IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 1, 30 pages.
https://doi.org/10.62056/angy4fvtw

Finding Balance in Unbalanced PSI: A New Construction from Single-Server PIR

Chengyu Lin[1], Zeyu Liu[2], Peihan Miao[3] and Max Tromanhauser[4]

[1] Espresso Systems, United States
[2] Yale University, United States
[3] Brown University, United States
[4] Cornell University, United States

Abstract. Private set intersection (PSI) enables two parties to jointly compute the intersection of their private sets without revealing any extra information to each other. In this work, we focus on the unbalanced setting where one party (a powerful server) holds a significantly larger set than the other party (a resource-limited client). We present a new protocol for this setting that achieves a better balance between low client-side storage and efficient online processing.

We first formalize a general framework to transform Private Information Retrieval (PIR) into PSI with techniques used in prior works. Building upon recent advancements in Private Information Retrieval (PIR), specifically the SimplePIR construction (Henzinger et al., USENIX Security'23), combined with our tailored techniques, our construction shows a great improvement in online efficiency. Concretely, when the client holds a single element, our protocol achieves more than 100× faster computation and over 4× lower communication compared to the state-of-the-art unbalanced PSI based on leveled fully homomorphic encryption (Chen et al., CCS'21). The client-side storage is only in the order of tens of megabytes, even for a gigabyte-sized set on the server. Moreover, since the framework is generic, any future improvement in PIR can further improve our construction.

1 Introduction

Consider two parties, each holding a private set of elements, who want to learn the intersection of the two sets without revealing any other information to each other. For example, two companies may want to identify their common customers, or an ad platform and an advertiser may want to determine which consumers who viewed an ad ended up making a purchase.

The above problem can be formulated as *private set intersection (PSI)*, which refers to a specialized secure two-party computation protocol that takes two private sets X, Y as input and outputs their intersection $X \cap Y$ to one or both of the participating parties. Over the years, PSI has found numerous applications in practice, including DNA testing and pattern matching [TKC07], remote diagnostics [BPSW07], online advertising [IKN$^+$20, MPR$^+$20], password breach monitoring [TPY$^+$19, Ali18, LKLM21, APP21], mobile private contact discovery [DRRT18, KRS$^+$19, Mar14, HWS$^+$21], privacy-preserving contact tracing for infectious diseases [TSS$^+$20, CCF$^+$20], and many more. Tremendous progress has been made towards realizing PSI efficiently [KKRT16, RR17, CLR17, PSWW18, PRTY19, PSTY19, CM20, PRTY20, GPR$^+$21, CMdG$^+$21, RS21].

E-mail: linmrain@gmail.com (Chengyu Lin), zeyu.liu@yale.edu (Zeyu Liu), peihan_miao@brown.edu (Peihan Miao), mft55@cornell.edu (Max Tromanhauser)

Received: 2025-01-13 Accepted: 2025-03-11

Most work in PSI focuses on the *balanced* setting where the input sets are similarly sized, which is not the best fit for some real-world use cases such as password breach monitoring and private contact discovery on mobile devices. In these applications, the input set of the service provider is significantly larger than the input set of the user, sometimes by a factor of millions if not billions. If we apply the PSI protocols designed for the balanced setting, both the computation and communication complexity would grow linearly with the size of the larger set. This can be highly prohibitive, especially for a user with limited resources (e.g., a mobile phone or a wearable device).

To accommodate these applications, techniques have been developed for the **unbalanced** setting with **one-sided** output. In particular, a server holding a *large* set X interacts with a resource-limited client holding a *small* set Y in a PSI protocol, where only the client learns the intersection $X \cap Y$ and the server learns nothing.

The existing work on unbalanced PSI with one-sided output can be categorized into two approaches: those based on oblivious pseudorandom function (OPRF) [FIPR05, PSSW09, KLS+17, RA18, KRS+19] and those based on leveled fully homomorphic encryption (FHE) [CLR17, CHLR18, CMdG+21]. They both follow a common paradigm, where the server first performs a one-time, offline pre-processing step on its set X and possibly sends some pre-processed data to the client, which is stored on the client side. Once the client determines its set Y, it can initiate the online phase by sending a PSI query to the server. For practicality, the online computation and communication costs are typically much lower compared to the pre-processing phase.

When evaluating the practical efficiency of PSI protocols in this paradigm, two important metrics to consider are 1) the client's storage requirement after the pre-processing phase, and 2) the online processing speed. Looking at prior work, the OPRF-based constructions achieve fast computation and low communication in the online phase. However, they usually require the client to have a large storage of size $O(|X|)$ from pre-processing. In contrast, most constructions based on FHE do not require client offline storage, but the online processing speed is much slower due to the heavy online computation and high communication overhead, especially on the server side.

> *Can we achieve a balanced approach that has both sublinear client storage and fast online processing?*

1.1 Our Results

In this work, we make positive progress towards addressing the above question by presenting a new PSI protocol that achieves a better balance between these two extremes. Our protocol for unbalanced PSI with one-sided output strikes a favorable trade-off, requiring low (although non-zero) local storage on the client side, while achieving significantly lower online costs compared to the FHE-based constructions. Although our techniques and their connections have appeared in prior work, we provide a systematic treatment and demonstrate their concrete efficiency for PSI.

Approaching unbalanced PSI through the lens of PIR. Private information retrieval (PIR) [CGKS95] is another important cryptographic primitive that shares similarities with unbalanced PSI in terms of the *unbalanced* input size. Informally, PIR allows a client to retrieve a particular entry from a database stored on the server without revealing any information about its query to the server. Notably, a recent line of work [CK20, SACM21, CHK22, LMW23, LP23] adopts the offline/online paradigm, where the server pre-processes the database in the offline phase to enable efficient online query processing.

In this work, we leverage these recent advancements in PIR to achieve similar trade-offs in unbalanced PSI. We summarize the technical ideas behind our results below and give a more detailed construction overview in section 3.

Generic construction using PIR and OPRF. There are two key challenges in constructing PSI from PIR.

The first is the difference in their functionalities. In PSI, the client wants to learn if a particular element y is in the server's set X, while in PIR, the client wants to retrieve a particular entry from the server's database. This gap can be bridged by a variant of PIR known as *PIR by Keyword* or *Keyword PIR* [CGN98], where the server holds a set of elements X and the client holds a single element y. The client wants to learn whether $y \in X$ without revealing any information about y to the server, which is precisely what we need for PSI.[1]

The second challenge comes from the difference in the security guarantees: while PSI requires privacy for both parties, PIR only protects client privacy. This can be solved by leveraging an oblivious pseudorandom function (OPRF) as in [FIPR05]. The server first samples a key k for a PRF $F_k(\cdot)$ and evaluates the PRF on all its elements to obtain $X' := \{F_k(x) \mid x \in X\}$. Next, the client engages in an OPRF protocol with the server to learn all the PRF evaluations on the client's elements, namely $Y' = \{F_k(y) \mid y \in Y\}$, without leaking any information about Y to the server. Now the original PSI problem is reduced to a new PSI problem for $X' \cap Y'$, but we no longer need to protect sender privacy due to the security guarantees of OPRF.

Concrete instantiations and practical efficiency. To leverage recent advancements in PIR, we build our construction using the state-of-the-art single-server offline/online PIR construction SimplePIR [HHCG$^+$23] and the Diffie-Hellman-based OPRF [HFH99, JL10]. Asymptotically, our construction requires an offline storage of $O(\sqrt{|X|})$ on the client side and an online communication of $O(\sqrt{|X|})$, which follows from the SimplePIR construction.

To further enhance the practicality of our construction, we construct keyword PIR on OPRF values in a way that introduces essentially no overhead to the underlying SimplePIR protocol. Notably, we are able to construct our keyword PIR from SimplePIR using only a simple hash to avoid pre-processing overhead. The idea of constructing keyword PIR from SimplePIR also appeared in a prior work [CD24], for which they propose a novel technique called binary fuse filters. However, our benchmarks (section 5) suggest that their approach does not offer a clear computational advantage over our hash-based method when applied to PSI. Thus, it can be an interesting future direction to explore whether plugging binary fuse filters into the PSI framework would be beneficial, as different optimizations may be needed.

Moreover, we provide various techniques tailored for SimplePIR to optimize our concrete efficiency, including re-arranging the database (which is also introduced in [DPC23]), modulus switching, and tight parameter analysis.

Our protocol is particularly efficient when the server's set is significantly larger than the client's set (e.g., when the ratio $|X|/|Y| \geq 2^{20}$). Concretely, when $|Y| = 1$ and $|X|$ is in the range of $2^{20} - 2^{28}$, our construction is more than $100\times$ faster than the state-of-the-art FHE-based PSI [CMdG$^+$21] in terms of online computation and also achieves more than $4\times$ improvement in online communication. Our offline computation remains comparable to prior works, but requires an extra offline communication and client storage. However, this requirement is small: only tens of megabytes for a database that is thousands of megabytes large. For $|Y| > 1$, our protocol does not have an as obvious advantage as the $|Y| = 1$ case, but the online computation is still about one order of magnitude faster than the prior work. We additionally give an estimation of how our construction can be applied to applications such as password breach checkup. Concretely, it costs more than two orders of magnitude less than prior constructions. Our concrete instantiation is based on Learning with Error

[1] Note that in the recent work by Patel et al. [PSY23], Keyword PIR refers to the setting where the client wants to retrieve a data entry associated with y without revealing y to the server, differing from the original definition in [CGN98]. For simplicity, we use the original definition.

(for SimplePIR), Decisional Deffie-Hellman (for OPRF)[2], and random oracle model.

Extensions. We highlight a few extensions to our construction. First, we explore techniques to remove offline storage as well as reduce the round complexity in the online phase. Second, we can extend our construction to achieve labeled PSI [CHLR18], where the server holds associated values v_i for each element $x_i \in X$, and the client learns the values associated with the elements in the intersection, namely $\{v_i | x_i \in X \cap Y\}$.

Finally, we emphasize that further usage of the generic PSI construction may be of independent interest since it can be instantiated with any OPRF and PIR constructions. Therefore, future advancements in these primitives can directly improve the efficiency of our PSI construction. To show this, we make an estimation for PSI from other keyword PIR protocols [PSY23] using the framework we formalize. It also shows an advantage over the prior constructions, while having different trade-offs compared to our construction.

Our contributions. To summarize, we

- formalize the framework for constructing unbalanced PSI with one-sided output from OPRF and PIR[3];
- instantiate our PSI construction with SimplePIR and develop novel techniques to further improve the concrete efficiency (Section 3) and provide formal security proofs for our protocols (Section 4);
- implement our protocol and demonstrate performance improvement compared with prior work, showing that for applications like password breach checkup, our construction offers an appealing solution (only seconds to check against a database with 2^{32} passwords, compared to hundreds of seconds using prior constructions)[4];
- present various extensions to our protocol achieving expanded functionalities and better flexibility.

