T[k]$ satisfies $h_v(x)$ for some $v \in [t]$. So the most they can do is to fill the empty bins with additional real values instead of dummy values, resulting in an input of size m, which in practice is only a constant value ϵ larger than a honest party's input set (see Section 2.3). Since each entry $T[k]$ is evaluated using an independently random programmed function F_k, the resulting values $r_k := F_k(T_k)$ are all independently random and the receiver can not learn any additional information about the sender's inputs[3]. Finally, a malicious receiver can choose to input different values r'_k to $\mathcal{F}_{\mathsf{ABB}}$ instead of $F_k(T_x[k])$. Note that this only has an effect if the values satisfy $r'_k + s'_k = (0, z)$ for some $z \in S$, since otherwise the **Rerand** call will return a uniformly random value. However, s'_k is uniformly random from the receiver's point of view unless $T_x[k] = (y, k)$

[3] Unless the sender reuses their setup phase, as discussed before

for some programmed (y, k). So the most they can achieve in this way is to change the associated value encoded in shares corresponding to matching values, i.e., such that $r_k + s_k = (0, \tilde{y} + \Delta_k)$ for some $\Delta_k \in S$ of the receiver's choice.

If the sender is malicious, the main area where they can deviate from the protocol is in the programming of the OPPRF. For example, they can choose to only program $(y, h_{v^*}(y))$ for some $v^* \in [t]$ and not program $(y, h_v(y))$ for $v \neq v^*$. Now this has the effect that if $y \in X$, it will only show up in the outputs as $r_k + s_k = (0, \tilde{y})$ if the receiver placed y at $T_x[k]$ in their Cuckoo hash table for $k = h_{v^*}(y)$. So if the sender later learns that $r_k + s_k = (0, \tilde{y})$, this leaks information about the receiver's Cuckoo hashing choices, which might also depend on items outside of the intersection. Note that the sender does not directly learn any additional information from the Core-PSI protocol outputs s_k.

On the more extreme end, the sender can act maliciously in the underlying OPPRF protocol, as discussed in Section 4.3. That is, after receiving random functions F'_k, they can supply an arbitrary vector $\boldsymbol{P}$ such that the programmed function is defined as $F_k(x) := F'_k(x) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(x; \theta)$. It is interesting to consider how the adversary may leverage this choice of $\boldsymbol{P}$ against the receiver. It is clear that the adversary can enforce at most $|\boldsymbol{P}|$ linear relations, which means it can program pre-chosen outputs of at most $|\boldsymbol{P}|$ points. However, instead it can also program at most $|\boldsymbol{P}|$ difference relations of the form

$$(F'_k(y_1) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(y_1; \theta)\rangle) - (F'_k(y_2) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(y_2; \theta)\rangle) = z_1 - z_2. \tag{1}$$

In the context of Core-PSI, the adversary can effectively program the receiver's output secret shares for $|\boldsymbol{P}| + m$ points by choosing its own output shares $(s'_k)_{i=1}^m$ appropriately as

$$s'_k = z_1 - (F'_k(y_1) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(y_1; \theta)\rangle) = z_2 - (F'_k(y_2) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(y_2; \theta)\rangle). \tag{2}$$

This would let a malicious sender program at most $|\boldsymbol{P}| + m \leq q(M(m)) + m$ points in the VOPPRF instead of the $t \cdot |Y| \leq M(m)$ points a honest sender will program, but again does not let them learn any additional information about the receiver's input set.

The same trick might also be used by honest senders to program one extra item in the OPPRF for each bucket for additional savings in the communication cost of the protocol. For the first two items in each bucket $k \in [m]$, they can encode $\boldsymbol{P}$ as in (1) with $z_1 := (0, \tilde{y}_1)$ and $z_2 := (0, \tilde{y}_2)$. Then they can set their share s'_k for bucket k as in (2) and encode the remaining items y in this bucket as usual as $F'_k(y) + \langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(y; \theta)\rangle = (0, \tilde{y}) - s'_k$. Since the encoding $\boldsymbol{P}$ is computed as a solution to a system of equations $\boldsymbol{M} \cdot \boldsymbol{P} = \boldsymbol{v}$, where the entries of $\boldsymbol{v}$ are all independent uniformly random, and the rows of $\boldsymbol{M}$ are now either of the form $d_{|\boldsymbol{P}|}(y; \theta)$ or of the form $d_{|\boldsymbol{P}|}(y_1; \theta) - d_{|\boldsymbol{P}|}(y_2; \theta)$, the vector $\boldsymbol{P}$ will be uniformly random as long as it is selected randomly from all the solutions (provided that a solution exists). Since the receiver only learns a single OPRF evaluation for each index $k \in [m]$, this approach does not leak any additional information about the sender's set. However, since state-of-the art OKVS constructions [RR22, BPSY23] are optimized towards encoding relations of the form $\langle \boldsymbol{P}, d_{|\boldsymbol{P}|}(y; \theta)\rangle = z$ as opposed to of the form $\langle \boldsymbol{P}, (d_{|\boldsymbol{P}|}(y_1; \theta) - d_{|\boldsymbol{P}|}(y_2; \theta))\rangle = z$, it is unclear how this will influence the computational complexity of the resulting Core-PSI protocol. In terms of communication, this could reduce the size of the OKVS encoding from approximately tn to $(t - \epsilon)n$ in the best case, an ≈ 1.73 factor improvement for practical parameters $t = 3$ and $\epsilon = 1.27$. We leave it to future work to explore this approach further.

6 Core-PSI from Shared-Output PRF

In this section we present a novel Core-PSI protocol from a primitive we call a shared-output PRF (SOPRF). This primitive, on input a key K from the sender and an input x from the receiver, outputs secret shares of a (pseudo)random function F evaluated on

(K, x). The main structure of the resulting Core-PSI protocol is similar to the protocol from Section 5.2: The sender and receiver evaluate the SOPRF on the receiver's input set X and a random key K sampled by the sender, obtaining a secret sharing of $F(K, x_i)$ for $i = 1, \ldots, |X|$. The sender can locally evaluate $F(K, y)$ for $y \in Y$ and encode an OKVS sending y to $(0, \tilde{y}) - F(K, y)$. The receiver can now decode this OKVS on x_i and add the result to the SOPRF output. The parties now hold a sharing of $(0, \tilde{y})$ if there exists $(y, \tilde{y}) \in Y$ such that $x_i = y$, and a sharing of a random value otherwise. To make sure the random value corresponding to non-matching entries is independent from the sender's transcript, the parties rerandomize the shared values before receiving the shares. A formal description of the protocol $\Pi^*_{\mathsf{Core\text{-}PSI}}$ is given in Figure 13.

Because the sender's output shares essentially correspond to their SOPRF shares, they are already indexed by the receiver's set indices, and thus the parties do not need to perform the Cuckoo/simple hashing step that is done at the start of the Core-PSI protocol from Section 5.2. Because of this, the protocol realizes a simpler and stronger Core-PSI functionality, which we denote by $\mathcal{F}^*_{\mathsf{Core\text{-}PSI}}$ and which is detailed in Figure 12. Additionally, this limits some of the ways described in Section 5.3 in which a malicious party can deviate from the protocol. The main remaining obstacle towards malicious security is now to guarantee that the sender programs the OKVS correctly. If the underlying function $F(\cdot)$ is modeled as a random oracle, the hardness of the OKVS overfitting game (Definition 2) limits the number of items the sender can encode in the OKVS as $y \mapsto (0, \tilde{y}) - F(K, y)$. Moreover, by observing the queries to $F(\cdot)$, the simulator is able to extract these items from the OKVS encoding. In this way, we are able to realize the functionality $\mathcal{F}^*_{\mathsf{Core\text{-}PSI}}$ with malicious security. It is important to note that we do *not* obtain fully maliciously secure Circuit-PSI in this way, since a malicious party can still corrupt their output shares without the other party noticing. See Section 6.2 for more details. In Section 6.1, we prove that the protocol $\Pi^*_{\mathsf{Core\text{-}PSI}}$ is secure against semi-honest adversaries when the function $F(\cdot)$ is a public PRF. As an example, we show that (an extended version of) the Dodis-Yampolskiy PRF allows for efficient evaluation inside MPC to realize the SOPRF functionality.

Finally, we extend the $\Pi^*_{\mathsf{Core\text{-}PSI}}$ protocol with a re-usable sender setup phase similar to the sender setup phase of the underlying Π_{vopprf} protocol in the $\Pi_{\mathsf{Core\text{-}PSI}}$ protocol from Section 5.2. Notably, we are able to overcome the vulnerability from Remark 2 with our $\Pi^*_{\mathsf{Core\text{-}PSI}}$ protocol, and allow the sender to re-use their setup phase with multiple corrupted receivers, even in the malicious setting. This is because the receiver only obtains a random secret share of the PRF output, and can therefore not fully unmask the programmed values from the sender's OKVS. Additionally, since we took away the simple hashing step, the sender no longer needs to program multiple items to map to the same unmasked value.

6.1 Semi-Honest Setting

We show that when the function $F(\cdot)$ underlying the protocol Π^*_{CorePSI} is instantiated by a PRF, then it realizes the functionality $\mathcal{F}^*_{\mathsf{CorePSI}}$ against semi-honest adversaries. Furthermore, the senders setup phase $(\sigma, \theta, \boldsymbol{P})$ can be securely re-used with multiple corrupted receivers, as well as during multiple executions with the same corrupted receiver. We will refer to a PRF with shared outputs, that is, the output of $\mathbf{Eval}^F([\![x]\!], [\![K]\!])$ where the underlying function F is a PRF, as a shared-output PRF (SOPRF).

Theorem 5. *The protocol Π^*_{CorePSI} realizes the functionality $\mathcal{F}^*_{\mathsf{CorePSI}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{ABB}}$-, $\mathcal{F}_{\mathsf{BB}}$-hybrid model, where the function $F(\cdot)$ is a PRF with output length $\ell \geq \lambda + \log_2 m + \log_2 N_c$, where $m := \max_i |X^i|$ and N_c is the total number of corrupted parties.*

Proof. First consider the setting of corrupted senders $\mathcal{S}_i$. The simulator, on input σ_i, Y^i, $(s^i_k)_{k=1}^{|X^i|}$, can send random identifiers to answer queries to $\mathcal{F}_{\mathsf{ABB}}$, and returns $(s^i_k)_{k=1}^{|X^i|}$ upon the **OutputShares** query to $\mathcal{F}_{\mathsf{ABB}}$.

Figure 12 Ideal Core-PSI functionality $\mathcal{F}^*_{\mathsf{CorePSI}}$.

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_{\mathcal{R}}]$, and sending parties $\mathcal{S}_j$, $j \in [N_{\mathcal{S}}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on an input space $\mathcal{X}$, a key alphabet $\mathcal{K}$, a finite ring S as value alphabet and a finite ring R. $M' \in \mathbb{N}$ is an upper bound. $\mathcal{L}^{\mathcal{S}_j}$ is a list of sender setup phases. Initialize $\mathcal{L}^{\mathcal{S}_j} := \emptyset$.

Functionality:

- On query $(\texttt{setup}, \sigma, Y)$ from $\mathcal{S}_j$:
  - If $\exists Y' : (\sigma, Y') \in \mathcal{L}^{\mathcal{S}}_j$ then do nothing.
  - Else, add (σ, Y) to $\mathcal{L}^{\mathcal{S}_j}$ and send $(\mathsf{id}_{\mathcal{S}_j}, \sigma, |Y|)$ to $\mathcal{A}$.
- On input $(\mathsf{sid}, \texttt{evaluate}, X)$ from $\mathcal{R}_i$ and input $(\mathsf{sid}, \texttt{evaluate}, \sigma, Y)$ from $\mathcal{S}_j$:
  - Abort if $|Y| > M'$ or if $(\sigma, Y) \notin \mathcal{L}^{\mathcal{S}}_j$. Send (sid, σ) to $\mathcal{A}$.
  - For $k = 1, \ldots, |X|$, sample $r_k, s_k \xleftarrow{\$} R \times S$ under the constraint that
$$r_k + s_k = (0, \tilde{y}) \text{ if } \exists (y, \tilde{y}) \in Y \text{ such that } x_k = y.$$
  - Output $|Y|$, $(r_k)_{k=1}^{|X|}$ to $\mathcal{R}_i$ and $(s_k)_{k=1}^{|X|}$ to $\mathcal{S}_j$.

Now consider the setting of corrupted receivers $\mathcal{R}_i$. The simulator, on input X^i, σ_i, $|Y^i|$, $(r^i_k)_{k=1}^{|X^i|}$), can simulate honest sender's setup phases by receiving $(\sigma_i, |Y^i|)$ from $\mathcal{F}^*_{\mathsf{CorePSI}}$ and sampling $\theta^i \xleftarrow{\$} \{0,1\}^\kappa$ and $\boldsymbol{P}^i \xleftarrow{\$} R^{q(|Y^i|)}$. These are computationally indistinguishable from the real-protocol OKVS encodings since the OKVS is doubly oblivious and the function $F(\cdot)$ is a PRF. The simulator furthermore returns random identifiers upon the queries to $\mathcal{F}_{\mathsf{ABB}}$, and returns $(r^i_k)_{k=1}^{|X^i|}$ upon the **OutputShares** call.

It remains to show that the $(r^i_k)_{k=1}^{|X^i|}$, $(s^i_k)_{k=1}^{|X^i|}$ are indistinguishable between the real and simulated transcript. Consider a hybrid which behaves exactly like a real protocol execution, but where F is a truly random function. Note that if there exists $(y^i, \tilde{y}^i) \in Y^i$ such that $x^i_k = y^i$, then $r^i_k + s^i_k = (0, \tilde{y}^i)$, and the shares are sampled uniformly under this constraint. If there does not exist a matching element in Y^i, then $x^{i'}_k + a^i_k = F(\tau^i, x^i_k) + \mathsf{Decode}(\boldsymbol{P}^i, x^i_k; \theta^i)$, where $F(\tau^i, x^i_k)$ is distributed uniformly random and independent from $\mathsf{Decode}(\boldsymbol{P}^i, x^i_k; \theta^i)$; hence $b^i_k := x^{i'}_k + a^i_k = (0, z)$ for some $z \in S$ with probability $2^{-\ell}$. So except with this probability c^i_k is going to be an independent uniformly random value, and the same holds for the outputs r^i_k, s^i_k. The statistical distance between this hybrid and the simulated transcript is therefore upper bounded by $N_c \cdot m \cdot 2^{-\ell} \leq 2^{-\lambda}$. Hence we can conclude that the real and simulated protocol execution are computationally indistinguishable. □

Example 1 (Extended Dodis-Yampolskiy PRF)**.** One candidate function that can be evaluated efficiently inside MPC is the Dodis-Yampolskiy PRF [DY05], which is defined as $F(K, x) := g^{1/(K+x)}$, where $\mathbb{G} = \langle g \rangle$ is a cyclic group of prime order p, $K \xleftarrow{\$} \mathbb{Z}_p$ and $x \in \mathbb{Z}_Q$ for some $Q \in \mathbb{N}$. It is a PRF under the Q-DDHI assumption (See Appendix D) if Q is polynomial in the security parameter κ.[4]

However, for our Core-PSI protocol in Figure 13, it is more convenient if the output of the function F is pseudorandom in a finite field $\mathbb{F}$, since the sender needs to additively mask their associated data $\tilde{y}$ by the output of F. Hence, we let $q = 2p + 1$ be a strong

[4]It is actually proven in [DY05] to be a verifiable random function (VRF) in a bilinear group, which can straightforwardly be adapted to a proof that it is a PRF in a cyclic group

Figure 13 Core-PSI protocol Π^*_{CorePSI}.

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_\mathcal{R}]$, and sending parties $\mathcal{S}_j$, $j \in [N_\mathcal{S}]$, each holding unique identifiers $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on a key alphabet $\mathcal{X}$, a finite ring S as value alphabet and a finite ring R. $\mathcal{L}^{\mathcal{S}_j}$ is a list of sender setup phases. $\mathsf{OKVS} := (\mathsf{Encode}, \mathsf{Decode})$ is a linear OKVS with expansion function $q : \mathbb{N} \to \mathbb{N}$. $F(\cdot) : \{0,1\}^* \to R \times S$ is either a public PRF or a random oracle. Initialize $\mathcal{L}^{\mathcal{S}_j} := \emptyset$. $M \in \mathbb{N}$ is an upper bound.

Subprotocol $\mathsf{Setup}_\mathcal{S}$ with $\mathcal{S}_j$-input (σ, Y)

- If $\exists Y', K, \theta, \boldsymbol{P} : (\sigma, Y', K, \theta, \boldsymbol{P}) \in \mathcal{L}^{\mathcal{S}_j}$, then $\mathcal{S}_j$ does nothing.
- Else:
  - $\mathcal{S}_j$ samples $K \xleftarrow{\$} \mathcal{K}$, $\theta \xleftarrow{\$} \{0,1\}^\kappa$ and computes $\boldsymbol{P} \leftarrow \mathsf{Encode}(\{(y, (0, \tilde{y}) - F(K, \sigma, y)) : (y, \tilde{y}) \in Y\}; \theta)$.
  - $\mathcal{S}_j$ updates $\mathcal{L}^{\mathcal{S}_j} \leftarrow \mathcal{L}^{\mathcal{S}_j} \cup \{(\sigma, Y, \theta, \boldsymbol{P})\}$.
  - $\mathcal{S}_j$ sends $(\mathtt{post}, \mathsf{sid}, \mathsf{id}_\sigma, (\sigma, \theta, \boldsymbol{P}))$ to $\mathcal{F}_{\mathsf{BB}}$.

Subprotocol $\mathsf{Evaluate}$ with $\mathcal{R}_i$-input X and $\mathcal{S}_j$-input σ:

- $\mathcal{S}_j$ sends σ to $\mathcal{R}_i$.
- $\mathcal{R}_i$ sends $(\mathtt{read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BB}}$, searches $\mathcal{M}$ for the first message $(\mathsf{id}_{\mathcal{S}_j}, \mathsf{sid}, \mathsf{id}_\sigma, (\sigma, \theta, \boldsymbol{P}))$ with $\boldsymbol{P} \in R^*$ and $\theta \in \{0,1\}^\kappa$, and aborts if it does not exist or if $|\boldsymbol{P}| > q(M)$.
- $\mathcal{R}_i$ sends $\mathbf{Input}(X, \mathsf{id}_{\mathcal{R}_i}, X)$ and $\mathcal{S}_j$ sends $\mathbf{Input}((K, \sigma), \mathsf{id}_{\mathcal{S}_j}, \tau)$ to $\mathcal{F}_{\mathsf{ABB}}$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{Eval}^F([\![X]\!], [\![\tau]\!], X')$ to $\mathcal{F}_{\mathsf{ABB}}$ to receive $[\![x'_k]\!] = [\![F(\tau, x_k)]\!]$ for $k = 1, \ldots |X|$.
- $\mathcal{R}_i$ sends $\mathbf{Input}(\mathsf{Decode}(\boldsymbol{P}, x_k; \theta), \mathsf{id}_{\mathcal{R}_i}, a_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{Add}([\![x'_k]\!], [\![a_k]\!], b_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{Rerand}([\![b_k]\!], c_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ and $\mathcal{S}_j$ send $\mathbf{OutputShares}([\![c_k]\!], r_k, s_k)$ to $\mathcal{F}_{\mathsf{ABB}}$ for $k = 1, \ldots, |X|$.
- $\mathcal{R}_i$ outputs $(r_k)_{k=1}^{|X|}$ and $\mathcal{S}_j$ outputs $(s_k)_{k=1}^{|X|}$.

prime, $\tilde{g}$ a generator of the multiplicative group $\mathbb{F}_q^*$, and consider the function

$$\tilde{F}(K, x) := \tilde{g}^{[1/(K+x) \bmod p]} \bmod q$$

with $K \xleftarrow{\$} \mathbb{Z}_p$ and $x \in \mathbb{Z}_Q$. We conjecture that this function will be a PRF in $\mathbb{F}_q^*$ (and therefore also in $\mathbb{F}_q$), which we support by the following reasons. First, we note that the distribution of $\{\tilde{g}^y \mid y \xleftarrow{\$} \mathbb{Z}_{q-1}\}$ is computationally indistinguishable from $\{\tilde{g}^y \mid y \xleftarrow{\$} \mathbb{Z}_p\}$ under the discrete logarithm assumption in $\mathbb{F}_q^*$, since the most significant bit of $y \in \mathbb{Z}_{q-1}$ is a hard-core predicate for the discrete logarithm problem [BM82]. Secondly, the modular inverse $1/(K + x) \bmod p$ always exists[5]. Finally, note that $\tilde{F}(K, x)$ reveals the parity of $1/(K + x) \bmod p$, but that this does not reveal any information about the parity of $(K + x) \bmod p$. We can formulate a comparable assumption to the original Q-DDHI assumption, underlying the pseudorandomness of $\tilde{F}$, as the Q-EDDHI assumption in Appendix D. We refer to $\tilde{F}$ as the *extended* Dodis-Yampolskiy PRF.

In Figure 14, we present a protocol for evaluating the extended Dodis-Yampolskiy shared-output PRF (EDY-SOPRF). Correctness of the protocol is easily checked, and privacy follows immediately since the only value $[r(x + K) \bmod p]$ that is opened to the parties is uniformly random. Using preprocessed random shared values and multiplication triples to compute multiplications, the online communication complexity of this protocol consists of $6\,\mathbb{Z}_p$ and $4\,\mathbb{F}_q^*$ elements ($\approx 10 \cdot \log_2 q$ bits) per evaluation.

[5] unless $x \equiv -K \bmod p$, which happens with negligible probability

Figure 14 Semi-Honest protocol $\Pi_{\mathsf{EDY\text{-}SOPRF}}$ in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model

Parameters: Let $\mathbb{F}_q^* = \langle \tilde{g} \rangle$ be the multiplicative group of a finite field of strong prime order $q = 2p + 1$. Extended Dodis-Yampolskiy PRF on input $x \in \mathbb{Z}_Q$ and key $K \in \mathbb{Z}_p$ defined as $F(K, x) := \tilde{g}^{[1/(K+x) \bmod p]} \bmod q$. Sender $\mathcal{S}$ holding a key $K \in \mathbb{Z}_p$ and receiver $\mathcal{R}$ with input $x \in \mathbb{Z}_Q$.

Protocol:

- Compute $[\![x + K]\!]_p \leftarrow \textbf{Add}([\![x]\!]_p, [\![K]\!]_p)$.
- Get $[\![r]\!]_p \leftarrow \textbf{Random}$ and compute $[\![r(x + K)]\!]_p \leftarrow \textbf{Multiply}([\![r]\!]_p, [\![x + K]\!]_p)$.
- Output $[z \bmod p] \leftarrow \textbf{Output}([\![r(x + K)]\!]_p)$ to $\mathcal{R}$ and $\mathcal{S}$.
- $\mathcal{R}$ and $\mathcal{S}$ locally compute $[z^{-1} \bmod p]$ and input $[\![z^{-1}]\!] \leftarrow \textbf{Input}([z^{-1} \bmod p], z^{-1})$.
- Compute $[\![y]\!]_p \leftarrow \textbf{Multiply}([\![z^{-1}]\!]_p, [\![r]\!]_p)$.
- $\mathcal{R}$ and $\mathcal{S}$ call $(y_1, y_2) \leftarrow \textbf{OutputShares}([\![y]\!]_p)$.
- $\mathcal{R}$ and $\mathcal{S}$ locally compute $F_1 := [\tilde{g}^{y_1} \bmod q]$ and $F_2 := [\tilde{g}^{y_2} \bmod q]$, respectively.
- $\mathcal{R}$ inputs $[\![F_1]\!]_q \leftarrow \textbf{Input}(F_1)$ and $\mathcal{S}$ inputs $[\![F_2]\!]_q \leftarrow \textbf{Input}(F_2)$.
- Compute $[\![F(K, x)]\!]_q \leftarrow \textbf{Multiply}([\![F_1]\!]_q, [\![F_2]\!]_q)$.

6.2 Malicious Setting

When the underlying function $F(\cdot)$ is modeled as a random oracle, the protocol Π^*_{CorePSI} is secure against malicious adversaries. The main idea behind this is that we can extract a malicious sender's effective input set corresponding to a setup phase $(\sigma, \theta, \boldsymbol{P})$ by observing their queries (τ, y) to $F(\cdot)$ and checking which of these satisfy $\mathsf{Decode}(\boldsymbol{P}, y; \theta) + F(\tau, y) = (0, \tilde{y})$ for some $\tilde{y} \in S$. By imposing a suitable upper bound $q(M)$ on the size of the encoding $\boldsymbol{P}$, we can guarantee that no more than M' elements will be extracted in this way. In other words, we require the (M, M')-OKVS overfitting game from Definition 2 to be hard. We want to note that even though the protocol realizes the functionality $\mathcal{F}^*_{\mathsf{CorePSI}}$ against malicious adversaries, a corrupted party can still change their output shares before proceeding to the post-computation phase, without the other party being able to notice this. This is of course not satisfactory when one is aiming at a full maliciously secure Circuit-PSI solution. We will refer to the protocol as being "almost maliciously" secure, and discuss this subtlety in more detail in Remark 4.