1.2 Related Work

Unbalanced PSI from OPRF. The first PSI protocol based on the Oblivious Pseudorandom Function (OPRF) was proposed by Freedman et al. [FIPR05]. In their work, they instantiated the OPRF using the renowned Noar-Reingold (NR) pseudorandom function [NR97]. Subsequently, Pinkas et al. [PSSW09] utilized an AES-based OPRF and garbled circuits (GC) [Yao86] to construct another PSI protocol. Building on these developments, Kiss et al. [KLS+17] and Davi Resende and Aranha [RA18] made further contributions, and the state-of-the-art OPRF-based protocol was presented by Kales et al. [KRS+19].

To provide an overview of this line of research, we explain the high-level idea here. Initially, the server generates a secret OPRF key. During the offline/pre-processing stage, the server transmits the OPRF values of all the elements in its larger set to the client through a hash table, usually a Cuckoo filter [FAKM14]. In the subsequent online phase, the client determines the intersection by first evaluating the OPRF values of all the elements in its smaller set and then verifying them against the hash table (or a Cuckoo filter) received earlier. It is important to note that such protocols typically involve linear communication in the server's set during the pre-processing phase and linear communication in the client's set during the online phase.

Unbalanced PSI from FHE. Another line of work on unbalanced PSI is based on leveled FHE [CLR17, CHLR18, CMdG+21]. These works achieve linear communication

[2] Another variant called One-More Gap Deffie-Hellman is needed for malicious security. See section 4.1 for details.

[3] This is implicitly used in [DRRT18] in a non-black-box way. See a more detailed discussion in Section 1.2.

[4] Our implementation is available at doi:10.5281/zenodo.15131756

in the client's set and logarithmic in the server's set. Thus, the local storage requirement of the client is minimized. All of these works are based on the BFV/BGV homomorphic encryption schemes [Bra12, FV12, BGV14] and thus result in a relatively large overhead in terms of online computation and communication.

Building unbalanced PSI from PIR. To the best of our knowledge, [DRRT18, HSW23] are the only works that directly constructs unbalanced PSI from PIR. However, they both rely on two-server PIR, which requires two non-colluding servers holding the same database and thus have a stronger environmental assumption. Furthermore, [DRRT18] construction uses the underlying PIR in a non-black-box way (modifying the underlying scheme accordingly) and thus less generic (as switching their underlying PIR construction requires a non-black-box change). In [DRRT18], their online communication is relatively large: for a database of size hundreds of megabytes, their online communication for a single PSI query is tens of megabytes. In [HSW23], they also require the clients to refresh hints after a certain amount of queries and thus require the client to be stateful.

PIR schemes. Achieving concretely efficient PIR constructions for practical applications has been an active area of research [DC14, KLL$^+$15, GLM16, ABFK16, ACLS18, GH19, PT20, ALP$^+$21a, MCR21, MW22, LLM22]. Classic protocols have no offline phase, and thus have relatively limited online efficiency.

A recent line of work on offline/online PIR [CK20, SACM21, KC21, CHK22, LP22, LMW23, HHCG$^+$23, ZLTS23, LP23, ZPSZ23, FLLP24] take advantage of offline preprocessing together with client storage. The server pre-processes the database and sends some processed data called *hint* to the client. Later, the client uses the hint to query for the data entry and achieves high online efficiency. SimplePIR [HHCG$^+$23] along this line of work achieves the best concrete online efficiency for single-server PIR.

Doubly efficient PIR [BIM00, CHR17, BIPW17, LMW23] is another related line of work, where the server also performs an offline pre-processing step but does not require client-side storage. Instead, it takes advantage of extra storage on the server side to achieve better online efficiency. Recent work by Lin et al. [LMW23] has shown that with a pre-processing step of $\tilde{O}(N)$ time, where N is the size of the database, a client can retrieve the entry with computation and communication cost both being $\mathsf{polylog}(N)$. However, since this work does not provide concrete efficiency but only serves as an asymptotic result (see [OPPW23] for its concrete performance), we do not employ it to realize our PSI protocol.

Keyword PIR. A generic transformation from PIR to keyword PIR was introduced along with its definition [CGN98] but required overhead proportional to the underlying search data structure. Recent progress has introduced a transformation without any overhead [PSY23] but that requires either $O(N^2)$ extra pre-processing time for the server (where N is the database size) or $O(N)$ extra pre-processing time along with enlarged client or server storage. We provide an estimation of how their keyword PIR construction performs compared to ours and prior works, when used for PSI, in section 5.

OPRFs for server privacy. Using OPRFs to achieve server-side privacy was introduced in [FIPR05] to construct a fully-private keyword search protocol from keyword PIR. In keyword search, a client wants to learn the value associated with a keyword in a server's database (or $\perp$ if it is not in the database), which can be reframed as labeled PSI.

2 Preliminaries

Notation. Let $[N]$ denote $\{1, \ldots, N\}$. We use κ, λ to denote the computational and statistical security parameters, respectively. The logarithm always has base 2 unless otherwise specified. $\mathsf{negl}(\cdot)$ denotes a negligible function, i.e., a function f such that

$f(n) < 1/p(n)$ holds for any polynomial $p(n)$ and sufficiently large n. $\mathsf{poly}(\cdot)$ denotes a polynomial function. PPT stands for "probabilistic polynomial time." $\log(\cdot)$ denotes a logarithmic function. $\mathsf{polylog}(\cdot)$ denotes a poly-logarithmic function. We omit a $\mathsf{polylog}(N)$ factor in $\tilde{O}(\cdot)$, namely $\tilde{O}(N) = O(N\mathsf{polylog}(N))$. $|x|$ denotes the size of x, where x can be a set or a vector. Let x be a vector, $x[i]$ denotes the i-th element of the vector. Let $x \xleftarrow{\$} \mathbb{Z}_q$ denote x being sampled uniformly at random from $\mathbb{Z}_q$.

Private Set Intersection (PSI). PSI is a specialized secure two-party computation [Yao86]. We follow the standard ideal/real-world paradigm for defining secure two-party computation against semi-honest or malicious adversaries (see e.g., [Lin16] for the formal definitions). The ideal functionality of PSI is formalized in fig. 1.

Public Parameters. The honest server and client have respective set sizes N and M. If the server is maliciously corrupted, then its set size is N'.

Inputs. The server S inputs a set X where $|X| = N$ if S is honest and $|X| = N'$ otherwise. The client C inputs a set Y where $|Y| = M$.

Output. The client C receives the set intersection $I = X \cap Y$ and the server S receives $\perp$

Figure 1: Ideal functionality for private set intersection.

Private Information Retrieval (PIR). We formalize an offline/online PIR protocol as follows. A server holds a database T of N data entries, and a client wants to access $T[i]$ for some $i \in [N]$. The server can pre-process the database during the offline phase, and send the pre-processed data hint to the client. During the online phase, the client sends some query qry to the server. The server replies with rsp.

The *correctness* of PIR guarantees that with hint and rsp, the client can correctly recover $T[i]$. The *receiver privacy* of PIR guarantees that for any $i \neq i' \in [N]$, qry for i is indistinguishable from qry for i' to the server.

Decisional Diffie-Hellman (DDH) Assumption. Let g be a generator of a group $\mathbb{G}$ of order q. The DDH problem is hard in $\mathbb{G}$ if for any PPT adversary $\mathcal{A}$, $|\Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g^a, g^b, g^c) = 1]| \leq \mathsf{negl}(\kappa)$, for the probability over the random $a, b, c \xleftarrow{\$} \mathbb{Z}_q$.

One-More Gap Diffie-Hellman (OMGDH) Assumption. Let g be a generator of a group $\mathbb{G}$ of order q. We say (N, Q)-OMGDH is hard in $\mathbb{G}$ if for any PPT adversary $\mathcal{A}$, $\Pr[\{(g_i, g_i^k)\}_{i \in [Q+1]} \leftarrow \mathcal{A}^{(\cdot)^k, DL_k(\cdot,\cdot)}(g_1, \ldots, g_N)] \leq \mathsf{negl}(\kappa)$, where the probability is over random $(g_1, \ldots, g_N) \xleftarrow{\$} \mathbb{G}^N$ and $k \xleftarrow{\$} \mathbb{Z}_q$; $(\cdot)^k$ is an oracle that takes any $h \in \mathbb{G}$ and returns h^k, and $\mathcal{A}$ can call this oracle at most Q times in parallel; $DL_k(\cdot,\cdot)$ is an oracle that on input tuple (g, h) returns 1 if $h = g^k$ and 0 otherwise.

Learning with Error (LWE). Let n, q, σ and distribution $\mathcal{D}$ be LWE parameters, and let χ_σ denote a discrete Gaussian distribution with mean 0 and standard deviation of σ. LWE is hard if for any PPT adversary $\mathcal{A}$, $|\Pr[\mathcal{A}(\vec{a}, u) = 1] - \Pr[\mathcal{A}(\vec{a}, \langle \vec{a}, \vec{s} \rangle + e)]| \leq \mathsf{negl}(\kappa)$, where probability is over $\vec{a} \xleftarrow{\$} \mathbb{Z}_q, u \xleftarrow{\$} \mathbb{Z}_q$, $\vec{s} \leftarrow \mathcal{D}$, and $e \leftarrow \chi_\sigma$.

Regev Encryption. The LWE Regev encryption scheme has an additional parameter p as *plaintext modulus*. The encryption of a $\mathbb{Z}_p$ element m under secret key $\vec{s} \leftarrow \mathcal{D}$ is $(\vec{a}, b \leftarrow \langle \vec{a}, \vec{s} \rangle + e + m \cdot \Delta)$ where $\Delta = \lfloor q/p \rfloor, e \leftarrow \chi_\sigma$ and $\vec{a} \xleftarrow{\$} \mathbb{Z}_q$. With all but negligible probability, it can be correctly decrypted to $m = \left\lceil \frac{b - \langle \vec{a}, \vec{s} \rangle}{\Delta} \right\rfloor$ as long as $\Pr[|e| > \lfloor \Delta/2 \rfloor] \leq \mathsf{negl}(\kappa)$, which means $\mathsf{erf}(\frac{\Delta/2}{\sqrt{2}\sigma}) \leq \mathsf{negl}(\kappa)$, where $\mathsf{erf}(\cdot)$ is the Gauss error function.

The LWE Regev encryption is linearly homomorphic. Let $(\vec{a}, b)$ be the encryption of m and $(\vec{a}', b')$ be the encryption of m'. The encryption of $c \cdot m$ for any plaintext $c \in \mathbb{Z}_p$ can be obtained by a scalar multiplication $c \cdot (\vec{a}, b) = (c \cdot \vec{a}, c \cdot b)$. The encryption of $m + m'$ can be obtained by the entry-wise addition of the ciphertext vectors $(\vec{a}, b) + (\vec{a}', b') = (\vec{a} + \vec{a}', b + b')$. Note that both operations require the resulting error to remain sufficiently small.