We evaluated the upper bound $\varepsilon := \binom{Q}{M'} \cdot 2^{(q(M)-M')\ell}$ from [PRTY20, App. A] on the success probability of any adversary against the (M, M')-overfitting game, both with $\ell = \kappa = 128$ or $\ell = \lambda + \log_2 q(M)$ with $\lambda = 40$. Here we put $Q = 2^\ell$, consider some input set sizes $\approx q(M)$ and an appropriate choice of upper bound M' to make ε negligible.

Table 1: Upper bounds ε for an adversary winning the (M, M')-OKVS overfitting problem.

$q(M)$	2^{16}		2^{20}		2^{24}	
M'	$8q(M)$	$4q(M)$	$7q(M)$	$3q(M)$	$6q(M)$	$3q(M)$
ℓ	128	56	128	60	128	64
ε	$< 10^{-10^5}$	$< 10^{-10^5}$	$< 10^{-10^6}$	$< 10^{-10^5}$	$< 10^{-10^8}$	$< 10^{-10^7}$

Theorem 6. *The protocol Π^*_{CorePSI} realizes the functionality $\mathcal{F}^*_{\mathsf{CorePSI}}$ against malicious adversaries in the random oracle, $\mathcal{F}_{\mathsf{ABB}}$-, $\mathcal{F}_{\mathsf{BB}}$-hybrid model, where the function $F(\cdot)$ is modeled as a random oracle. The parameters ℓ, $M \approx q(M)$ and M' can be chosen, for example, as in Table 1.*

Proof. First consider corrupted senders $\mathcal{S}_i$. When $F(\cdot)$ is modeled as a random oracle, the simulator can extract the effective input set $\overline{Y}^i$ of $\mathcal{S}_i$ corresponding to a setup phase σ_i as follows:

- The simulator keeps track of $\mathcal{S}_i$'s queries $(z, F(z))$ to $F(\cdot)$ in a list $\mathcal{Q}_i$, and answers them randomly as usual.
- As $\mathcal{S}_i$ sends σ, fetch $(\sigma, \theta, \boldsymbol{P})$ from the bulletin board, or aborts if it does not exist or $|\boldsymbol{P}| \geq q(M)$.
- Wait for $\mathcal{S}_i$ to send $\mathbf{Input}(\tau, \mathsf{id}_{\mathcal{S}_i}, \tau)$ to $\mathcal{F}_{\mathsf{ABB}}$.
- For queries $((\tau, y), F(\tau, y)) \in \mathcal{Q}_i$, if $\mathsf{Decode}(\boldsymbol{P}, y; \theta) + F(\tau, y) = (0, \tilde{y})$ for some $\tilde{y} \in S$, then add $(y, \tilde{y})$ to $\overline{Y}^i$.
- Send $(\mathtt{setup}, \tau, \overline{Y}^i)$ and $(\mathsf{sid}_i, \mathtt{evaluate}, \tau, \overline{Y}^i)$ to $\mathcal{F}^*_{\mathsf{CorePSI}}$, receive $(s^i_j)_{j=1}^{|X^i|}$.
- Return $(s^i_j)_{j=1}^{|X^i|}$ to $\mathcal{S}^i$ as the output from $\mathcal{F}_{\mathsf{ABB}}$ to the **OutputShares** call.

Note that the s^i_j are distributed uniformly random in $R \times S$ in the simulated as well as in the real protocol outputs. More importantly, we want the joint distribution with the r^i_j to be indistinguishable. In the real protocol execution, if the parties sent $\mathbf{Eval}_F([\![X^i]\!], [\![\tau]\!], X^{i'})$ to $\mathcal{F}_{\mathsf{ABB}}$, then the outputs r^i_j, s^i_j are random shares of $(0, \tilde{y})$ if there exists some value $(y, \tilde{y})$ such that $\mathcal{S}_i$ encoded the OKVS such that $\mathsf{Decode}(\boldsymbol{P}, y; \theta) + F(\tau, y) = (0, \tilde{y})$ and $x^i_j = y$. This agrees with the ideal functionality outputs in the simulated execution. Moreover, the number of items the simulator extracts in $\overline{Y}^i$ is upper bounded by M' with all but negligible probability. If the PRF key τ' which $\mathcal{S}^i$ used to program the OKVS on some matching point $x^i_j = y$ does not agree with the key τ sent to the $\mathcal{F}_{\mathsf{ABB}}$ functionality, then the outputs r^i_j, s^i_j are random shares of an independent uniformly random value unless the first part of $F(\tau, x^i_j)$ and $F(\tau', y)$ agree, which happens with probability $2^{-\ell}$. This agrees with the simulated execution since the simulator will not have added $(y, \tilde{y})$ to $\overline{Y}^i$ in this case. On unprogrammed points $x^i_j \in X^i$, the outputs r^i_j, s^i_j will be random shares of an independent uniformly random value unless the first half of $F(\tau, x^i_j)$ and $-\mathsf{Decode}(\boldsymbol{P}^i, x^i_j; \theta)$ agree, which again happens with probability $2^{-\ell}$. Overall this results in statistical distance at most $|X^i| \cdot 2^{-\ell} \leq 2^{-\lambda}$ between the real and simulated executions (ignoring negligible terms in κ).

Now consider corrupt receivers $\mathcal{R}_i$. The simulator obtains the honest sender's setup phase identifiers (σ_i, m_i) from $\mathcal{F}^*_{\mathsf{CorePSI}}$, and can sample $\theta^i \xleftarrow{\$} \{0,1\}^\kappa$, $\boldsymbol{P}^i \xleftarrow{\$} R^{q(m_i)}$ to simulate the sender's setup phases, which are indistinguishable from the real-protocol OKVS encodings since OKVS is doubly oblivious. The simulator can determine the receiver's effective inputs as follows:

- Wait for $\mathcal{R}_i$ to send $\mathbf{Input}(X^i, \mathsf{id}_{\mathcal{R}_i}, X^i)$ to $\mathcal{F}_{\mathsf{ABB}}$.
- Send $(\mathsf{sid}_i, \mathtt{evaluate}, X^i)$ to $\mathcal{F}^*_{\mathsf{CorePSI}}$ and receive $(r^i_j)_{j=1}^{|X^i|}$.
- Wait for $\mathcal{R}_i$ sends $\mathbf{Input}(a^i_j, \mathsf{id}_{\mathcal{R}_i}, a^i_j)$ to $\mathcal{F}_{\mathsf{ABB}}$.
- If $a^i_j = \mathsf{Decode}(\boldsymbol{P}^i, x^i_j; \theta) + (0, \Delta^i_j)$ for some $\Delta^i_j \in S$, put $r^i_j \leftarrow r^i_j + (0, \Delta^i_j)$. Otherwise replace $r^i_j \xleftarrow{\$} R \times S$.
- Return $(r^i_j)_{j=1}^{|X^i|}$ to $\mathcal{R}_i$ as the output from $\mathcal{F}_{\mathsf{ABB}}$ to the **OutputShares** call.

The r^i_j are again all distributed uniformly random, just as in the real protocol. If $\mathcal{R}_i$ follows the protocol honestly, indistinguishability of the joint distribution with the s^i_j follows since the honest sender $\mathcal{S}$ programmed $\boldsymbol{P}^i$ to map elements $(y, \tilde{y}) \in Y^i$ to $(0, \tilde{y}) - F(K_i, \sigma_i, y)$; hence after rerandomization the values $b^i_j := F(K_i, \sigma_i, x^i_j) + \mathsf{Decode}(\boldsymbol{P}^i, x^i_j; \theta_i)$ satisfy

$$c^i_j = \begin{cases} (0, \tilde{y}) & \text{if } \exists (y, \tilde{y}) \in Y^i \text{ such that } x^i_j = y, \\ \xleftarrow{\$} R \times S & \text{otherwise,} \end{cases}$$

up to the event that $F(K_i, \sigma_i, x^i_j) + \mathsf{Decode}(\boldsymbol{P}^i, x^i_j; \theta_i) = (0, z)$ for some $z \in S$ on an unprogrammed point, which happens with probability $2^{-\ell}$. Now since r^i_j, s^i_j are uniformly

chosen shares of c_j^i, we see that they are distributed statistically close to the ideal functionality outputs on input X^i (and honest sender's input σ_i, Y^i) with distance at most $|X^i| \cdot 2^{-\ell} \leq 2^{-\lambda}$. The only place where a malicious receiver can deviate is by sending wrongly constructed values $a_j^i \neq \mathsf{Decode}(\boldsymbol{P}^i, x_j^i; \theta)$ to $\mathcal{F}_{\mathsf{ABB}}$. These result in valid matches showing up in the output only if and only if $F(K_i, \sigma_i, x_j^i) = (0, z) - a_j^i$ for some $z \in S$. Assuming that $\mathcal{R}_i$ did not query $F(\cdot)$ on any input of the form (K_i, σ_i, x_j^i), equality on the first component happens with probability $2^{-\ell}$. However, $\mathcal{R}_i$ is free to change the second component of the output by sending $a_j^i := \mathsf{Decode}(\boldsymbol{P}^i, x_j^i; \theta) + (0, \Delta_j^i)$ for some Δ_j^i, which has the effect that for matching values $x_j^i = y$, the outputs r_j^i, s_j^i are random shares of $(0, \tilde{y} + \Delta_j^i)$, just as in the simulated execution. Otherwise, it results in r_j^i, s_j^i being independent uniformly random, just as in the simulated execution. Consider a hybrid which behaves exactly as the real execution but aborts if the receiver poses any query of the form (K_i, σ_i, x). This hybrid aborts with at most negligible probability in κ since $K_i \xleftarrow{\$} \mathcal{K}$ from the receiver's point of view, with $|\mathcal{K}| \geq 2^\kappa$. The statistical distance between this hybrid and the simulated execution is upper bounded by $|X^i| \cdot 2^{-\ell} \leq 2^{-\lambda}$ by our above discussion. □

Example 2. To instantiate F any cryptographic pseudorandom function would suffice, but a better choice are functions designed especially for MPC protocols, such as LowMC [ARS+15] and MiMC [AGR+16]. LowMC [ARS+15] is not well-suited for our applications since it operates over $\mathbb{F}_2$, which would require expensive conversion operations when mainly arithmetic shares over a field $\mathbb{F}_p$ of large prime characteristic p are used. MiMC is better suited than LowMC, since it works over $\mathbb{F}_p$ instead of $\mathbb{F}_2$. MiMC requires repeatedly computing a round function $F_i(x) := (x + k + c_i)^3$, where k is the key and the c_i are random constants chosen by the sender. The number of rounds required for $\kappa = 128$ bits of security is $r = 73$. As in [GRR+16], we estimate the communication complexity of the online phase by the number of sent openings needed to perform these multiplications, which gives $3r \cdot \log_2 |\mathbb{F}_p| = 219 \cdot \log_2 |\mathbb{F}_p|$ bits of communication per evaluation. Grassi et al. [GRR+16] implemented several MPC-friendly PRFs in SPDZ [DPSZ12] and concluded that MiMC provides a good compromise between latency and throughput compared to other PRFs which operate over prime fields.