3 Our PSI Protocol

In this section, we present our unbalanced PSI protocol with one-sided output. We give a construction overview in section 3.1 and discuss various optimization techniques when instantiating our protocol with SimplePIR in section 3.2. The protocols for clients holding a single element and multiple elements are presented in fig. 2 and fig. 3, respectively.

3.1 Construction Overview

Starting point. We start with the extremely unbalanced PSI problem where the client's set contains a single element. Specifically, the server S holds a large set X of size N and the client C holds a single element y. The client wants to learn whether $y \in X$.

We first follow the OPRF-based PSI paradigm [FIPR05]. Specifically, the server S generates a secret key k for a pseudorandom function (PRF) $F_k(\cdot)$ and sends all the PRF evaluations of its elements, $X' := \{F_k(x) | x \in X\}$, to the client C. Afterwards, S and C engage in an OPRF protocol, which is a specialized secure two-party computation protocol, where C learns $y' = F_k(y)$ and S learns nothing. Finally, C simply checks whether $y' \in X'$. By the security guarantees of OPRF, S learns nothing about y while C learns nothing about k beyond $F_k(y)$, hence $X' \setminus \{y'\}$ is computationally indistinguishable from a random set. Therefore, C learns nothing other than whether $y \in X$.

However, this protocol requires $O(N)$ communication from the server to the client, which can be impractical for a large set X. Moreover, if the OPRF evaluations X' are sent in the pre-processing phase, it would require significant storage on the client side.

Embedding keyword PIR. To address the issue above, we can utilize a variant of PIR named *PIR by Keyword* or *Keyword PIR* [CGN98] instead of requiring the server to send the entire set X'. In keyword PIR, the server holds N elements $S = \{s_1, \ldots, s_N\}$ and the client holds a single element w. The client wants to learn whether $w = s_j$ for some $j \in [N]$ without revealing any information about w to the server. This primitive directly serves our purpose. In more detail, after S computes X' and C obtains y' from OPRF, C can make a keyword PIR query to learn whether $y' \in X'$ without revealing y' to S. The remaining challenge lies in constructing an efficient keyword PIR protocol.

Constructing keyword PIR from PIR. We can now plug any generic keyword PIR protocol into our framework. In section 5.2, we provide a performance estimation for unbalanced PSI from other keyword PIR constructions [PSY23], following our framework.

Nevertheless, we present an alternative approach to constructing keyword PIR from PIR, which is particularly tailored for optimal performance when instantiated with SimplePIR (see section 3.2). This approach can also achieve malicious security almost for free. In summary, we do hashing to bins as used in [ACLS18, ALP+21b, LPR+20]. Specifically, we construct keyword PIR from PIR in a black-box way using a hash function $H : \{0,1\}^* \to [\tau]$ that maps elements into a hash table of size τ[5].

The server S first creates a hash table T of size τ, putting every element $x' \in X'$ into the hash bin $T[H(x')]$, namely $T[\ell] := \{x' | x' \in X' \wedge H(x') = \ell\}$ for all $\ell \in [\tau]$. We can bound the maximum number of elements in any hash bin (with overwhelming probability), denoted by γ. Then the server pads each hash bin with dummy elements to reach a size of γ. We can now view T as a database consisting of τ entries, where each entry contains γ elements. The client C then simply computes $\ell^{\mathsf{C}} := H(y')$ and makes a PIR query for

[5] H does not need to be a cryptographic hash function.

$T[\ell^{\mathsf{C}}]$. Finally, C can conclude $y' \in X'$ if and only if $y' \in T[\ell^{\mathsf{C}}]$. By the receiver security of PIR, S does not learn anything about y'.

Regarding the parameters, we first set $\tau = N = |X|$, which results in $\gamma = O(\log N \log\log N)$.[6] These parameters can be further tuned for improved performance, as discussed in section 3.2. Given any PIR with sublinear communication complexity in N, we can achieve sublinear communication in our protocol as well.

Optimization: padding to the largest bin. In the above keyword PIR construction, we observe that the size of each hash bin in T does *not* reveal any information to the client. This is because the elements in the hash table are PRF values and hash locations are computed based on these PRF values, which can be sent directly to the client as in the OPRF-based PSI protocol. Therefore, there is no need for any padding in terms of security guarantees. For the PIR protocol to go through, it suffices to pad each bin to the size of the actual largest bin in T instead of the theoretical upper bound on the maximum size of any bin, which drastically reduces the size of the database for PIR. Moreover, the server can pad with 0-strings to further reduce the computational cost in PIR.[7]

The OPRF construction. The missing component in our construction is the realization of OPRF. Prior works on PSI have proposed various types of OPRF constructions, including Noar-Reingold-based [FIPR05], Diffie-Hellman-based [HFH99, JL10], garbled circuit-based [PSSW09, KLS+17, RA18, KRS+19], and OT-based [KKRT16, PRTY19, CM20]. Our construction is generic and can work with any OPRF, but to best serve our PSI purpose, we look for OPRF constructions that satisfy the following properties: 1) the server's OPRF key k can be reused across multiple clients, 2) the protocol can easily be made maliciously secure, and 3) the protocol is practically efficient, especially in the online phase. Considering these factors, we choose the OPRF construction presented in [JL10].

In this construction, S and C agree on two hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \mathbb{G} \times \mathbb{G} \to \{0,1\}^\delta$. The PRF is computed as $F_k(x) := H_2(H_1(x), H_1(x)^k)$ for a randomly sampled key k. To jointly compute $F_k(y)$, C randomly samples k_{C} and sends $z := H_1(y)^{k_{\mathsf{C}}}$ to S. S then replies with $z' := z^k$. After getting z' back, C computes $z'' \leftarrow H_2(z, (z')^{k_{\mathsf{C}}^{-1}})$, which gives $F_k(y)$.

Achieving malicious security. The above protocol is semi-honest secure in the random oracle model assuming DDH is hard in $\mathbb{G}$. To enhance its security against malicious adversaries, S only needs to attach a proof of knowledge (PoK) for the key k along with its response z', assuming OMGDH is hard in $\mathbb{G}$ (as in [JL10], but also pointed out by [CNCG+23, dCL24]). See a more detailed discussion in Section 4.2.

Handling multiple elements in the client's set. Now we discuss the scenario where the client C has multiple elements in its set, namely C holds a set Y of size $M \geq 1$. One straightforward approach is to apply the single-element PSI on every element in Y. However, this approach can be computationally expensive on the server side if the server's online computational complexity in PIR grows linearly with the database size, which is the case in most PIR protocols.

To reduce the server's computation cost, we adopt the technique of Cuckoo hashing [PR04]. Specifically, S and C agree on three hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [m]$ to map elements into a hash table of size m. The client C first creates a hash table of size m and puts each element $y_i \in Y$ into one of three bins located at $\{h_1(y_i), h_2(y_i), h_3(y_i)\}$, ensuring that each hash bin contains at most one element. The Cuckoo hashing parameter m is chosen such that this step fails with negligible probability. On the server side, S also creates a hash table of size m and puts each element $x_i \in X$ into all three bins located

[6] As long as $\gamma = \omega(\log N)$, the probability that any bin exceeds the size of γ is negligible.

[7] Note that this is possible since the padding is not required for security. Padding is simply to satisfy the requirements of SimplePIR, where the database is viewed as a square, and thus we need to ensure that each bin has the same size (i.e., each row has the same length).

Inputs: The server S holds a large set $X = \{x_1, \ldots, x_N\}$ where $x_i \in \{0,1\}^*$ for each $i \in [N]$ (assume X is randomly shuffled). The client C holds a single element $y \in \{0,1\}^*$.

Setup: S and C agree on the security parameters κ, λ, protocol parameters N, δ, τ, a cyclic group $\mathbb{G}$ of prime order q with generator g, three hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \mathbb{G} \times \mathbb{G} \to \{0,1\}^\delta$, and $H_3 : \{0,1\}^\delta \to [\tau]$.

Pre-processing Phase: S does the following:

1. Randomly sample $k_{\mathsf{S}} \xleftarrow{\$} \mathbb{Z}_q$.
2. Initialize an empty table T of size τ, namely $T[i] := \emptyset$ for all $i \in [\tau]$.
3. For each $i \in [N]$:
   (a) Compute $u_i := H_2(H_1(x_i), H_1(x_i)^{k_{\mathsf{S}}})$ and $\ell_i := H_3(u_i)$.
   (b) Let $T[\ell_i] := T[\ell_i] \cup \{u_i\}$.
4. Let γ denote the size of the largest entry in T, namely $\gamma := \max_{i \in [\tau]} |T[i]|$. For each $i \in [\tau]$, if $|T[i]| < \gamma$, then pad it with dummy strings of length δ (e.g., 0^δ) to reach a size of γ.
5. View T as a database with τ entries, each entry containing a set of γ strings of length δ. Perform the pre-processing step of PIR, and send the pre-processed data hint to C, together with γ.

Online Phase:

1. C randomly samples $k_{\mathsf{C}} \xleftarrow{\$} \mathbb{Z}_q$, computes $z := H_1(y)^{k_{\mathsf{C}}}$, and sends z to S.
2. Upon receiving z, S computes $z' := z^{k_{\mathsf{S}}}$ and sends it back to C.
3. Upon receiving z' back, C does the following:
   (a) Compute $z'' := H_2\left(H_1(y), (z')^{k_{\mathsf{C}}^{-1}}\right)$ and $\ell^{\mathsf{C}} := H_3(z'')$.
   (b) Prepare a PIR query for the ℓ^{C}-th entry of the database T and send it to S.
4. Upon receiving the PIR query, S computes the PIR response and sends it back to C.
5. Upon receiving the PIR response, C does the following:
   (a) Recover the entire ℓ^{C}-th entry of T as a set of strings $R = \{r_1, \ldots, r_\gamma\}$.
   (b) Output $\{y\}$ if $z'' \in R$ and $\emptyset$ otherwise.

Figure 2: Our PSI protocol where the client has a single element.

at $\{h_1(x_i), h_2(x_i), h_3(x_i)\}$. We can then bound the maximum number of elements in any hash bin and have the server pad each hash bin with dummy elements to reach that size. Finally, S and C run a single-element PSI protocol for each hash bin. As a result, the server's total online computation remains linear in the database size.

Optimization: no need for padding. Consider the padding step in the above multi-element protocol. Now it is necessary, for security reasons, to pad each hash bin to the theoretical upper bound since the size of each bin could reveal information about the server's set X. However, if we apply the Cuckoo hashing on the PRF values instead of the original elements, the same idea applies, and padding is no longer required.