Remark 4. Importantly, we want to note that although the protocol $\Pi^*_{\mathsf{Core\text{-}PSI}}$ realizes the ideal functionality $\mathcal{F}^*_{\mathsf{Core\text{-}PSI}}$ against malicious adversaries, this is not sufficient to obtain a fully maliciously secure Circuit-PSI solution. Namely, even when the parties forward the outputs of the ideal functionality $\mathcal{F}^*_{\mathsf{Core\text{-}PSI}}$ to an ideal functionality for the post-computation, they are free to change their shares in between. To combat this, we would need to adapt the ideal functionality $\mathcal{F}^*_{\mathsf{Core\text{-}PSI}}$ to output *authenticated* shares of the intersection. That is, using the arithmetic black-box formalization, output $[\![c_k]\!]$, where

$$c_k = \begin{cases} (0, \tilde{y}) & \text{if } \exists (y, \tilde{y}) \in Y \text{ such that } x_k = y, \\ \xleftarrow{\$} R \times S & \text{otherwise.} \end{cases}$$

It seems like our protocol $\Pi^*_{\mathsf{Core\text{-}PSI}}$ actually realizes this if we remove the last two lines from Figure 13, but since the receiver can input arbitrary a_k to $\mathcal{F}_{\mathsf{ABB}}$, they can shift b_k by an arbitrary value $\Delta_k \in R \times S$ before rerandomization. As we saw in the proof of Theorem 6, the only consequences are that the receiver can make matching values show up as non-matching values or change the associated value $\tilde{y} \mapsto \tilde{y} + \Delta$ for matching values. Importantly, they are only able to make non-matching values show up as matching values with negligible probability. For these reasons, we refer to our protocol as being "almost maliciously" secure. Finally, since the sender's output shares come directly from the maliciously secure random function evaluation, our protocol *does* realize the stronger functionality described above against *malicious senders* and *semi-honest receivers*. We

leave it to future work to extend our protocol to realize this stronger functionality against malicious receivers as well.

7 Unbalanced Core-PSI

In case the sender's set Y is much larger than the receiver's set size X, that is, $|Y| \gg |X|$, one would like the communication of the Core-PSI protocol to depend sublinearly on $|Y|$. Inspired by the solution from Hetz et al. [HSW23] for asymmetric plain PSI, we propose to combine our Core-PSI protocols with private information retrieval (PIR) to achieve this. The main idea is that, instead of sending the sender's OKVS encoding to the receiver, the receiver uses PIR to decode their items from the OKVS. More precisely, to decode x the receiver queries the entries of $\boldsymbol{P}$ at the indices where $d(x;\theta)$ is non-zero, such that they can locally compute $\mathsf{Decode}(\boldsymbol{P}, x; \theta) = \langle \boldsymbol{P}, d(x;\theta) \rangle$. This approach has concurrently been suggested by Hao et al. [HLP+23], which they named *oblivious key-value retrieval*. The rest of the asymmetric Core-PSI protocol proceeds just as in Figure 11. Hao et al. [HLP+23] make the observation that the receiver only needs to issue PIR queries for the non-dummy entries of their Cuckoo hash table, since the receiver already knows to expect random decoded values on dummy items and therefore might as well sample them locally.

The efficiency of this approach crucially relies on the fact that the OKVS decoding vector $d(x;\theta)$ is sparse, such that only a small number of entries needs to be retrieved using PIR. Hao et al. [HLP+23] opt to use garbled Cuckoo table-like OKVS constructions, such as [PRTY20, GPR+21, RS21, RR22], where the decoding vector consists of a relatively long sparse part and a relatively short dense part. In their solution, the short part of the OKVS encoding vector is transmitted directly to the receiver, whereas the relevant entries from the long part are retrieved using PIR. We instead suggest to use the random band matrix-based OKVS construction from Bienstock et al. [BPSY23], where the decoding vector consists of a short dense vector of length w at a random location, and zeroes everywhere else. Because of this, the receiver only needs to query a single location of the OKVS encoding vector, upon which the sender returns all subsequent w entries. Note that this requires a PIR scheme which performs well on relatively large entries, that is, of length $\mu = w \cdot (\ell + \sigma)$, where ℓ is the OPRF output length required for correctness/security, σ is the length of the associated data and $w = O(\lambda)$, for example, $\mu \approx \lambda \cdot \kappa = 640$ bytes for $\lambda = 40$ and $\kappa = 128$.

One limitation of the unbalanced Circuit-PSI protocol of Hao et al. [HLP+23] is that the sender can neither re-use their OKVS encoding with multiple clients nor multiple times with the same client, because clients would be able to notice collisions similarly to Remark 2. Combining our shared-output PRF based Core-PSI protocol from Figure 13 with PIR decoding, the sender is able to re-use their OKVS encoding. This also lowers the number of PIR queries the receiver needs to make as well as the size of the PIR datastructure since we do not need the Cuckoo hashing and simple hashing steps anymore.

Additionally, one could combine this aproach with the idea from Hetz et al. [HSW23] to split the OKVS encoding $\boldsymbol{P}$ up in β partitions of size $N_{\mathsf{PIR}} = |\boldsymbol{P}|/\beta$ and run PIR instances of size N_{PIR}. In order to not reveal which partitions are being queried, the receiver needs to pose dummy queries to each of the partitions. For the solution using the OKVS from [BPSY23] as described above, the receiver only needs to pose a single query to a single (random) partition to decode an item. Hence, just as in [HSW23], one can use a balls-into-bins type of analysis to determine the total number of queries the receiver needs to make to each partition.

For the underlying PIR scheme one can either opt for a single-server (batch) PIR scheme, such as [HLP+23] does, or for a two-server PIR scheme, such as [HSW23] does. A two-server solution requires us to work in a slightly stronger model where we need the existence of an additional helper server which we assume not to collude with the sender. Note that

PIR schemes without a preprocessing phase can achieve polylogarithmic communication cost per query, but require linear computation cost per query for the server [BIM00]. Many recent works construct practical single-server PIR protocols in the client-dependent preprocessing model [HHCG+22, ZPZS24, MIR23, GZS23]. Practical constructions of two-server PIR schemes also work in this client-dependent preprocessing model [KC21, LP23]. To further amortize sender communication and computation, one would ideally like to combine our re-usable sender setup phase described above with a PIR scheme with a re-usable, client-independent preprocessing phase. A recent breakthrough [LMW23] and follow-up work [OPPW23] increase hope towards such a "doubly-efficient" PIR solution, but unfortunately no truly practical construction for large databases exists so far. We leave it to future work to explore the best choice of PIR scheme to instantiate our construction.

8 Post-processing for Core-PSI

Core-PSI models the core functionality inside Circuit-PSI, making the final comparison step for boolean flags optional, for potentially more efficient Circuit-PSI solutions. Our Core-PSI protocols do include a final rerandomization step to make sure that the receiver's protocol output shares for non-matching items are distributed independent from the sender's transcript. That is, such that even when (derivatives of) the receiver's output shares are later shared with the sender, they are not able to learn any information about the receiver's input items outside of the intersection. We realize that this rerandomization step is not always necessary or that sometimes a secure equality checking step is needed. We will discuss two alternate options for post-processing in this section, but acknowledge that a more detailed study of which type of post-processing is best suited for which application is out of scope for this paper.

Example 3 (**Core-PSI with Polynomial Postcomputation**)**.** As mentioned in Section 1.1, there are scenarios when Core-PSI leads to a more efficient solution compared to Circuit-PSI. For example, consider the setting where a single receiver $\mathcal{R}$ runs an execution of Core-PSI with N different senders $\mathcal{S}_i$ and the parties obtain output shares $(r_k^i)_{k\in[m]}$, $(s_k^i)_{k\in[m]}$, respectively, for $i \in [N]$. Then any post-computation which can be expressed as a multi-variate polynomial in the $m \cdot N$ variables $r_k^i + s_k^i$ can be computed without first computing secure comparisons as in Circuit-PSI. For example, the parties can compute whether an item $k \in [m]$ of the receiver lies in *all* (resp. *at least one*) of the sender's sets by securely computing the *sum* $\sum_i (r_k^i + s_k^i)$ (resp. the *product* $\prod_i (r_k^i + s_k^i)$). Note that for this type of postcomputation, the parties can skip the **Rerand** call that is done at the end of the individual Core-PSI protocols and instead only rerandomize the outputs after evaluating the multivariate polynomial.

Instead of rerandomizing, the parties can securely compare the output of this computation with zero to realize multiparty Circuit-PSI or quorum Circuit-PSI [CDG+21]. This significantly reduces the amount of secure comparisons needed compared to first computing secure comparisons for each $i \in [N]$ and each $k \in [m]$. The above can be generalized to verify whether these statements hold for at least one (resp. all) of the items in a subset of the receiver's set by additionally taking a product (resp. sum) over the index k.

Example 4 (**Circuit-PSI**)**.** To realize the Circuit-PSI functionality from [RS21] (see Fig. 1), the parties can replace the **Rerand** call at the end of our Core-PSI protocols with an **Equality** call to $\mathcal{F}_{\mathsf{ABB}}$. This either results in a semi-honestly secure Circuit-PSI protocol using $\Pi_{\mathsf{Core\text{-}PSI}}$ (or $\Pi^*_{\mathsf{Core\text{-}PSI}}$ when the underlying function is a PRF) or an *"almost maliciously"* secure Circuit-PSI protocol using $\Pi^*_{\mathsf{Core\text{-}PSI}}$ when the underlying function is a random oracle.

Plugging in our semi-honest protocol $\Pi_{\mathsf{Core\text{-}PSI}}$ with our VOLE-VOPRF protocol $\Pi_{\mathsf{vole\text{-}voprf}}$ and a state-of-the art OKVS construction [RR22, BPSY23] leads to a slightly

more communication and computation efficient Circuit-PSI protocol than the state of the art [RR22, BPSY23]. See Section 9 for more details on these instantiations and their comparison to related work.

Plugging in our maliciously secure protocol $\Pi^*_{\mathsf{Core\text{-}PSI}}$, instantiating the random oracle by a cryptographic PRF, and the arithmetic black box functionality by an actively secure MPC protocol such as [DPSZ12], we obtain the first major step towards a maliciously secure Circuit-PSI protocol since the protocol from Huang et al. [HEK12]. Notably, our resulting protocol has linear communication and computation complexity as opposed to the $O(n \log n)$ complexity of [HEK12], but is not able to provide a maliciously secure Circuit-PSI solution all the way through to the post-computation phase, as discussed in Remark 4. For short, our Circuit-PSI protocol is not able to guarantee *authenticated* output shares for the functionality in Figure 1 since a malicious receiver can add arbitrary shifts $b_i \mapsto b_i + \Delta_i$ to their output shares. We would however in this way obtain a protocol that is secure against malicious senders *and* outputs authenticated secret shares. That is, a Circuit-PSI protocol secure against malicious senders and semi-honest receivers.

9 Theoretical Performance

In this section we estimate the communication costs and the amortization savings of our protocols in the following multi-party Core-PSI settings:

- **1-to-N**: A single sender runs N executions of a Core-PSI protocol using the same input set with each of the receivers;
- **N-to-1**: N senders run an execution of a Core-PSI protocol with a single receiver who uses the same input set in each execution;
- **N-to-N**: Each of the $N + 1$ parties runs $2N$ executions using the same input: N times as a receiver with each of the other parties, and again as a sender [6];
- **N-query**: A single sender runs N executions of a Core-PSI protocol using the same input set with the same receiver, who uses a different, relatively small, input set in each execution.

The amortizations that can be used in each setting depend mostly on the sender and receiver setup phase of the underlying VOPRF/SOPRF protocol. If the VOPRF supports a receiver setup phase, then the receiver can re-use this setup phase across multiple executions of the VOPPRF-based Core-PSI protocol from Section 5.2 in the N-to-N and N-to-1 setting. This is the case for the VOLE-VOPRF from Section 3.2 and the CGA-VOPRF from Section 3.3. If the VOPRF supports a sender setup phase, then this means that the sender can re-use the same OKVS encoding in the VOPPRF protocol from Section 4.2, and therefore use the same encoding in the VOPPRF-based Core-PSI protocol from Section 5.2 in the 1-to-N and N-to-N setting. This is the case for the CGA-VOPRF from Section 3.3, but one important caveat is that only one of the parties may be corrupted and multiple executions with the same receiver are not allowed in this setting, as mentioned in Remark 2. In the SOPRF-based Core-PSI protocol from Section 6, the sender *can* re-use their OKVS encoding securely with multiple corrupted receivers, or multiple times with the same receiver, which applies to the 1-to-N, N-to-N and N-query settings. The factors saved by these amortizations are indicated in Table 2. Note that we do not take into account the potential savings in the asymmetric setting by combining our Core-PSI protocols with PIR, as sketched in Section 7, which could lower

[6] Each pair of parties execute Core-PSI twice: in both directions. This is useful since for each receiver, all N executions are with respect to the same cuckoo hash table encoding of that receiver's input, this makes it easy to MPC-compute functions f over that receiver's input with all senders' inputs 'left-joined' to it.

Table 2: Theoretical communication cost of CorePSI protocols. The symbols indicate: * based on our own estimate, replacing the final comparison step by a rerandomization step to realize CorePSI ([RR22] has identical complexity); † satisfying "almost malicious" security. The factor saved in the N-to-1 setting is indicated in red; The factor saved in the 1-to-N amortization is indicated in blue; ♢ At most one party may be corrupted and multiple executions with the same receiver are not allowed in 1-to-N setting.

Protocol	Communication (bits)
[BPSY23]*	$\epsilon\eta\ell n_x + t\eta(\ell+\sigma)n_y + 2^{14.5}\kappa + 2\epsilon(\ell+\sigma)n_x + C_{\mathsf{sh}}(\epsilon\lceil(\ell+\sigma)/\ell\rceil n_x)$
VOLE-VOPRF	$\epsilon\ell n_x/N + t\eta(\ell+\sigma)n_y + 2^{14.5}\kappa + 2\epsilon(\ell+\sigma)n_x + C_{\mathsf{sh}}(\epsilon\lceil(\ell+\sigma)/\ell\rceil n_x)$
CGA-VOPRF♢	$(2+2/N)\epsilon\kappa n_x + t\eta(\ell+\sigma)n_y/N + 2\epsilon(\ell+\sigma)n_x + C_{\mathsf{sh}}(\epsilon\lceil(\ell+\sigma)/\ell\rceil n_x)$
EDY-SOPRF	$(10+2)\omega\lceil(\ell+\sigma)/\omega\rceil n_x + \eta\omega\lceil(\ell+\sigma)/\omega\rceil n_y/N + C_{\mathsf{sh}}(4\lceil(\ell+\sigma)/\omega\rceil n_x)$
MiMC-SOPRF†	$(3r+2+2)\kappa\lceil(\ell+\sigma)/\kappa\rceil n_x + \eta\kappa\lceil(\ell+\sigma)/\kappa\rceil n_y/N + C_{\mathsf{mal}}((2r+2)\lceil(\ell+\sigma)/\kappa\rceil n_x)$

the communication complexity in the 1-to-N and N-query setting even further. We leave it to future work to explore this.

Since we newly introduce the Core-PSI functionality, we compare our protocols to the state-of-the-art semi-honest Circuit-PSI protocols [RR22, BPSY23]. Note that these protocols only differ in the OKVS construction they use; the protocol from [RR22] is in general more efficient, but the protocol from [BPSY23] achieves lower communication, making it more efficient in low-bandwidth settings. We replace the final comparison step in these protocols by a rerandomization step such that they realize our Core-PSI functionality in the semi-honest setting. Our VOLE-based protocol is also computationally more efficient than [RR22, BPSY23], since our VOLE-VOPRF does not require the parties to encode/decode an OKVS to evaluate the OPRF, whereas the VOLE-OPRF used by [RR22, BPSY23] does. Note that all known Circuit-PSI protocols except for [HEK12] target semi-honest security.

The theoretical communication cost of our Core-PSI protocols is detailed in Table 2. In the two-party setting we denote the receiver's set size by n_x and the sender's set size by n_y. In the multi-party setting where there are multiple senders and/or receivers, all input sets are assumed (roughly) equally large of size denoted by n. Except in the N-query setting where the sender has a large input set of size $n_y = n$ and the receiver multiple small query sets of size n_x. Recall that we denote λ for the statistical security parameter, κ for the computational security parameter. Let ϵ denote the expansion factor of the Cuckoo hashing scheme and let t denote the number of hash functions used in the Cuckoo hashing scheme. Furthermore, η denotes the expansion factor of the OKVS. We let $\ell = \lambda + \log_2(\epsilon n_x)$ denote the output length of the VOPRF/SOPRF needed for correctness and denote the associated data length by σ. For the VOLE- and CGA-based VOPRF, we can vary the output length by choosing the hash function H to have output length $\ell + \sigma$. For the EDY-SOPRF, the output length needed for security is $\omega = \log_2 q$, and so the function needs to be evaluated $\lceil(\ell+\sigma)/\omega\rceil$ times to have output length at least $\ell + \sigma$. Similarly for the MiMC-SOPRF the output length needed for security is κ so the function needs to be evaluated $\lceil(\ell+\sigma)/\kappa\rceil$ times, and r denotes the number of rounds for MiMC. Finally, $C_{\mathsf{sh}}(M)$ denotes the communication cost for generating M multiplication triples in the semi-honest setting and $C_{\mathsf{mal}}(M)$ for generating M triples in the malicious setting.

Concrete Estimates. Additionally, in Table 3, we give some concrete estimates for the communication cost of our Core-PSI protocols and the amortization savings in various multiple-execution settings. In the multi-execution settings we estimate the communication cost *per execution* of the party who is involved in multiple executions. That is, in the 1-to-N and N-query settings we consider the sender's communication cost, in the N-to-1 setting the receiver's cost and in the other settings the cost of any party. Both posting and reading a message m to/from the bulletin board is estimated as $|m|$ bits of communication,

Table 3: Theoretical concrete communication cost estimates for our VOPRF/SOPRF-based CorePSI protocols between the unamortized and the applicable amortized multi-party settings. Based on parameter choices: $\epsilon = 1.27$, $\eta = 1.05$, $t = 3$, $r = 73$, $\kappa = 128$. The lowest values in each setting are indicated in **bold**. * Based on our own estimate, replacing the final comparison step by a rerandomization step to realize CorePSI. † Satisfying "almost malicious" security. ◇ At most one party may be corrupted and multiple executions with the same receiver are not allowed. ♡ No amortization is used.

Protocol	**Setting**	**Comm. per execution in bits** ($n = n_y = 2^{20}$, $n_x = 2^{10}$)					
		$N = 2$			$N = 10$		
		$\sigma = 0$	$\sigma = 60$	$\sigma = 1\,980$	$\sigma = 0$	$\sigma = 60$	$\sigma = 1\,980$
[RR22]*	Unamortized	$497n$	$874n$	$12\,909n$	$497n$	$874n$	$12\,909n$
[BPSY23]*	Unamortized	$451n$	$795n$	$11\,787n$	$451n$	$795n$	$11\,787n$
VOLE-VOPRF	Unamortized	**447n**	**792n**	**11 783n**	**447n**	**792n**	**11 783n**
	N-to-1	**409n**	**753n**	**11 745n**	**378n**	**723n**	**11 714n**
	N-to-N	**428n**	**772n**	$11\,764n$	**413n**	**757n**	$11\,749n$
	N-query♡	$207n_y$	$399n_y$	$6\,489n_y$	$207n_y$	$399n_y$	$6\,489n_y$
CGA-VOPRF	Unamortized	$1\,018n$	$1\,362n$	$12\,354n$	$1\,018n$	$1\,362n$	$12\,354n$
	1-to-N◇	$923n$	$1\,172n$	**9 123n**	$847n$	$1\,020n$	**6 538n**
	N-to-1	$855n$	$1\,200n$	$12\,192n$	$725n$	$1\,070n$	$12\,062n$
	N-to-N◇	$889n$	$1\,186n$	**10 657n**	$786n$	$1\,045n$	**9 300n**
	N-query♡	$205n_y$	$396n_y$	$6\,487n_y$	$205n_y$	$396n_y$	$6\,487n_y$
EDY-SOPRF	Unamortized	$26\,807n$	$26\,807n$	$26\,807n$	$26\,807n$	$26\,807n$	$26\,807n$
	1-to-N	$25\,732n$	$25\,732n$	$25\,732n$	$24\,872n$	$24\,872n$	$24\,872n$
	N-to-N	$26\,269n$	$26\,269n$	$26\,269n$	$25\,839n$	$25\,839n$	$25\,839n$
	N-query	$1\,169n_y$	$1\,169n_y$	**1 169n$_y$**	$309n_y$	$309n_y$	**309n$_y$**
MiMC-SOPRF†	Unamortized	$28\,719n$	$28\,719n$	$458\,899n$	$28\,719n$	$28\,719n$	$458\,899n$
	1-to-N	$28\,651n$	$28\,651n$	$457\,824n$	$28\,598n$	$28\,598n$	$456\,964n$
	N-to-N	$25\,685n$	$25\,685n$	$458\,362n$	$28\,658n$	$28\,658n$	$457\,932n$
	N-query	**124n$_y$**	**124n$_y$**	$1\,546n_y$	**70n$_y$**	**70n$_y$**	$694n_y$

these are never counted twice as we only count the communication cost from one side.

We use $\lambda = 40$, $\kappa = 128$, $\epsilon = 1.27$ and $t = 3$, as is common in other comparisons [RS21, RR22, BPSY23]. We use the communication-efficient OKVS construction from Bienstock et al. with expansion facton $\eta = 1.05$. We put $\ell = \lambda + \log_2(\epsilon n_x) \approx 60$ and handle associated data of various lengths $\sigma \in \{0, 60, 1980\}$. In the applicable settings, we consider amortization over $N \in \{2, 10\}$ executions. In the VOLE setting we use $\log_2 |\mathbb{B}| = \lambda + \log_2(\epsilon n_x)$, $\log_2 |\mathbb{F}| = \kappa$ and use the estimate $2^{14.5}\kappa$ from [RR22] for the communication cost of obtaining the VOLE correlation, to facilitate a fair comparison. For the cryptographic group action (CGA) based construction we assume an elliptic curve based instantiation $U := E(\mathbb{F}_q)$ over a field of size $\log_2 |\mathbb{F}_q| = 2\kappa$. For the extended Dodis-Yampolskiy (EDY) based construction we use a field of size $\omega = 2048$ bits, which is common for $\kappa = 128$ bits of security. For the MiMC-based construction we use the estimate from [GRR+16] for evaluating MiMC using a secret-sharing based protocol, sending $3r$ field elements per evaluation during the online phase, where $r = 73$ and $\log_2 |\mathbb{F}_p| = 128$ for a security level $\kappa = 128$. An additional 2 field elements of communication per evaluation are needed to share the parties' inputs as (authenticated) secret shares. We estimate the preprocessing costs using the pseudorandom correlation generator from Boyle et al. [BCG+20]. This gives an estimate of $C_{\mathsf{sh}}(M) := 2w^2 \log_2 2M + 7w^2 \log_2 |\mathbb{F}| + w^2(2\kappa + 3) \log_2 4M$ bits of communication for generating M triples in the semi-honest setting [BCG+20, Thm. 6.5] (using that one can obtain a multiplication triple from two OLE correlations), and $C_{\mathsf{mal}}(M) := (2w + 34w^2) \log_2 M + (26w + 14w^2) \log_2 |\mathbb{F}| + (2w + w^2)(2\kappa + 3) \log_2 2M$ bits of communication for generating M authenticated triples in the malicious setting [BCG+20, Thm. 6.6]. Here w is the number of noise coordinates for the underlying (reducible) ring-LPN assumption, and we choose $w = 64$ for $\kappa = 128$ bits of security and field size $\log_2 |\mathbb{F}| = 128$ since this seems to offer the most favorable tradeoff between communication and computation complexity [BCG+20, Tables 1, 3 & 4]. We (over)estimate the number

of (authenticated) multiplication triples needed to compute the multiplications as $2r$ per evaluation, and an additional 2 authenticated multiplication triples to share the parties' inputs as authenticated shares in the malicious setting. We leave it to future work to optimize the PCG constructions of Boyle et al. [BCG+20] for generating authenticated shares $[s]$ and square multiplication tuples $[s], [s^2]$ for random values s.

Computational Savings. Finally, we would like to point out the computational work saved by using the amortizations for the 1-to-N, N-to-1, N-to-N and N-query settings.

- In the VOLE setting (Section 3.2), the only amortization is that the receiver can re-use their first VOPRF message with multiple receivers in the N-to-1 and N-to-N setting if the underlying VOLE protocol is programmable. This means for a VOLE protocol in the style of Boyle et al. [BCG+19b], the receiver only has to compute an LPN encoding (of length $m = \epsilon n_x$ over a field of size $|\mathbb{B}| = \lambda + \log_2 m$) once for all executions, which saves a significant part of the receiver's computation.
- In the CGA setting (Section 3.3), there are two main avenues for amortization. Let us again assume an elliptic curve based instantiation $E(\mathbb{F}_q)$ over a field of size $\log_2 |\mathbb{F}_q| = 2\kappa$. In the N-to-1 setting, the receiver has to compute ϵn_x of the $2\epsilon n_x$ exponentiations per VOPRF execution only once for all of the N executions, which saves the receiver a factor $2/(1 + 1/N)$ elliptic curve exponentiations. In the 1-to-N setting, the sender only has to compute the OKVS encoding once for all N executions, if at most one receiver is corrupted and the setup phase is not re-used multiple times with the same receiver. This additionally means the sender only has to evaluate the OPRF once on their tn_y inputs for all executions, which saves them a factor $\approx (\epsilon + t)/(\epsilon + t/N)$ elliptic curve exponentiations. In the N-to-N setting each party enjoys each of these amortization in half of the executions.
- In the EDY and MiMC setting (Section 6.1 and 6.2), the only amortization is that the sender can re-use their OKVS encoding with multiple receivers in the 1-to-N, N-to-N and N-query setting, even when all of the receivers are corrupted. This saves the sender a significant amount of work since they only have to evaluate the PRF once on their inputs and only have to compute the OKVS encoding once.