3.2 Tailoring SimplePIR

We instantiate the above PSI construction based on SimplePIR [HHCG+23], which is the fastest single-server PIR scheme known to date with sublinear communication.[8] We now open this black box for better performance.

SimplePIR. SimplePIR essentially realizes the square root PIR introduced in [KO97] with an LWE Regev encryption scheme using preprocessing to boost online performance.

[8]While Piano [ZPSZ23] is faster in terms of online time, the communication cost in pre-processing is linear in N. Although the local storage of the client is not linear in N, this linear pre-processing communication does not directly fit the PSI applications we have in mind.

Inputs: The server S holds a large set $X = \{x_1, \ldots, x_N\}$ where $x_i \in \{0,1\}^*$ for each $i \in [N]$ (assume X is randomly shuffled). The client C holds a small set $Y = \{y_1, \ldots, y_M\}$ where $y_i \in \{0,1\}^*$ for each $i \in [M]$.

Setup: S and C agree on the security parameters κ, λ, protocol parameters N, M, m, δ, a cyclic group $\mathbb{G}$ of prime order q with generator g, five hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \mathbb{G} \times \mathbb{G} \to \{0,1\}^\delta$, and $h_1, h_2, h_3 : \{0,1\}^\delta \to [m]$.

Pre-processing Phase: S does the following:

1. Randomly sample $k_{\mathsf{S}} \xleftarrow{\$} \mathbb{Z}_q$
2. Initialize m empty hash bins $X_1, \ldots, X_m := \emptyset$.
3. For each $i \in [N]$:
   (a) Compute $u_i := H_2(H_1(x_i), H_1(x_i)^{k_{\mathsf{S}}})$.
   (b) Let $X_j := X_j \cup \{u_i\}$ for each $j \in \{h_1(u_i), h_2(u_i), h_3(u_i)\}$.
4. For each $j \in [m]$, let $N_j = |X_j|$ and denote $X_j = \{u_{j,1}, \ldots, u_{j,N_j}\}$. Proceed as in the single-element pre-processing phase:
   (a) Choose a parameter τ_j and a hash function $H_{3,j} : \{0,1\}^\delta \to [\tau_j]$.
   (b) Initialize an empty table T_j of size τ_j, namely $T_j[i] := \emptyset$ for all $i \in [\tau_j]$.
   (c) For each $i \in [N_j]$, compute $\ell_{j,i} = H_{3,j}(u_{j,i})$ and let $T_j[\ell_{j,i}] := T[\ell_{j,i}] \cup \{u_{j,i}\}$.
   (d) Let γ_j denote the size of the largest entry in T_j, namely $\gamma_j := \max_{i \in [\tau_j]} |T_j[i]|$. For each $i \in [\tau_j]$, if $|T_j[i]| < \gamma$, then pad it with dummy strings of length δ to reach a size of γ_j.
   (e) View T_j as a database with τ_j entries, each entry containing a set of γ_j strings of length δ. Perform the pre-processing step of PIR to obtain hint_j.
5. Send $\{(N_j, \tau_j, H_{3,j}, \gamma_j, \mathsf{hint}_j)\}_{j \in [m]}$ to C.

Online Phase:

1. C randomly samples $k_{\mathsf{C}} \xleftarrow{\$} \mathbb{Z}_q$, computes $z_i := H_1(y_i)^{k_{\mathsf{C}}}$ for all $i \in [M]$, and sends $\{z_i\}_{i \in [M]}$ to S.
2. Upon receiving $\{z_i\}_{i \in [M]}$, S computes $z'_i := z_i^{k_{\mathsf{S}}}$ for all $i \in [M]$ and sends $\{z'_i\}_{i \in [M]}$ back to C.
3. Upon receiving $\{z'_i\}_{i \in [M]}$ back, C does the following:
   (a) For each $i \in [M]$, compute $z''_i := H_2\left(H_1(y_i), (z'_i)^{k_{\mathsf{C}}^{-1}}\right)$.
   (b) Initialize m empty hash bins $Y_1, \ldots, Y_m := \emptyset$.
   (c) Perform Cuckoo hashing using h_1, h_2, h_3 on $\{z''_i\}_{i \in [M]}$ and put (y_i, z''_i) into one of the hash bins $Y_{h_1(z''_i)}, Y_{h_2(z''_i)}, Y_{h_3(z''_i)}$ such that each bin contains exactly one tuple. Pad each empty bin with a dummy random tuple.
   (d) For each $j \in [m]$, let the tuple in Y_j be (y_j, z''_j):
      i. Compute $\ell_j^{\mathsf{C}} := H_{3,j}(z''_j)$.
      ii. Prepare a PIR query qry_j for the ℓ_j^{C}-th entry of the database T_j.
   (e) Send all the PIR queries $\{\mathsf{qry}_j\}_{j \in [m]}$ to S.
4. Upon receiving the PIR queries, S computes the response rsp_j for qry_j using T_j for each $j \in [m]$, and sends $\{\mathsf{rsp}_j\}_{j \in [m]}$ back to C.
5. Upon receiving the PIR responses, C does the following:
   (a) For each $j \in [m]$, recover the entire ℓ_j^{C}-th entry of T_j as a set of strings $R_j = \{r_{j,1}, \ldots, r_{j,\gamma_j}\}$.
   (b) Output the intersection $I := \left\{y_j | z''_j \in R_j, j \in [m]\right\}$.

Figure 3: Our PSI protocol where the client has multiple elements.

Specifically, for a database of size N with each data entry in $\mathbb{Z}_p$, SimplePIR models it as a matrix $D \in \mathbb{Z}_p^{\sqrt{N}} \times \mathbb{Z}_p^{\sqrt{N}}$. Retrieval of a single element is done by retrieving the entire column of D where the target element lies.

The client sends an encrypted indicator vector $\mathsf{qry} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_{\sqrt{N}})$ where ct_i encrypts 1 if the queried entry is in the i-th column and ct_i encrypts 0 otherwise. Since the LWE Regev encryption scheme is linearly homomorphic, the server replies with the result from a homomorphic matrix-vector multiplication $\mathsf{rsp} \leftarrow D \times \mathsf{qry}$, which encrypts the i-th column of the matrix D. The underlying homomorphic operations are either homomorphic addition or scalar multiplication, since the database D is given in plaintext form.

Dive into more details. We can view qry as a $\sqrt{N}$-by-$(n+1)$ matrix where each row corresponds to an LWE Regev ciphertext of the form $(\vec{a}, b) \in \mathbb{Z}_q^{n+1}$ where n is the LWE dimension, q is the ciphertext modulus and $\vec{a} \in \mathbb{Z}_q^n$ is sampled uniformly.

The homomorphic matrix-vector multiplication $D \times \mathsf{qry}$ can be viewed as a matrix multiplication of a $\sqrt{N}$-by-$\sqrt{N}$ matrix D and a $\sqrt{N}$-by-$(n+1)$ matrix qry, whose each row corresponds to a ciphertext $\mathsf{ct}_i = (\vec{a}_i, b_i)$. Now let $\mathsf{qry}_a = [a_1^\top, a_2^\top, \ldots, a_{\sqrt{N}}^\top]^\top$ be the first n columns of query matrix qry and $\mathsf{qry}_b = [b_1, b_2, \ldots, b_{\sqrt{N}}]^\top$ be the last column of qry. The result ciphertext vector, or viewed as the result matrix, is $D \times \mathsf{qry} = D \times [\mathsf{qry}_a, \mathsf{qry}_b] = [D \times \mathsf{qry}_a, D \times \mathsf{qry}_b]$.

A significant insight of SimplePIR is that the first part, $D \times \mathsf{qry}_a$, only involves multiplying the database D with a uniformly random matrix qry_a, which is independent of the client's input. Exploiting this property, the server S can generate a random matrix qry_a by uniformly sampling it from a short seed s. During the pre-processing phase, the server sends both the seed s and $\mathsf{hint} = D \times \mathsf{qry}_a$ to the client. In the subsequent online phase, the client reconstructs qry_a using the seed s and generates the final column qry_b of the query ciphertext matrix [9]. The server's computation is significantly reduced to performing a smaller matrix-vector multiplication, $\mathsf{rsp} \leftarrow D \times \mathsf{qry}_b$, during the online phase. The server then sends rsp back to the client. Finally, the client decrypts the combined result, considering both the received $\mathsf{hint} = D \times \mathsf{qry}_a$ and rsp.

The hint is thus of size $\sqrt{N} \cdot n \cdot \log q$ (which is the pre-processing communication cost), and the online upload and download communication are both $\sqrt{N} \cdot \log q$ (which together is the online communication cost). The server needs to evaluate $N \cdot n$ $\mathbb{Z}_q$-multiplications and additions during the pre-processing phase, and N $\mathbb{Z}_q$-multiplications and additions during the online phase.

With this background, we can proceed to find more balances when instantiating the underlying PIR protocol with SimplePIR.

Re-arrange the database. SimplePIR arranges its database into a square $\sqrt{N}$-by-$\sqrt{N}$ matrix D. To find a better balance between the hint size and the rsp size, we can re-arrange D to be rectangular $\in \mathbb{Z}_p^{\alpha \times \beta}$, without changing the pre-processing and online computation cost. The hint size (i.e. the size of $D \times \mathsf{qry}_a$) is then $\alpha \cdot n \cdot \log q$. The upload cost (i.e. the size of qry_b) is $\beta \cdot \log q$. The download cost (i.e. the size of $D \times \mathsf{qry}_b$) is $\alpha \cdot n \cdot \log q$. The parameters α and β can be chosen according to the application. We discuss our choices below. Note that re-arranging the database is also used in [DPC23].

Retrieving multiple elements for free. A key observation of SimplePIR is that we are retrieving back an entire column of database matrix D, which contains α $\mathbb{Z}_p$ elements. Thus, while an entry in hash table T has $\gamma \cdot \delta$ bits, where γ is the number of elements per data entry, by setting $\alpha = C \cdot \gamma \cdot \delta / \lfloor \log p \rfloor$, for some $C \in \mathbb{Z}^+$, we can retrieve the entire hash table entry with a single PIR query.

Tuning the hash table size τ. Recall that our table T has $\tau \cdot (\gamma \cdot \delta)$ bits, where γ is the size of T's largest entry. Every entry with fewer than γ elements needs to be padded to γ elements (with zeros). Thus, as we decrease the table size τ, the variance in the entry sizes in T becomes smaller, hence the database size $\tau \cdot (\gamma \cdot \delta)$ also decreases.

[9] Note that reusing the same matrix for polynomial amount of queries is still secure as shown in [PVW08].