References

[AES03] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97, 2003.

[AFMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In *ASIACRYPT (2)*, volume 12492 of *Lecture Notes in Computer Science*, pages 411–439. Springer, 2020.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[BBD+22] Jeremy Booher, Ross Bowden, Javad Doliskani, Tako Boris Fouotsa, Steven D. Galbraith, Sabrina Kunzweiler, Simon-Philipp Merz, Christophe Petit, Benjamin Smith, Katherine E. Stange, Yan Bo Ti, Christelle Vincent, José Felipe Voloch, Charlotte Weitkämper, and Lukas Zobernig. Failing to hash into supersingular isogeny graphs. Cryptology ePrint Archive, Paper 2022/518, 2022. URL: https://eprint.iacr.org/2022/518.

[BC22] Dung Bui and Geoffroy Couteau. Private set intersection from pseudorandom correlation generators. Cryptology ePrint Archive, Report 2022/334, 2022. https://ia.cr/2022/334.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS*, pages 291–308. ACM, 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO (3)*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518. Springer, 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient Pseudorandom Correlation Generators from Ring-LPN. In *CRYPTO (2)*, volume 12171 of *Lecture Notes in Computer Science*, pages 387–416. Springer, 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *CCS*, pages 896–912. ACM, 2018.

[BDD20] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: efficient secure computation with financial penalties. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 404–420. Springer, 2020.

[BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2016.

[Bia10] Jean-François Biasse. Improvements in the computation of ideal class groups of imaginary quadratic number fields. *Adv. Math. Commun.*, 4(2):141–154, 2010.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2000.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In *ASIACRYPT (1)*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *FOCS*, pages 112–117. IEEE Computer Society, 1982.

[BPSY23] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, PSU and volume-hiding multi-maps. In *USENIX Security Symposium*. USENIX Association, 2023.

[CDG+21] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In *CCS*, pages 1182–1204. ACM, 2021.

[CDJ16] Chongwon Cho, Dana Dachman-Soled, and Stanislaw Jarecki. Efficient concurrent covert computation of string equality and set intersection. In *CT-RSA*, volume 9610 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2016.

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *CCS*, pages 719–728. ACM, 2017.

[CGS22] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPPRF. *Proc. Priv. Enhancing Technol.*, 2022(1):353–372, 2022.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *CCS*, pages 1223–1237. ACM, 2018.

[CLM+18] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: an efficient post-quantum commutative group action. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–427. Springer, 2018.

[Clo18] Cloudflare. Nimbus. https://blog.cloudflare.com/introducing-certificate-transparency-and-nimbus/, 2018. Accessed: 10-10-2013.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *CCS*, pages 1243–1255. ACM, 2017.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS*, pages 1135–1150. ACM, 2021.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *SCN*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Paper 2006/291, 2006. URL: https://eprint.iacr.org/2006/291.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO (3)*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534. Springer, 2021.

[CT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.

[CT12] Emiliano De Cristofaro and Gene Tsudik. Experimenting with fast private set intersection. In *TRUST*, volume 7344 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2012.

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 125–142, 2009.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 2018(4):159–178, 2018.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

[FFK+23] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. SCALLOP: scaling the CSI-FiSh. Cryptology ePrint Archive, Paper 2023/058, 2023. URL: https://eprint.iacr.org/2023/058.

[FHNP16] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *J. Cryptol.*, 29(1):115–155, 2016.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[FNO19] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *WPES@CCS*, pages 14–25. ACM, 2019.

[FNO22] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. 3-party distributed ORAM from oblivious set membership. In *SCN*, volume 13409 of *Lecture Notes in Computer Science*, pages 437–461. Springer, 2022.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *EUROCRYPT (3)*, volume 11478 of *Lecture Notes in Computer Science*, pages 154–185. Springer, 2019.

[Goo13] Google. Certificate transparancy. https://certificate.transparency.dev/, 2013. Accessed: 10-10-2023.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO (2)*, volume 12826 of *Lecture Notes in Computer Science*, pages 395–425. Springer, 2021.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In *CCS*, pages 430–443. ACM, 2016.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *CRYPTO (2)*, volume 11693 of *Lecture Notes in Computer Science*, pages 3–29. Springer, 2019.

[GZS23] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. Efficient pre-processing pir without public-key cryptography. Cryptology ePrint Archive, Paper 2023/1574, 2023. https://eprint.iacr.org/2023/1574. URL: https://eprint.iacr.org/2023/1574.

[Haz15] Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In *TCC (2)*, volume 9015 of *Lecture Notes in Computer Science*, pages 90–120. Springer, 2015.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*. The Internet Society, 2012.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *EC*, pages 78–86. ACM, 1999.

[HHCG+22] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949, 2022. URL: https://eprint.iacr.org/2022/949.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 155–175. Springer, 2008.

[HLP+23] Meng Hao, Weiran Liu, Liqiang Peng, Hongwei Li, Cong Zhang, Hanxiao Chen, and Tianwei Zhang. Unbalanced circuit-psi from oblivious key-value retrieval. Cryptology ePrint Archive, Paper 2023/1636, 2023. https://eprint.iacr.org/2023/1636. URL: https://eprint.iacr.org/2023/1636.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2010.

[HOS17] Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *CSF*, pages 276–291. IEEE Computer Society, 2017.

[HSW23] Laura Hetz, Thomas Schneider, and Christian Weinert. Scaling mobile private contact discovery to billions of users. Cryptology ePrint Archive, Paper 2023/758, 2023. https://eprint.iacr.org/2023/758. URL: https://eprint.iacr.org/2023/758.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *Public Key Cryptography (1)*, volume 10174 of *Lecture Notes in Computer Science*, pages 175–203. Springer, 2017.

[Jac99] Michael J. Jacobson. *Subexponential class group computation in quadratic orders*. PhD thesis, Darmstadt University of Technology, Germany, 1999.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 233–253. Springer, 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 *EuroS&P*, pages 276–291. IEEE, 2016.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2009.

[KC21] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *USENIX Security Symposium*, pages 875–892. USENIX Association, 2021.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, pages 818–829. ACM, 2016.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proc. Priv. Enhancing Technol.*, 2017(4):177–197, 2017.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS*, pages 1257–1272. ACM, 2017.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security Symposium*, pages 1447–1464. USENIX Association, 2019.

[KRTW19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT (2)*, volume 11922 of *Lecture Notes in Computer Science*, pages 636–666. Springer, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[LKLM21] Kristin Lauter, Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. Password monitor: Safeguarding passwords in microsoft edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/, 2021. Accessed: 11-10-2023.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In *STOC*, pages 595–608. ACM, 2023.

[LP23] Arthur Lazzaretti and Charalampos Papamanthou. Treepir: Sublinear-time and polylog-bandwidth private information retrieval from DDH. In *CRYPTO (2)*, volume 14082 of *Lecture Notes in Computer Science*, pages 284–314. Springer, 2023.

[LPR+21] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from PIR with default. In *ASIACRYPT (2)*, volume 13091 of *Lecture Notes in Computer Science*, pages 605–634. Springer, 2021.

[Mea86] Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE Symposium on Security and Privacy*, pages 134–137. IEEE Computer Society, 1986.

[MIR23] Muhammad Haris Mughees, Sun I, and Ling Ren. Simple and practical amortized sublinear private information retrieval. Cryptology ePrint Archive, Paper 2023/1072, 2023. https://eprint.iacr.org/2023/1072. URL: https://eprint.iacr.org/2023/1072.

[MPC18] Luca Melis, Apostolos Pyrgelis, and Emiliano De Cristofaro. On collaborative predictive blacklisting. *Comput. Commun. Rev.*, 48(5):9–20, 2018.

[MPP10] Mark Manulis, Benny Pinkas, and Bertram Poettering. Privacy-preserving group discovery with linear complexity. In *ACNS*, volume 6123 of *Lecture Notes in Computer Science*, pages 420–437, 2010.

[MRR20] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast database joins and PSI for secret shared data. In *CCS*, pages 1271–1287. ACM, 2020.

[MZ22] Hart Montgomery and Mark Zhandry. Full quantum equivalence of group action DLog and CDH, and more. Cryptology ePrint Archive, Paper 2022/1135, 2022. URL: https://eprint.iacr.org/2022/1135.

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

[OPPW23] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. Cryptology ePrint Archive, Paper 2023/1510, 2023. https://eprint.iacr.org/2023/1510. URL: https://eprint.iacr.org/2023/1510.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *ESA*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-Light: Lightweight private set intersection from sparse OT extension. In *CRYPTO (3)*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT (2)*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, pages 515–530. USENIX Association, 2015.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT (3)*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX Security Symposium*, pages 797–812. USENIX Association, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[QYYZ22] Zhi Qiu, Kang Yang, Yu Yu, and Lijing Zhou. Maliciously secure multi-party PSI with lower bandwidth and faster computation. In *ICICS*, volume 13407 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2022.

[RR22] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *CCS*, pages 2505–2517. ACM, 2022.

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In *CRYPTO (4)*, volume 14084 of *Lecture Notes in Computer Science*, pages 602–632. Springer, 2023.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and Circuit-PSI from Vector-OLE. In *EUROCRYPT (2)*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.

[Sha80] Adi Shamir. On the power of commutativity in cryptography. In *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 582–595. Springer, 1980.

[SJ23] Yongha Son and Jinhyuck Jeong. PSI with computation or circuit-psi for unbalanced sets from homomorphic encryption. In *AsiaCCS*, pages 342–356. ACM, 2023.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, 273:305–347, 1971.

[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 *IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE, 2021.

[ZPZS24] M. Zhou, A. Park, W. Zheng, and E. Shi. Piano: Extremely simple, single-server pir with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 54–54, Los Alamitos, CA, USA, may 2024. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00055, doi:10.1109/SP54263.2024.00055.

A Ideal Functionality for CGA-VOPRF

Figure 15 Ideal VOPRF functionality $\mathcal{F}^*_{\mathsf{voprf}}$

Parameters: Receiving parties $\mathcal{R}_i$, $i \in [N_\mathcal{R}]$, and sending parties $\mathcal{S}_j$, $j \in [N_\mathcal{S}]$, each holding a unique identifier $\mathsf{id}_{\mathcal{R}_i}$ and $\mathsf{id}_{\mathcal{S}_j}$, respectively, agreeing on an input space $\mathcal{X}$ and a finite ring R as output space. PRF is an internal random oracle for random functions $\mathcal{F} = \{f : \mathcal{X} \to R\}$. $\mathcal{L}^{\mathcal{R}_i}$ is a list of $\mathcal{R}_i$'s setup phases, initialized empty.
Functionality:

- On query $(\mathtt{setup}, \rho, X)$ from $\mathcal{R}_i$:
  - If $\exists X : (\rho, X) \in \mathcal{L}^{\mathcal{R}_i}$, do nothing.
  - Else, add (ρ, X) to $\mathcal{L}^{\mathcal{R}_i}$ and send $(\mathsf{id}_{\mathcal{R}_i}, \rho, |X|)$ to $\mathcal{A}$.
- If $\mathsf{mr} = \mathsf{yes}$ then on query $(\mathtt{setup}, \sigma, m)$ from $\mathcal{S}_j$, return $(\mathcal{O}^{\mathsf{PRF}(\mathsf{id}_{\mathcal{S}_j}, \sigma, m, k)})_{k=1}^{m}$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{S}_j}, \mathtt{evaluate}, \rho)$ from $\mathcal{R}_i$:
  - If $\exists X : (\rho, X) \in \mathcal{L}^{\mathcal{R}_i}$, send $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \rho, |X|)$ to $\mathcal{S}_j$.
  - Else, send abort to $\mathcal{R}_i$ and $\mathcal{S}_j$.
- On input $(\mathsf{sid}, \mathsf{id}_{\mathcal{R}_i}, \mathtt{evaluate}, \sigma)$ from $\mathcal{S}_j$:
  - If $\mathsf{mr} = \mathsf{no}$, sample $F \xleftarrow{\$} \mathcal{F}^{|X|}$.
  - If $\mathsf{mr} = \mathsf{yes}$, set $F := (\mathsf{PRF}(\mathsf{id}_{\mathcal{S}_j}, \sigma, |X|, k))_{k=1}^{|X|}$.
  - If $\mathcal{R}_i$ is corrupted then send $(\mathsf{id}_{\mathcal{S}_j}, \mathsf{sid}, \sigma)$ to $\mathcal{A}$ and wait until $\mathcal{A}$ has made exactly $|X|$ queries of the form $(\mathsf{sid}, v_k, X'_k)$ with $v_k \in [|X|]$:
    - If $v_k = v_\ell$ for $1 \le \ell < k$, i.e., an index is repeated, then abort.
    - Else set $X_{v_k} := X'_k$ and return $F_{v_k}(X'_k)$.
  - Output $(\mathcal{O}^{F_k})_{k=1}^{|X|}$ to $\mathcal{S}_j$ and $\sigma, (F_k(X_k))_{k=1}^{|X|}$ to $\mathcal{R}_i$.

B Additional Full Proofs

Theorem 7. *The event* FAIL *from the proof of Theorem 2 occurs with at most negligible probability if the $(N_\mathcal{R}, Q)$-OM-CDH-GA assumption (Definition 5) holds with respect to $(\mathcal{U}_\kappa)_{\kappa \in \mathbb{N}}$. Here $N_\mathcal{R}$ is the total number of receiving parties and Q is a (polynomial) upper bound on the total number of H and H' queries the receiving parties $(\mathcal{R}^i)_{i \in [N_\mathcal{R}]}$ make*

Proof. We construct a $(N_\mathcal{R}, Q)$-OM-CDH-GA adversary $\mathcal{B}^{K\star(\cdot), \mathsf{DDH}(\cdot), (\mathcal{R}_i)_i}$, on input $(E, K\star E, a_1, \ldots, a_Q)$, consisting of subroutines $\mathcal{B}_i^{K\star(\cdot), \mathsf{DDH}(\cdot), \mathcal{R}_i}$, $i \in \mathcal{C}_\mathcal{R}$, which each interact with the ideal functionality $\mathcal{F}^*_{\mathsf{voprf}}$ on behalf of $\mathcal{R}^i$ and $\mathcal{A}$. The adversary $\mathcal{B}^{K\star(\cdot), \mathsf{DDH}(\cdot), (\mathcal{R}_i)_i}$ keeps track of query counters c, c', a list of setup phases $\mathcal{L}^\mathcal{B}$, a list $\mathcal{L}^{\mathsf{DH}}$ of CDH pairs $(a, K\star a)$, a bit b^*, lists of random oracle queries $\mathcal{Q}_1$, $\mathcal{Q}_2$ and a simulated bulletin board $\mathcal{M}$ throughout the subroutines. Initialize $b^*, c := 0$, $c' := N_\mathcal{R}$ and $\mathcal{L}^\mathcal{B}, \mathcal{L}^{\mathsf{DH}}, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{M} := \emptyset$.

$\underline{\mathcal{B}_i^{K\star(\cdot), \mathsf{DDH}(\cdot), \mathcal{R}_i}}$:

- When $\mathcal{R}_i$ queries the random oracle H on input $x \in \mathcal{X}$.
  - If $\exists a : (x, a) \in \mathcal{Q}_1$, return $H(x) := a$.
  - Else, return $H(x) := a_c$ and update $\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{(x, a_c)\}$, $c \leftarrow c + 1$.
- Whenever $\mathcal{R}^i$ queries the random oracle H' on $(\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x)$:
  1. If $\exists z : ((\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x), z) \in \mathcal{Q}_2$, return $H'(\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x) := z$.

2. If $\exists a' : (x, a') \in \mathcal{Q}_1$, set $a := a'$. Otherwise, set $a := a_c$ and update $\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{(x, a_c)\}$, $c \leftarrow c + 1$.
3. If $\exists i' \in \mathcal{C}_\mathcal{R}$, $(i', \sigma_{i'}, m_{i'}, (K_j^{\sigma_{i'}})_j) \in \mathcal{L}^\mathcal{B}$ and $j \in [m_{i'}]$ such that $\sigma = \sigma_{i'}$, $m = m_{i'}$ and $u = K_j^{\sigma_{i'}} \star a$, then:
   - For each $l \in \mathcal{C}_\mathcal{R}$ for which there exists $(l, \sigma_l, m_l, (K_j^{\sigma_l})_j) \in \mathcal{L}^\mathcal{B}$ with $(\sigma_l, m_l) = (\sigma_{i'}, m_{i'})$:
     * Send (sid_l, j, x) to $\mathcal{F}^*_{\mathsf{voprf}}$ on behalf of $\mathcal{A}$.
     * If $\mathcal{F}^*_{\mathsf{voprf}}$ returns $F_j^{\sigma_l}(x)$, update $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{((\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x), F_j^{\sigma_l}(x))\}$ and return $H'(\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x) := F_j^{\sigma_l}(x)$ to $\mathcal{R}_i$.
     * If $\mathcal{F}^*_{\mathsf{voprf}}$ returns $\perp$, continue.
   - If the loop does not return anything, output FAIL.
4. Otherwise, sample $z \xleftarrow{\$} \mathcal{Y}$, return $H'(\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x) := z$ and update $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{((\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x), z)\}$.

- When $\mathcal{R}_i$ sends $(\mathtt{post}, \mathsf{sid}, \mathsf{id}_\rho, (\rho, (a_i^\rho)_{i=1}^{m_\rho}))$ to $\mathcal{F}_{\mathsf{ABB}}$, update $\mathcal{M} \leftarrow \mathcal{M} \cup \{(\mathsf{id}_{\mathcal{R}_i}, \mathsf{sid}, \mathsf{id}_\rho, (\rho, (a_i^\rho)_{i=1}^{m_\rho}))\}$.
- When $\mathcal{R}_i$ sends $(\mathtt{read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ABB}}$, return $\mathcal{M}$ to $\mathcal{R}_i$.
- Wait for $\mathcal{R}_i$ to send ρ_i and search $\mathcal{M}$ for the first message $(\mathsf{id}_{\mathcal{R}_i}, \mathsf{sid}, \mathsf{id}_{\rho_i}, (\rho_i, (a_j^{\rho_i})_{j=1}^{m_i}))$ with $a_j^{\rho_i} \in U$ for all $j \in [m_i]$, or abort if it does not exist.
- Send $(\mathsf{sid}_i, \mathsf{id}_\mathcal{S}, \mathtt{evaluate}, \rho_i)$ to $\mathcal{F}^*_{\mathsf{voprf}}$ on behalf of $\mathcal{R}_i$.
- As the honest party $\mathcal{S}$ sends $(\mathsf{sid}_i, \mathsf{id}_{\mathcal{R}_i}, \mathtt{evaluate}, \sigma_i)$ to $\mathcal{F}^*_{\mathsf{voprf}}$, receive $(\mathsf{sid}_i, \sigma_i)$ from $\mathcal{F}^*_{\mathsf{voprf}}$ on behalf of $\mathcal{A}$. Send σ_i to $\mathcal{R}_i$.
  - If $\exists (i', \sigma_{i'}, m_{i'}, (K_j^{\sigma_{i'}})_j) \in \mathcal{L}^\mathcal{B}$ with $(\sigma_{i'}, m_{i'}) = (\sigma_i, m_i)$, put $K_j^{\sigma_i} := K_j^{\sigma_{i'}}$ for each j and update $\mathcal{L}^\mathcal{B} \leftarrow \mathcal{L}^\mathcal{B} \cup \{(i, \sigma_i, m_i, (K_j^{\sigma_i})_j)\}$.
  - Else, if $b^* = 0$, then with probability $1/|\mathcal{C}_\mathcal{R}|$ do:
    * Set $(\sigma^*, m^*) := (\sigma_i, m_i)$.
    * Sample $j^* \xleftarrow{\$} [m^*]$ and sample $K_j^{\sigma^*} \xleftarrow{\$} \mathbb{H}$ for each $j \in [m^*] \setminus \{j^*\}$.
    * Update $\mathcal{L}^\mathcal{B} \leftarrow \mathcal{L}^\mathcal{B} \cup \{(i, \sigma^*, m^*, (K_j^{\sigma^*})_{j \in [m^*] \setminus \{j^*\}})\}$ and $b^* \leftarrow 1$.
  - Else, sample $K_j^{\sigma_i} \xleftarrow{\$} \mathbb{H}$ for each $j \in [m_i]$ and update $\mathcal{L}^\mathcal{B} \leftarrow \mathcal{L}^\mathcal{B} \cup \{(i, \sigma_i, m_i, (K_j^{\sigma_i})_{j \in [m_i]})\}$.
- If $(\sigma_i, m_i) = (\sigma^*, m^*)$:
  - Query $b_{j^*}^{\sigma_i, \rho_i} \leftarrow K \star a_{j^*}^{\rho_i}$, compute $b_j^{\sigma_i, \rho_i} := K_j^{\sigma^*} \star a_j^{\rho_i}$ for each $j \in [m^*] \setminus \{j^*\}$ and send $(b_j^{\sigma_i, \rho_i})_{j=1}^{m_i}$ to $\mathcal{R}_i$. Update $c' \leftarrow c' - 1$.
  - Whenever $\mathcal{R}^i$ queries the random oracle H' on $(\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x)$, simulate H' as before, but before step 3, if nothing has been returned yet, query $b \leftarrow \mathsf{DDH}(E, K \star E, a, u)$, where $a := H(x)$, and if $b = 1$ and $(\sigma, m, j) = (\sigma^*, m^*, j^*)$:
    * Update $\mathcal{L}^{\mathsf{DH}} \leftarrow \mathcal{L}^{\mathsf{DH}} \cup \{(a, u)\}$ and for each $l \in \mathcal{C}^\mathcal{R}$ for which there exists $(l, m_l, \sigma_l, (K_j^{\sigma_l})_j) \in \mathcal{L}^\mathcal{B}$ with $(\sigma_l, m_l) = (\sigma^*, m^*)$:
      · Send (sid_l, j^*, x) to $\mathcal{F}^*_{\mathsf{voprf}}$ on behalf of $\mathcal{A}$.
      · If $\mathcal{F}^*_{\mathsf{voprf}}$ returns $F_{j^*}^{\sigma^*}(x)$, return $H'(\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x) := F_{j^*}^{\sigma^*}(x)$ to $\mathcal{R}_i$ and update $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{((\mathsf{id}_\mathcal{S}, \sigma, m, j, u, x), F_{j^*}^{\sigma^*}(x))\}$.
      · If $\mathcal{F}^*_{\mathsf{voprf}}$ returns $\perp$, continue.
    * If the loop does not return anything, this means that $(\mathcal{R}_i)_{i \in \mathcal{C}_\mathcal{R}}$ computed one more CDH tuple than the number of executions up to this point. For the c' remaining $K \star (\cdot)$ queries:
      · Pick $a \in (a_1, \ldots, a_Q)$ for which $\nexists u : (a, u) \in \mathcal{L}^{\mathsf{DH}}$.
      · Query $u \leftarrow K \star a$ and update $\mathcal{L}^{\mathsf{DH}} \leftarrow \mathcal{L}^{\mathsf{DH}} \cup \{(a, u)\}$, $c' \leftarrow c' - 1$.
      · If $c' = 0$, terminate and output $\mathcal{L}^{\mathsf{DH}}$.
- If $(\sigma_i, m_i) \neq (\sigma^*, m^*)$:

– Compute $b_j^{\sigma_i,\rho_i} := K_j^{\sigma_i} \star a_j^{\rho_i}$ for each $j \in [m_i]$ and send $(b_j^{\sigma_i,\rho_i})_{j=1}^{m_i}$ to $\mathcal{R}_i$.