As mentioned above, we need $\alpha = C \cdot \gamma \cdot \delta / \lfloor \log p \rfloor$ to retrieve the entire entry with one query and thus $\beta = \tau / C$, for some $C \in \mathbb{Z}^+$. However, as discussed, efficiency grows when τ decreases. Thus, if we set $\beta = \tau/C$ for some $C > 1$, we can instead simply set a new $\tau' \leftarrow \tau/C, \beta \leftarrow \tau'$, and use τ' as the size of T for better efficiency. Thus, we set $\beta = \tau$ and $\alpha = \gamma \cdot \delta / \lfloor \log p \rfloor$ (i.e., $C = 1$), and then directly adjust τ for better efficiency.

This results in $|\mathsf{hint}| = \gamma \cdot \delta / \lfloor \log p \rfloor \cdot n \cdot \lceil \log q \rceil$ bits, $|\mathsf{qry}| = \tau \cdot \lceil \log q \rceil$ bits, and $|\mathsf{rsp}| = |\mathsf{hint}|/n$ bits. Thus, we can adjust τ according to the desired $\mathsf{hint}, \mathsf{qry}, \mathsf{rsp}$ sizes. For our purpose, we set τ such that $|\mathsf{hint}| \approx \sqrt{|X| \cdot \delta}$, where $|X| \cdot \delta$ is the cost of sending the entire X' (recall that $X' := \{F_k(x) | x \in X\}$).

Modulus switching. We introduce an additional technique to further reduce the communication of SimplePIR, called modulus switching [BV11, DM15]. Recall that $\mathsf{hint} \in \mathbb{Z}_q^{\alpha \times n}, \mathsf{rsp} \in \mathbb{Z}_q^{\alpha \times 1}$ and $(\mathsf{hint}, \mathsf{rsp})$ together forms α LWE ciphertexts.

Recall that an LWE ciphertext $(\vec{a}, b) \in \mathbb{Z}_q^{n+1}$ with respect to secret key $\vec{s} \in \mathbb{Z}^n$ satisfies the following: $b - \langle \vec{a}, \vec{s} \rangle = e + m \cdot \Delta$ where $e \in \mathbb{Z}_q$ is a small error, $m \in \mathbb{Z}_p$ is a message, and $\Delta = \lfloor q/p \rfloor$. There is usually a big gap between the ciphertext modulus q and the plaintext modulus p (i.e., $p \ll q$). Thus, we can work on a smaller ciphertext modulus $q' < q$ to reduce the communication, while preserving the LWE ciphertext structure. A modulus switching procedure for LWE ciphertexts from q to $q' \leq q$ is defined as $(\vec{a}', b') \leftarrow \mathsf{round}(\frac{q'}{q}(\vec{a}, b)) \in \mathbb{Z}_{q'}^{n+1}$. The resulting ciphertext $(\vec{a}', b')$ satisfies $b' - \langle \vec{a}', \vec{s} \rangle = e' + m \cdot \Delta'$, where $e \in \mathbb{Z}_q$ is a new error term (to-be bounded), and $\Delta' = \lfloor q'/p \rfloor$.

This means that when sending $\mathsf{hint} \in \mathbb{Z}_q^{\alpha \times n}$ and $\mathsf{rsp} \in \mathbb{Z}_q^{\alpha}$, we can modulus switch them down to some $q' \ll q$ before sending them back. The communication cost can thus be reduced by a factor of $\log q / \log(q')$. The correctness holds as long as $\Pr[|e'| \leq \Delta/2] \geq 1 - \mathsf{negl}(\lambda)$.

Ternary LWE keys and randomized rounding. To make sure e' is small, we employ two additional techniques. The first is using *ternary keys* for LWE secret keys, namely sample the secret key as $s \xleftarrow{\$} \{0, 1, -1\}^n$. The second is *randomized rounding.* Specifically, to round a decimal value $c.d$, we round it to $c+1$ with probability d, and round it to c with probability $1 - d$. By using these two techniques, $e' = O(\frac{q'}{q} e + \sqrt{n})$ [DM15]. Concretely, if e has a standard deviation of σ (for Gaussian distribution χ_σ), then e' has a standard deviation of $\sigma' = \frac{q'}{q}\sigma + \sigma_{MS}$ where $\sigma_{MS} = \sqrt{\frac{n^2+1}{3}}$ (for Gaussian distribution $\chi_{\sigma'}$) [LMP22, Sec 6.5].

Error analysis. Recall that qry is essentially LWE ciphertexts with some initial error, $D \times \mathsf{qry}$ thus results in new LWE ciphertexts with larger errors. As shown in [HHCG$^+$23, Sec C.2], to guarantee the correctness of SimplePIR, we need to choose LWE parameters σ (i.e., the error distribution standard deviation for the initial error generation) to satisfy: $2\exp(-\pi \cdot (\frac{\Delta}{2 \cdot \sigma \cdot \sqrt{2\pi} \cdot \sqrt{\beta \cdot p/2}})) \leq 2^{-\lambda}$, where $\Delta = \lfloor q/p \rfloor$.

Combining with modulus switching, we choose $\Delta_1 + \Delta_2 = \Delta$, σ, such that they satisfy $2\exp(-\pi \cdot (\frac{\Delta_1}{2 \cdot \sigma \cdot \sqrt{2\pi} \cdot \sqrt{\beta \cdot p/2}})) \leq 2^{-\lambda}/2$, and $\mathsf{erf}(\frac{(q'/q)\Delta_2/2}{\sqrt{2}\sigma_{MS}}) \leq 2^{-\lambda}/2$ [10]. By union bound, we have PIR correctness with probability $\geq 1 - 2^{-\lambda}/2 - 2^{-\lambda}/2 = 1 - 2^{-\lambda}$.

This is done via the following: given an initial q, p, calculate an initial Δ as specified above, and let the initial $\Delta_1 = \Delta_2 = \Delta/2$. If such a Δ_1 does not satisfy the first equation, then reduce p by a factor of 2, recalculate $\Delta, \Delta_1, \Delta_2$ until it is satisfied. Afterwards, find the minimum q' such that the second equation is satisfied (and if $q' = q$, we simply eliminate the modulus switching process). This process can be repeated by assigning Δ_1 and Δ_2 differently (e.g., $\Delta_1 = 0.9\Delta$, $\Delta_2 = 0.1\Delta$) to maximize p and q'.

[10] Recall that $e \leftarrow \chi_\sigma$, $\Pr[e \geq \Delta/2] \leq \mathsf{erf}(\frac{\Delta/2}{\sqrt{2}\sigma})$.

3.3 Parameters

In this section, we summarize the parameters required in our single-element PSI protocol (fig. 2) and multi-element PSI protocol (fig. 3).

- Computational security parameter κ and statistical security parameter λ.
- Server set size N and client set size M (in fig. 3).
- H_2's output length δ such that $(N \cdot M)/2^{\delta} \leq 2^{-\lambda}$.
- m (in fig. 3) such that Cuckoo hashing M elements into m bins fails with probability $\mathsf{negl}(\lambda)$.
- Hash table size $\tau = N$ (in fig. 2) and $\tau_j = N_j$ for each $j \in [m]$ (in fig. 3) (these parameters can be further tuned for optimized performance, as discussed in section 3.2).
- PIR parameters such that the underlying PIR protocol satisfies both correctness and receiver privacy.

4 Security Guarantees

4.1 Corrupted Client

In the existence of a corrupted client, our PSI protocols (fig. 2 for $M = 1$ and fig. 3 for an arbitrary M) achieve semi-honest security in the standard PSI definition shown in fig. 1 and malicious security in the adaptive variant of the PSI functionality [JL10], which allows adaptive queries from C. In more detail, the ideal functionality takes a set X from S as input, and for each query on input y_i made by C, for $i \in [M]$, the ideal functionality returns yes or no for whether $y_i \in X$. Although it is secure for the weaker adaptive PSI functionality, we can in fact show that a malicious client cannot change its input set after sending $\{z_i\}_{i \in [M]}$ in the online phase Step 1.

Remark 1. Note that the notion of "adaptive functionality" is different from "adaptive adversary" in standard MPC definitions. Specifically, "adaptive PSI functionality" indicates that a malicious client is allowed to adaptively query if an element y_i is in X in the ideal functionality. This is different from the standard PSI definition, where a malicious client is required to send its entire set Y to the ideal functionality all at once. It is weaker than the standard definition because the adversary is given more power in the ideal world. Adaptivity is needed in the security proof because the client's queries may only be extractable eventually (from random oracle queries), rather than during protocol execution. Although security is proven with the adaptive functionality, we can show that a malicious client cannot change its input set after sending $\{z_i\}$ in the online phase Step 1, hence it is unclear what advantage a malicious client could obtain from the adaptive feature of the functionality.

We state the theorems below and give the security proofs in section A.1 and section A.2, respectively. Note that theorem 2 achieves stronger malicious security by relying on a stronger computational assumption, namely OMGDH. The proofs for the single-element protocol in fig. 2 follow similarly.

Theorem 1. *If H_1, H_2 are modeled as random oracles and DDH is hard in $\mathbb{G}$, then our protocol in fig. 3 securely computes the PSI functionality in fig. 1 against a semi-honest client when the protocol parameters are chosen as described in section 3.3.*

Theorem 2. *If H_1, H_2 are modeled as random oracles, the OMGDH problem is hard in $\mathbb{G}$, then our protocol in fig. 3 securely computes the adaptive PSI functionality against a malicious client when the protocol parameters are chosen as described in section 3.3.*

4.2 Corrupted Server

In the existence of a corrupted server, our PSI protocols (fig. 2 for $M = 1$ and fig. 3 for an arbitrary M) achieve simulation-based security against a semi-honest server and client privacy against a malicious server in the standard PSI definition. We discuss extensions to serving multiple clients with multiple elements in their sets in section 4.2.2, as well as the challenges in proving full simulation-based security.

4.2.1 Original Protocol

Semi-honest server. We state the semi-honest security for an arbitrary M below and give the security proof in section A.3. The proof for $M = 1$ follows similarly.

Theorem 3. *If H_1, H_2 are modeled as random oracles, the DDH problem is hard in $\mathbb{G}$, and the underlying PIR protocol satisfies correctness and receiver privacy, then our protocol in fig. 3 securely computes the PSI functionality in fig. 1 against a semi-honest server when the protocol parameters are chosen as described in section 3.3.*

Client privacy against malicious server. We can achieve client privacy against a malicious server without making any changes to our protocol. At a high level, it means that the server cannot learn anything about the client's input from the interaction transcript. This security guarantee is the same as the one achieved in [CHLR18, CMdG$^+$21]. We state the theorem below and skip the proof as it follows the exact same structure as the proof of theorem 3 in arguing for client privacy.