- Put $\mathsf{transcript}_i := (\mathcal{O}^{H'}, (a_j^{\rho_i})_{j=1}^{m_i}, \sigma_i, (b_j^{\sigma_i,\rho_i})_{j=1}^{m_i}, \{F_j^{\sigma_i}(x) \mid ((\mathsf{id}_{\mathcal{S}}, \sigma_i, m_i, j, K_j^{\sigma_i} \star H(x), x), F_j^{\sigma_i}(x)) \in \mathcal{Q}_2\})$

If none of the subroutines $\mathcal{B}_i$ terminates or outputs FAIL, it is clear that the transcript $\bigcup_{i \in \mathcal{C}_{\mathcal{R}}} \mathsf{transcript}_i$ is distributed identically to the transcript output by $\mathsf{Sim}_{\mathcal{R}}$. We furthermore claim that if the event FAIL occurs for $\mathsf{Sim}_{\mathcal{R}}$, then $\mathcal{B}$ terminates and outputs a set $\mathcal{L}^{\mathsf{DH}}$ consisting of $N_{\mathcal{R}} + 1$ valid CDH tuples with probability $1/(m^* \cdot |\mathcal{C}_{\mathcal{R}}|) \geq 1/(M \cdot N_{\mathcal{R}})$, where M is a (polynomial) upper bound on the receiver's input set size. Indeed, the event FAIL occurs for $\mathsf{Sim}_{\mathcal{R}}^i$ if for any setup phase $\sigma_{i'}$ and any index $j \in [m_{i'}]$, the malicious receivers pose more queries of the form $(\mathsf{id}_{\mathcal{S}}, \sigma_{i'}, m_{i'}, j, K_j^{\sigma_{i'}} \star H(x), x)$ than the number of executions with setup phase $(\sigma_{i'}, m_{i'})$ up to that point. If this occurs, the adversary $\mathcal{B}$ guesses the correct setup phase $\sigma_{i'}$ and index j with probability $1/(m_{i'} \cdot |\mathcal{C}_{\mathcal{R}}|)$. Inserting the OM-CDH-GA challenge elements $a_1, \ldots, a_Q$ as the answers to H oracle queries and "inserting" the group element $K \in \mathbb{H}$ as the key $K_j^{\sigma_{i'}}$, we see that if the event FAIL occurs for this setup phase and index, the malicious receivers' queries give us one more CDH tuple $(K \star E, a, K \star a)$, where $a = H(x)$, than the number of executions with setup phase $\sigma_{i'}$ up to this point. We conclude that the event FAIL occurs with at most negligible probability if the $(N_{\mathcal{R}}, Q)$-OM-CDH-GA assumption holds. □

C Cryptographic Group Actions

Example 5 (Cyclic Groups). Let $U := \mathbb{G}$ be a cyclic group of prime order p and let $\mathbb{H} := \mathbb{Z}_p$. Then clearly the action $\star : \mathbb{Z}_p \times \mathbb{G} \to \mathbb{G}$, $a \star g := g^a$ is regular, and it is efficiently computable using the square-and-multiply algorithm.

Frequently used groups where the vectorization (DLog) and parallelization (CDH) problems are assumed to be hard, and have known efficient hash functions $H : \{0,1\}^* \to \mathbb{G}$ whose hashing path problem is assumed to be hard are:

- The subgroup $\mathbb{G} := \mathcal{QR}_q \subset \mathbb{Z}_q^*$ of quadratic residues in the multiplicative group modulo q, where $q = 2p + 1$ is a safe prime;
- The subgroup $\mathbb{G} := \langle G \rangle \subset E(\mathbb{F}_q)$ of order p in an elliptic curve group, where q is a prime power. In this case, the action $\star : \mathbb{Z}_p \times \mathbb{G} \to \mathbb{G}$ is often written multiplicatively as $a \star G := a \cdot G$.

Example 6 (Supersingular Elliptic Curves). Let $\mathcal{O}$ be an order in an imaginary quadratic number field $\mathbb{Q}(\sqrt{-p})$ with p prime and let $\mathrm{Cl}(\mathcal{O})$ be its ideal class group. Let $E/\mathbb{F}_p$ be an elliptic curve, let $\mathrm{End}_p(E)$ be the ring of $\mathbb{F}_p$-rational isogenies and let $\mathrm{Frob}_p \in \mathrm{End}_p(E)$ denote the Frobenius endomorphism. Then $\mathcal{Ell}_p(\mathcal{O}, \pi) := \{E/\mathbb{F}_p : \mathrm{End}_p(E) \cong \mathcal{O}, \pi = \mathrm{Frob}_p\}/\cong_{\mathbb{F}_p}$ is a principal homogeneous space for $\mathrm{Cl}(\mathcal{O})$ under the action $[\mathfrak{a}] \star E := E/\mathfrak{a}$.

There are however several obstacles that prevent $(\mathcal{O}, \mathrm{Cl}(\mathcal{O}), \star)$ from being a *hashable hard* homogeneous space: (1) It is in general not known how to efficiently sample elements uniformly at random from $\mathrm{Cl}(\mathcal{O})$; (2) The group action $[\mathfrak{a}] \star E$ can in general not be evaluated efficiently; (3) It is not known how to hash efficiently into $\mathcal{Ell}_p(\mathcal{O}, \pi)$ such that the hashing path problem is hard. A common workaround for (1) and (2) are to use a class group $\mathrm{Cl}(\mathcal{O})$ with known structure [BKV19, FFK+23] to factor elements $\mathfrak{a} = \mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$ in terms of prime ideals with small norm, since the group action can be efficiently evaluated for these classes using Vélu's formulas [Vél71]. However, in general it is infeasible to compute the structure of $\mathrm{Cl}(\mathcal{O})$ since the fastest algorithms to do so take time $L_{|\Delta|}[1/2, 1]$ [Jac99, Bia10]. If one only needs to be able to act efficiently with random elements (and their inverses) – as is the case for our OPRF construction – it can be sufficient to

heuristically argue that the class group is generated by some prime ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$ of small norm, as is done for CSIDH [CLM+18]. This latter setting fits within the framework of a *restricted effective group action* (REGA) [AFMP20]. Unfortunately, solving (3) is still an open problem [BBD+22]. Hence at the moment supersingular elliptic curves do not form a suitable candidate for the OPRF construction in Section 3.3.

D Additional Assumptions

Definition 6 (Q-DDHI Assumption [CHL05])**.** Let $(\mathcal{G}_\kappa)_{\kappa\in\mathbb{N}}$ be a family of cyclic groups of prime order and $Q \in \mathbb{N}$. We say that the Q-Decisional Diffie Hellman Inversion (Q-DDHI) assumption holds with respect to $(\mathcal{G}_\kappa)_{\kappa\in\mathbb{N}}$ if for any $\kappa \in \mathbb{N}$, any $(\mathbb{G}, g, p) \leftarrow \mathcal{G}_\kappa$ (where $\mathbb{G} = \langle g\rangle$ and $|\mathbb{G}| = p$):

$$\{(g, g^x, \ldots, g^{x^Q}, g^{1/x}) \mid x \xleftarrow{\$} \mathbb{Z}_p\} \approx_c \{(g, g^x, \ldots, g^{x^Q}, h) \mid x \xleftarrow{\$} \mathbb{Z}_p, h \xleftarrow{\$} \mathbb{G}\}.$$

Definition 7 (Extended Q-DDHI Assumption)**.** Let $(\mathcal{G}_\kappa)_{\kappa\in\mathbb{N}}$ be a family of finite fields $\mathbb{F}_q$ of strong prime order, and let $Q \in \mathbb{N}$. We say that the *Extended* Q-Decisional Diffie-Hellman Inversion (Q-EDDHI) assumption holds with respect to $(\mathcal{G}_\kappa)_{\kappa\in\mathbb{N}}$ if for any $\kappa \in \mathbb{N}$, any $(\mathbb{F}_q, \tilde{g}, q, p) \leftarrow \mathcal{G}_\kappa$ (where $\mathbb{F}_q^* = \langle\tilde{g}\rangle$ and $q = 2p + 1$):

$$\{(\tilde{g}, \tilde{g}^x, \ldots, \tilde{g}^{x^Q}, \tilde{g}^{1/x}) \mid x \xleftarrow{\$} \mathbb{Z}_p\} \approx_c \{(\tilde{g}, \tilde{g}^x, \ldots, \tilde{g}^{x^Q}, h) \mid x \xleftarrow{\$} \mathbb{Z}_p, h \xleftarrow{\$} \mathbb{F}_q^*\}.$$

E Other OPRFs

There are other OPRF protocols which may be plugged into our framework from Section 5 to obtain a Core-PSI protocol. We briefly discuss some of these choices and the amortizations we expect them to allow. Formally proving these statements is left to future work. Note that some of these constructions realize a slightly different OPRF definition where the ideal functionality returns the outputs of the specific PRF in consideration rather than the outputs of a random function as our ideal functionality in Figure 4 does. In these cases, pseudorandomness of the function needs to be proven separately from the simulation-based proof that the protocol realizes the ideal functionality. This definition does however seem more suited for proving security of an OPRF in the standard model. Note that for some of these constructions, the OPRF protocol allowing a sender setup phase coincides with the OPRF being suited for the offline/online PSI protocol structure of [KLS+17].

- The Diffie-Hellman-based OPRF $F_k(x) := H(x)^k$ with a blinded exponentiation protocol for evaluation, was implicitly used in early (semi-honest) PSI protocols [Sha80, Mea86, HFH99, AES03] with security based on the DDH assumption. To the authors' knowledge, no formal simulation-based proof of this protocol realizing an OPRF functionality has been written down. We expect that semi-honest security of the OPRF protocol can be proven under the DDH assumption, but leave this to future work. We do not expect the protocol to satisfy malicious security due to issues with respect to extracting the malicious receiver's inputs. The protocol allows similar sender and receiver setup phases as the protocol from Figure 7, which we expect can be re-used with multiple receivers and senders, respectively. The construction can be generalized to the group action setting similar to the cryptographic group action OPRF from subsection 3.3 as $F_k(x) := k \star H(x)$.

- The Naor-Reingold OPRF $F_{\boldsymbol{k}}(x) := g^{k_0 \cdot \prod_{i=1}^{n} k_i^{x_i}}$ [NR97] with oblivious evaluation using n OTs has been proven passively secure under the DDH assumption by Freedman et al. [FIPR05]. Hazay and Lindell proved that the protocol is actively secure if

the underlying OT protocol is actively secure [HL08, Prop. 2.4 in full version]. We expect that the sender's keys can be re-used with multiple receivers. The protocol can again be generalized to the group action setting as $F_{\boldsymbol{k}}(x) := (k_0 \cdot \prod_{i=1}^{n} k_i^{x_i}) \star E$ for $\boldsymbol{k} := (k_0, \ldots, k_n) \in \mathbb{H}^{n+1}$ and $E \in U$ an arbitrary base element.

- The RSA-based OPRF $F_d(x) := H^2(H^1(x)^d \mod N)$ with a blinded multiplication protocol for evaluation, where $N = pq$ is an RSA modulus and (e, d) is an RSA key pair, was implicitly used in the (semi-honest) PSI protocol of Cristofaro and Tsudik [CT10, CT12] with security based on the one-more-RSA assumption. Again no proof of security as a stand-alone OPRF protocol has been written down, but we expect the proof to be similar to Theorem 2. We expect that the sender can re-use the key pair with multiple receivers.

- The Dodis-Yampolskiy OPRF $F_k(x) := g^{1/(k+x)}$ [DY05] with oblivious evaluation using additively homomorphic encryption has been proven actively secure by Jarecki and Liu [JL09] assuming the hardness of factoring and the decisional q-Diffie-Hellman inversion assumption. We expect the sender can re-use their PRF key and their key pair for the homomorphic encryption scheme with multiple receivers.

- The VOLE-based OPRF from Rindal and Schoppmann [RS21] and the sVOLE-based generalization of Rindall and Raghuraman [RR22] allow the same amortization as the protocol from Section 3.2 by letting the receiver program part of the VOLE correlation and re-use the OKVS encoding of their set with multiple senders.

- Various OT-based OPRF constructions [KKRT16, PRTY19, CM20], which we do not expect to satisfy any meaningful amortizations since the sender does not have any influence over the choice of keys used in the OPRF protocol and the receiver can not program part of the OT correlation [BCG+19b]. As we have seen, the OPRFs from [RS21, RR22] and [BC22] can be seen as VOLE-based variants of the OT-based OPRFs from [PRTY19] and [KKRT16], respectively, and do allow amortization in the multiple sender setting since the receiver can program part of the VOLE correlation. We leave it to future work to explore a VOLE-based variant of the OT-based OPRF protocol of Chase and Miao [CM20].

Finally, we leave it to future work to explore other options for PRFs which can be efficiently evaluated inside MPC, and can therefore be used within our shared-output PRF based Core-PSI protocol in Section 6.