Theorem 4. *If H_1 is modeled as a random oracle, the DDH problem is hard in $\mathbb{G}$, and the underlying PIR protocol satisfies receiver privacy, then our protocol in fig. 3 achieves client privacy [HL08, Def 2.2] against a malicious server when the protocol parameters are chosen as described in section 3.3.*

4.2.2 Full Security Against Malicious Server

In addition to client privacy, prior works [CHLR18, CMdG$^+$21] on FHE-based unbalanced PSI proposed techniques to achieve simulation-based security with leakage against a malicious server. At a high level, the server in their protocols needs to homomorphically compute and return an encrypted $H(z)$ for a public hash function H and an OPRF value z encrypted by the client. Their assumption is that the server cannot homomorphically compute an encryption of $H(z)$ given an encryption of z and some pre-determined list of encryptions of powers of z, when H is a sufficiently complex hash function such as SHA256. The heuristic argument of this assumption comes from the difficulty of evaluating such a high-depth circuit using leveled HE, where the parameters are chosen to support a smaller multiplicative depth. However, the server is still able to make the intersection indirectly depend on the set $Y \setminus X$, which is modeled as a leakage circuit $\mathsf{leakage}(\cdot)$ in their ideal functionality for security *with leakage.*

Nevertheless, this issue does not apply to our scheme, and we discuss how to handle malicious servers. In particular, we discuss how to allow multiple users by allowing reuse of the pre-processing phase across clients; and then discuss the difficulty of handling an arbitrary size of Y.

Reusing pre-processing phase. We start by discussing how to reuse the pre-processing phase with multiple clients for the $|Y| = 1$ case. To resolve the aforementioned inconsistency issues, we can make the following modifications to the protocol in fig. 2. In the pre-processing phase, the server commits to the database T and gives a proof of knowledge (PoK) that hint is correctly computed on T. Additionally, the server provides g^{k_S} along with a PoK for k_S. In the online phase, the server provides a PoK for k_S when generating the OPRF response, as well as a PoK for the committed T when generating the PIR response. Again, these changes do not affect the security against corrupted clients. Note that, however, this would greatly affect the performance. For example, proving that the PIR response is correctly computed can be several times slower than generating the response itself.

We sketch a security proof for this modified protocol against a malicious server. First, the simulator is able to extract both k_S and the database T in the pre-processing phase from the PoKs provided by the server. Since the simulator also keeps track of all the queries to H_1 and H_2, it can extract the set X using a similar approach as in [JL10]. The simulator then sends the extracted set X to the ideal functionality. In the online phase, the two PoKs provided by the server ensure that the PIR and OPRF responses are consistent with the X committed (extracted) in the pre-processing phase. This consistency guarantees that the responses align with the set X sent to the ideal functionality, thus concluding the proof.

Handling arbitrary $|Y|$. The main challenge in handling multiple elements in a client's set comes from Cuckoo hashing. Specifically, the server is supposed to put each element $x_i \in X$ into three hash bins. However, if the malicious server decides to put it into only one of the three bins, then it becomes unclear whether the simulator should include x_i in the set sent to the ideal functionality. This is because whether the same element x_i will be put into that same bin on the client's side depends on the other elements in the client's set. Thus, achieving a simulation-based proof seems challenging unless we incorporate a more sophisticated proof of correctness for Cuckoo hashing.

5 Experimental Results

We implement our single-element PSI protocol in fig. 2 and multi-element PSI protocol in fig. 3 in a C++ library available at `doi:10.5281/zenodo.15131756`. We use the SimplePIR [HHCG+23] implementation in a Go library directly and optimize upon it using the techniques we have mentioned in section 3.2. All benchmarks are running on an Amazon AWS `c5.metal` instance with Intel Xeon Platinum 8275L CPU with 96 virtual cores and 192 GB of RAM.

5.1 Parameter Setting

For single-element PSI, we ran benchmarks for $|X| = 2^{20}, 2^{22}, 2^{24}, 2^{26}$, and 2^{28} using computational security parameter $\kappa = 128$, statistical security parameter $\lambda = 40$, and $H_3 : \mathbb{G} \times \mathbb{G} \to \{0,1\}^\delta$ output size $\delta = 80$ (to guarantee $\lambda = 40$ given the $|X|$).

Following prior work [CLR17, ACLS18], we used experimental analyses to choose the number of hashes and bins for our Cuckoo hashing based multi-element protocol in fig. 3. Our experiments found, for $|Y| = 16$ and only three hashes, $\sim$46 bins were required to run 2^{20} trials without error ($m \approx 2.875|Y|$) and $\sim$105 bins were required for $|Y| = 64$ ($m \approx 1.64|Y|$). Using four hashes and $m = 1.5|Y|$, both 16 and 64 are able to run 2^{20} trials without error, so these were chosen as the increase in the server set size was more desirable than increasing the number of PIR instances. Using these, we ran benchmarks for $|X| = 2^{26}$ with $|Y| = 16$ and 64.

In all benchmarks, τ was chosen to strike a balance between the offline and online phases (note the size estimation discussed in section 3.2). We choose the error bound according to our error analysis in section 3.2, and choose other LWE parameters according to [APS15] to guarantee 128-bit computational security.

5.2 Benchmark Comparison

Single-element PSI. In table 1, we compare the main efficiency metrics with APSI [CMdG+21], the state-of-the-art unbalanced PSI protocol with sublinear client storage (zero for APSI). We use green to highlight the efficiency metrics at which our protocol is better and red to indicate the ones at which our protocol is worse.

Table 1: Efficiency comparison with APSI [CMdG+21] for our single-element PSI in fig. 2. D is the number of threads. X is the server S set. C is the client. For $|X| = 2^{28}, 2^{24}, 2^{20}$, we use the APSI default parameters. For $|X| = 2^{26}, 2^{22}$, since APSI does not provide an interface to choose the optimal parameters, we tested different parameters they provide and chose the optimal one. The server online time measures the computational time taken by the server to run the PSI protocol when the client has a single element

$\|X\|$	Server Offline (s)				Server Online (s)			Communication (MB)					
	$D=1$		$D=32$		$D=1$		$D=32$	Offline		$\mathsf{C} \to \mathsf{S}$		$\mathsf{S} \to \mathsf{C}$	
	Ours	APSI	Ours	APSI	Our	APSI	APSI	Ours	APSI	Ours	APSI	Ours	APSI
2^{28}	-	-	983	1380	0.56	57	4.0	30.9	0	1.05	2.58	0.028	1.8
2^{26}	-	-	224	335	0.15	18	3.3	15.7	0	0.524	2.58	0.015	0.51
2^{24}	915	435	53.5	59	0.034	6.0	0.71	8.80	0	0.262	1.58	0.080	0.91
2^{22}	229	107	12.6	15	0.009	2.1	0.70	4.62	0	0.131	1.58	0.004	0.25
2^{20}	57.1	23.0	3.27	3.7	0.003	0.18	0.12	2.55	0	0.066	0.745	0.002	0.78

As shown in the table, our online runtime with $D = 1$ is about two orders of magnitude faster than APSI with $D = 1$, and about one magnitude faster than APSI with $D = 32$. Thus, we believe that multi-threading is not needed for our online time for most applications. However, note that our construction can be easily multi-threaded: a SimplePIR query is simply n LWE ciphertexts for some $n = O(\sqrt{|X|})$. Thus, we can simply divide the n LWE ciphertexts into T threads and process them separately. We believe setting $D = 32$ gives us a similar speed up as APSI, if not more. Moreover, our online communication cost is also at least 4x smaller.

On the other hand, our overall offline server computation time is slightly worse than APSI's but is still comparable (mainly due to the performance for $D = 1$). Recall that this is a one-time process and can be reused for all clients, and so this extra cost has relatively small impact. Also note that our multi-threaded offline time outperforms APSI, and this is because our offline phase can be easily multi-threaded without much overhead. However, their offline phase contains computation that is not easily multi-threadable (e.g., large polynomial evaluation).

The only major drawback of our protocol when compared to APSI is offline communication. We require the client to store a hint for the underlying SimplePIR protocol. However, as shown in the table, the hint is relatively small (only $<$ 10x larger than the online communication of APSI), and can be amortized over multiple queries. Moreover, it grows with $O(\sqrt{|X|})$ instead of being linear to $|X|$, and thus grows relatively slowly.

Multi-element PSI. We compare our multi-element protocol with APSI in table 2. We also add a naive use of our single-element protocol in table 1 for a more comprehensive comparison. We use bold texts to highlight the best of the three for a given efficiency metric. It is easy to see that for $|Y| > 1$, our construction has less advantage compared to APSI. However, both of our constructions' online time still greatly outperforms APSI, for the $|Y|$'s we test. Note that when $|Y|$ grows larger, our offline communication may grow too large and becomes impractical.[11] A similar argument applies to our online communication.

Comparing with other schemes. In addition to comparing our benchmarks to the APSI protocol, we also evaluated the works of Davi Resende and Aranha in [RA18] and Kales et al. [KRS+19] as shown in table 3. Asymptotically, the online server time of

[11] Our offline communication consists of multiple SimplePIR hints. The number of hints grows with $|Y|$, while each hint size decreases as $|Y|$ increases, and the former grows faster than the latter decreases.

Table 2: Efficiency comparison with APSI [CMdG+21] for our multi-element PSI and variables are defined in the same way as table 1. "Naive" means we simply repeat single-element described in fig. 2 for $|Y|$ times and "Cuckoo hashing" means that we use the Cuckoo hashing protocol described in fig. 3. We choose $|X| = 2^{26}$, as APSI with $|X| = 2^{28}$ and $|Y| > 1$ does not run on our instance (we conjecture the reason to be out of memory).

$\|X\| = 2^{26}$	$\|Y\|$	Server Offline Time (s) $D = 32$	Offline Comm (MB)	Server Online Time (s) $D = 1$	Server Online Time (s) $D = 32$	$\mathsf{C} \rightarrow \mathsf{S}$	$\mathsf{S} \rightarrow \mathsf{C}$
Ours	16 (Naive)	524	15.7	2.35	0.168	8.38	0.240
	16 (Cuckoo hashing)	542	75.5	**0.553**	**0.064**	12.6	**0.070**
APSI	16	520	**0**	40.5	2.1	**3.39**	2.30
Ours	64 (Naive)	524	15.7	9.55	0.288	33.5	0.960
	64 (Cuckoo hashing)	542	188	**0.928**	**0.059**	25.2	**0.167**
APSI	64	520	**0**	41	2.2	**3.39**	2.30

our construction and APSI are $O(|X|)$ while other works are $O(|Y|)$. The offline client communication cost and storage in other works are $O(|X|)$, while ours is $O(\sqrt{|X|})$ and APSI is $O(1)$. Our protocol demonstrates a better balance, e.g. for $|X| = 2^{26}$ and $|Y| = 1$, our server online time is as low as 0.15s, while the offline communication is only 15.7MB, significantly less than the OPRF-based protocols. It should be noted, as mentioned in [CMdG+21], that the protocol proposed in [RA18] utilized extremely aggressive Cuckoo filter parameters, for its exceptionally high performance during the online phase, resulting in an impractical high false-positive rate of 2^{-13}. We demonstrated that, for small $|Y|$, our protocol is computationally very efficient during the online phase, while also keeping the offline communication low as compared to other OPRF-based protocols. However, when the client's size $|Y|$ grows, we could suffer from high communication costs.

Cost estimation for password breach checkup. As mentioned in the introduction, one essential application of unbalanced PSI is password breach checkup [Ali18, LKLM21]. Essentially, a central server holds a database containing a large amount of leaked passwords due to data breaches. The users themselves have one or more passwords that they want to check whether it is already insecure against this database. Of course, the users do not want to leak their own passwords, and the server does not want to share other leaked passwords with the users performing the checkup.

An unbalanced PSI (especially our extremely unbalanced setting) is perfectly suited for such a setting. As suggested in [TPY+19, ALP+21b], a database may contain 2^{32} passwords, and a user may have one or several passwords to check against such a database. Since the online runtime for prior works remains prohibitively large (e.g., checking 2^{32} passwords can take more than 400 seconds), [ALP+21b] suggests dividing the password into buckets, each with size, say 2^{20} passwords. This leaks extra information as the server learns which bucket the client is checking against.

However, with our construction, such online cost is no longer unaffordable. For a single-core server, it takes only about 4 seconds to check for all the 2^{32} passwords against a single password (extrapolating from table 1). This efficiency comes at the expense of the user needing to store a hint of size ∼240MB. However, this hint can be reused for future checkups, making it a favorable trade-off.

PSI from Other Keyword PIR Constructions. We also estimate the performance of unbalanced PSI by plugging other state-of-the-art keyword PIR schemes into our framework. The estimation with the keyword PIR in [PSY23] is based on their numbers in Fig. 12.

Table 3: Comparisons to prior works. LowMC and ECNR are two protocols described in [KRS+19]. APSI is the one in [CMdG+21]. For $|Y| > 1$, we use the Cuckoo hashing protocol in fig. 3. All protocols are running in a single thread.

			Offline		Online	
$\|X\|$	$\|Y\|$	Protocol	Server Time (s)	Comm (MB)	Server Time (s)	Comm (MB)
2^{20}	1	[RA18]	13.5	3	0.013	< 0.001
		LowMC	5	4.2	0.212	0.059
		ECNR	151	4.2	0.257	0.04
		APSI	23	0	0.18	1.525
		Ours	57	2.55	0.002	0.07
2^{24}	1	[RA18]	217	48	0.013	< 0.001
		LowMC	80.5	67	0.22	0.059
		ECNR	2,403	67	0.256	0.04
		APSI	435	0	6	2.49
		Ours	925	8.8	0.034	0.27
2^{26}	1	[RA18]	870	192	0.013	< 0.001
		LowMC	323	268	0.220	0.059
		ECNR	9,617	268	0.202	0.04
		APSI	4,375	0	18	3.09
		Ours	3,730	15.7	0.15	0.54
	16	[RA18]	870	192	0.013	< 0.001
		LowMC	323	268	0.177	0.406
		ECNR	9,617	268	0.160	0.13
		APSI	4,680	0	40.5	5.69
		Ours	5,257	75.57	0.553	12.5
	64	[RA18]	870	192	0.015	0.002
		LowMC	323	268	0.179	1.51
		ECNR	9,616	268	0.173	0.41
		APSI	4,680	0	41	5.69
		Ours	5,623	188.1	0.928	25.3

For $|X| = 2^{26}, |Y| = 1$, as in table 1, their online server runtime is about $\sim$3 seconds, with online upload cost being $\sim$0.014 MB and online download cost being $\sim$0.021 MB. Compared to ours, the runtime is about 20x slower and the online download cost is slightly larger. However, their online upload cost is a lot smaller and requires no offline storage of the client. Alternatively, they could re-parametrize their keyword PIR to reduce the runtime to $\sim$1.5 seconds, while increasing their download cost to $\sim$0.086 MB. Either way, their construction provides different trade-offs compared to our construction. Similar results hold for other sizes of $|X|$, hence we skip the details of the estimation.

6 Extensions

Remove the offline communication. In the pre-processing phase of the single-element PSI protocol (fig. 2), S sends the pre-processed data (i.e., the hint) to C in Step 5. This requires the client to keep some local storage. While this is usually fine as hint is much smaller than the set X, there may be cases where this needs to be avoided.

Thankfully this is a simple adjustment: one can send the hint during the online phase along with the OPRF results. Since hint now is included in the online communication, it should be as small as possible. To ensure this, simply choose α, β such that $\alpha \cdot (n+1) \cdot \log q + \beta \cdot \log q$ is minimized.

For multi-element PSI, a slightly different approach is required since the knowledge of τ_i for each T_i is necessary to perform a PIR query. C and S can agree on τ_i in advance (e.g., let τ_i being the expected size for each T_i). Then, everything else remains the same as for our single-element PSI protocol.

Reduce the round complexity. Our single-element PSI protocol (fig. 2) has two rounds in the online phase: C first sends OPRF request in Step 1, and then uses the OPRF result to prepare the PIR query in Step 4. Recall that this is because our PIR (by keyword) database is constructed from X' instead of X itself.[12] If S uses X to construct the database instead of X', the client can prepare the OPRF request and PIR query at the same time. However, simply replacing X' with X leaks information. Recall our starting point in section 3, S can send the entire X' to the client, but it is insecure to send X directly to C.

Thus, S needs to construct T using X in the following way. We start with the case where $|Y| = 1$. S and C share a hash function $H : \{0,1\}^\delta \to [\tau]$, where $\tau = N$ (τ can be optimized, discussed below). First, S initializes an empty table T of size τ. Then, S computes $T[H(x)] \leftarrow T[H(x)] \cup F_k(x)$, for all $x \in X$. We can bound that each entry in the table has at most $\gamma = \omega(\log(N))$ elements with overwhelming probability. S thus pads random elements to each entry that has $< \gamma$ elements and randomly permutes each entry. Lastly, S uses T as the PIR database. To query for element y, C queries entry $H(y)$ without needing $F_k(y)$. Thus, the PIR query can be prepared together with the OPRF query. After receiving $T[H(y)], F_k(y)$ in response, C simply checks whether $F_k(y) \in T[H(y)]$.

However, one major issue is that now T is of size $\tau \cdot \gamma \geq N$ for some fixed τ, and the padded elements are random instead of zeros. Thus, the computation, instead of being $O(N)$, becomes $O(\tau \cdot \gamma)$. As $\tau = N$, the cost is $\tau \cdot \gamma = \omega(N \log(N))$. Moreover, recall that the PIR scheme needs to download one entry at a time, which means that the download cost is $O(\gamma) = \omega(\log(N))$. In contrast, in the original construction, γ is chosen dynamically after hashing which is much smaller with high probability.

Therefore, the trade-off is worse computation and communication complexity for a better round complexity. For $|Y| \geq 1$, again, C and S need to agree on τ_i for each T_i in advance. Everything else follows the exact same way.

Tuning τ and γ. As mentioned, τ can be further tuned in the alternative above. Recall that the smaller τ is, the smaller $\tau \cdot \gamma$ gets. Thus, instead of setting $\tau = N$, we can set $\tau < N$. Let Z be the random variable representing the size of a randomly populated entry in T with τ entries. Then use Chernoff bound we have $\upsilon \leftarrow \gamma/\mu - 1 \geq 0$, we have $\Pr[Z > (1+\upsilon)\mu] \leq (\frac{e^\upsilon}{(1+\upsilon)^{1+\upsilon}})^\mu$, where $\mu = N/\tau$. Then, we set γ to be the smallest integer such that $\Pr[Z > (1+\upsilon)\mu] \leq 2^{-\lambda}$. One can then reduce the computation and communication costs by fine-tuning τ and γ.

Extension to labeled PSI. Labeled PSI introduced in [CHLR18] does not directly return $X \cap Y$, but returns the payloads attached with $X \cap Y$. In other words, for each $x_i \in X$, there is a payload p_i associated with it. For each x_i that is also in Y, output p_i to the client C. Our construction can be extended to labeled PSI in a straightforward way. Instead of returning $F_k(x_i)$ during the PIR query, the server returns $(F_k(x_i), p_i \oplus F_{k'}(x_i))$ where $F_{k'}(\cdot)$ is another PRF. The OPRF query, therefore, outputs both $F_k(y_i)$ and $F_{k'}(y_i)$. C first check whether $F_k(y_i)$ is in the decoded PIR answer, and then use $F_{k'}(y_i)$ to decode for the payloads. With similar analysis in section A, we achieve the functionality of labeled PSI.

Using doubly efficient PIR. Since PIR to PSI construction is generic, one can replace SimplePIR with arbitrary PIR constructions. A recent work [LMW23] shows that with $O(|X|^{1+o(1)})$ server offline computation and storage, the client can retrieve an entry with $\mathsf{polylog}(|X|)$ time and communication. However, as mentioned, this work is not yet practical, and thus we use SimplePIR in our concrete instantiation.

[12] Recall that $X' := \{F_k(x) | x \in X\}$, and the database T has entry $T[\ell] := \{x' | x' \in X' \wedge H(x') = \ell\}$.

References

[ABFK16] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *PoPETs*, 2016(2):155–174, April 2016.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018. doi:10.1109/SP.2018.00062.

[Ali18] Junade Ali. Validating leaked passwords with k-anonymity, 2018. https://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/.

[ALP+21a] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1811–1828. USENIX Association, August 2021.

[ALP+21b] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–Computation trade-offs in PIR. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1811–1828. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/ali.

[APP21] Password Monitoring – Apple Platform Security. https://support.apple.com/en-al/guide/security/sec78e79fc3b/web, 2021.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. URL: https://doi.org/10.1515/jmc-2015-0016 [cited 2023-06-06], doi:doi:10.1515/jmc-2015-0016.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 55–73. Springer, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_4.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 662–693. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70503-3_22.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 498–507. ACM Press, October 2007. doi:10.1145/1315245.1315307.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011. doi:10.1109/FOCS.2011.12.

[CCF+20] Justin Chan, Landon P. Cox, Dean P. Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham M. Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob E. Sunshine, and Stefano Tessaro. PACT: privacy-sensitive protocols and mechanisms for mobile contact tracing. *IEEE Data Eng. Bull.*, 2020.

[CD24] Sofía Celi and Alex Davidson. Call me by my name: Simple, practical private information retrieval for keyword queries. CCS'24, 2024. URL: https://eprint.iacr.org/2024/092, doi:10.1145/3658644.3670271.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995. doi:10.1109/SFCS.1995.492461.

[CGN98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Cryptology ePrint Archive, Report 1998/003, 1998. https://eprint.iacr.org/1998/003.

[CHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 3–33. Springer, Heidelberg, May / June 2022. doi:10.1007/978-3-031-07085-3_1.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1223–1237. ACM Press, October 2018. doi:10.1145/3243734.3243836.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 694–726. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70503-3_23.

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 44–75. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_3.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017. doi:10.1145/3133956.3134061.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56877-1_2.

[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 Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1135–1150. ACM Press, November 2021. doi:10.1145/3460120.3484760.

[CNCG+23] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. Authenticated private information retrieval. Usenix Security'23, 2023. URL: https://eprint.iacr.org/2023/297.

[DC14] Changyu Dong and Liqun Chen. A fast single server private information retrieval protocol with low communication cost. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part I*, volume 8712 of *LNCS*, pages 380–399. Springer, Heidelberg, September 2014. doi:10.1007/978-3-319-11203-9_22.

[dCL24] Leo de Castro and Keewoo Lee. VeriSimplePIR: Verifiability in SimplePIR at no online cost for honest servers. Usenix Security'24, 2024. URL: https://eprint.iacr.org/2024/341.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_24.

[DPC23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, scalable, single-server private information retrieval. *PoPETs*, 2023(1):365–383, January 2023. doi:10.56553/popets-2023-0022.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *PoPETs*, 2018(4):159–178, October 2018. doi:10.1515/popets-2018-0037.

[FAKM14] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 75–88, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2674005.2674994.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005. doi:10.1007/978-3-540-30576-7_17.

[FLLP24] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. Thorpir: Single server pir via homomorphic thorp shuffles. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1448–1462, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3658644.3690326.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. URL: https://eprint.iacr.org/2012/144.

[GH19] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 438–464. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-36033-7_17.

[GLM16] Matthew Green, Watson Ladd, and Ian Miers. A protocol for privately reporting ad impressions at scale. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1591–1601. ACM Press, October 2016. doi:10.1145/2976749.2978407.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84245-1_14.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 78–86. ACM, 1999. doi:10.1145/336992.337012.

[HHCG+23] 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. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 155–175. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_10.

[HSW23] Laura Hetz, Thomas Schneider, and Christian Weinert. Scaling mobile private contact discovery tobillions of users. In *Computer Security – ESORICS 2023: 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25–29, 2023, Proceedings, Part I*, page 455–476, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-50594-2_23.

[HWS+21] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. All the numbers are US: Large-scale abuse of contact discovery in mobile messengers. In *NDSS 2021*. The Internet Society, February 2021.

[IKN+20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 370–389. IEEE, 2020. doi:10.1109/EuroSP48549.2020.00031.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 418–435. Springer, Heidelberg, September 2010. doi:10.1007/978-3-642-15317-4_26.

[KC21] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 875–892. USENIX Association, August 2021.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016. doi:10.1145/2976749.2978381.

[KLL+15] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. *PoPETs*, 2015(2):222–243, April 2015. doi:10.1515/popets-2015-0016.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *PoPETs*, 2017(4):177–197, October 2017. doi:10.1515/popets-2017-0044.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997. doi:10.1109/SFCS.1997.646125.

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

[Lin16] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. *Electron. Colloquium Comput. Complex.*, TR17, 2016. URL: https://api.semanticscholar.org/CorpusID:3331839, doi:10.1007/978-3-319-57048-8_6.

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

[LLM22] Chengyu Lin, Zeyu Liu, and Tal Malkin. XSPIR: Efficient symmetrically private information retrieval from ring-LWE. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part I*, volume 13554 of *LNCS*, pages 217–236. Springer, Heidelberg, September 2022. doi:10.1007/978-3-031-17140-6_11.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 130–160. Springer, Heidelberg, December 2022. doi:10.1007/978-3-031-22966-4_5.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 595–608, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585175.

[LP22] Arthur Lazzaretti and Charalampos Papamanthou. Near-optimal private information retrieval with preprocessing. Cryptology ePrint Archive, Paper 2022/830, 2022. URL: https://eprint.iacr.org/2022/830, doi:10.1007/978-3-031-48618-0_14.

[LP23] Arthur Lazzaretti and Charalampos Papamanthou. Treepir: Sublinear-time and polylog-bandwidth private information retrieval from ddh. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 284–314, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-38545-2_10.

[LPR+20] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from PIR with default. Cryptology ePrint Archive, Paper 2020/1011, 2020. URL: https://eprint.iacr.org/2020/1011, doi:10.1007/978-3-030-92075-3_21.

[Mar14] Moxie Marlinspike. The difficulty of private contact discovery. A company blog post (2014), 2014. https://signal.org/blog/contact-discovery/.

[MCR21] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2292–2306. ACM Press, November 2021. doi:10.1145/3460120.3485381.

[MPR+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56877-1_1.

[MW22] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947. IEEE Computer Society Press, May 2022. doi:10.1109/SP46214.2022.9833700.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997. doi:10.1109/SFCS.1997.646134.

[OPPW23] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. Cryptology ePrint Archive, Paper 2023/1510, 2023. Financial Cryptography and Data Security 2024. URL: https://eprint.iacr.org/2023/1510, doi:10.1007/978-3-031-78679-2_14.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 2004. doi:10.1016/j.jalgor.2003.12.002.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26954-8_13.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45724-2_25.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_15.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17659-4_5.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78372-7_5.

[PSY23] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be dense: Efficient keyword PIR for sparse databases. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3853–3870, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/patel.

[PT20] Jeongeun Park and Mehdi Tibouchi. SHECS-PIR: Somewhat homomorphic encryption-based compact and scalable private information retrieval. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 86–106. Springer, Heidelberg, September 2020. doi:10.1007/978-3-030-59013-0_5.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008. doi:10.1007/978-3-540-85174-5_31.

[RA18] Amanda C. Davi Resende and Diego F. Aranha. Faster unbalanced private set intersection. In Sarah Meiklejohn and Kazue Sako, editors, *FC 2018*, volume 10957 of *LNCS*, pages 203–221. Springer, Heidelberg, February / March 2018. doi:10.1007/978-3-662-58387-6_11.

[RR17] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1229–1242. ACM Press, October / November 2017. doi:10.1145/3133956.3134044.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Heidelberg, October 2021. doi:10.1007/978-3-030-77886-6_31.

[SACM21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 641–669, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84259-8_22.

[TKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy preserving error resilient dna searching through oblivious automata. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 519–528. ACM Press, October 2007. doi:10.1145/1315245.1315309.

[TPY+19] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 1555–1571, USA, 2019. USENIX Association.

[TSS+20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 2020.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. doi:10.1109/SFCS.1986.25.

[ZLTS23] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 395–425. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30545-0_14.

[ZPSZ23] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server PIR with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452, 2023. URL: https://eprint.iacr.org/2023/452.

A Security Proofs

A.1 Proof of theorem 1

We construct a simulator Sim that simulates C's view as follows. Sim is given C's input set Y and the output $I = X \cap Y$ (but no information about $X \setminus Y$). Sim runs the honest C's protocol to generate its view, playing the role of an honest server S with the following exceptions:

1. For each element $t \in Y$, sample a random $v_t \xleftarrow{\$} \mathbb{G}$ as $H_1(t)^{k_\mathsf{S}}$.
2. In the pre-processing phase, Sim follows the protocol execution of an honest S except that it skips Step 1 in sampling k_S. When computing u_i's in Step 3a, Sim computes $|I|$ of them by $\{H_2(H_1(t), v_t)\}_{t \in I}$, samples the remaining $(N - |I|)$ of them randomly from $\{0,1\}^\delta$, and then randomly shuffles all these u_i's.[13]
3. In the online phase, upon receiving $\{z_i = H_1(y_i)^{k_\mathsf{C}}\}_{i \in [M]}$ in Step 1, Sim computes $\{z'_i := (v_{y_i})^{k_\mathsf{C}}\}_{i \in [M]}$ and sends it back to C.
4. Sim follows the rest of the protocol execution honestly and outputs C's view.

Since the ideal-world server gets $\perp$ from the ideal functionality and the real-world server also outputs $\perp$, we only need to argue that C's view in the real-world protocol execution with the honest server S is indistinguishable from its view when interacting with Sim in the ideal world. We sketch a hybrid argument below.

$\mathcal{H}_0$ C's view in the real world.

[13] If we assume each entry of the table is an unordered or randomly ordered set, then it would be unnecessary to shuffle the elements.

$\mathcal{H}_1$ Same as $\mathcal{H}_0$ except that for each element $t \in X \cup Y$, sample a random $v_t \xleftarrow{\$} \mathbb{G}$ as $H_1(t)^{k_{\mathsf{S}}}$. In particular, in the pre-processing phase Step 3a, u_i is computed as $u_i := H_2(H_1(x_i), v_{x_i})$; in the online phase Step 2, z'_i is computed as $z'_i := (v_{y_i})^{k_{\mathsf{C}}}$. This hybrid is computationally indistinguishable from $\mathcal{H}_0$ because H_1 is modeled as a random oracle and that DDH holds in $\mathbb{G}$. In more detail, we can construct a sequence of hybrids from $\mathcal{H}_0$ to $\mathcal{H}_1$ to change from $H_1(t)^{k_{\mathsf{S}}}$ to v_t one by one. To argue indistinguishability between every pair of intermediate consecutive hybrids, we can construct a reduction Red to break the DDH assumption of $\mathbb{G}$. On receiving a DDH challenge tuple (g_1, g_2, g_3), Red sets $H_1(t) := g_1, g^{k_{\mathsf{S}}} := g_2$, and $H_1(t)^{k_{\mathsf{S}}} := g_3$. Red builds a table $\mathcal{T}_1 = \{(x, \phi)\}$ to answer all the hash queries to H_1, where (t, g_1) is first added to $\mathcal{T}_1$. To answer an H_1 query on x that has never been queried before, Red picks a random $\alpha_x \xleftarrow{\$} \mathbb{Z}_q$, adds an entry $(x, \phi = g^{\alpha_x})$ to $\mathcal{T}_1$, and returns ϕ as $H_1(x)$. Whenever Red needs to compute $H_1(x)^{k_{\mathsf{S}}}$ for some $x \neq t$, it computes it as $(g_2)^{\alpha_x}$. Distinguishing between $H_1(t)^{k_{\mathsf{S}}}$ and v_t directly corresponds to distinguishing a DDH tuple from a random tuple for (g_1, g_2, g_3).

$\mathcal{H}_2$ Same as $\mathcal{H}_1$ except that in the pre-processing phase Step 3a, for each $x_i \notin I$, we replace its u_i by a random string from $\{0,1\}^{\delta}$. We can again construct a sequence of hybrids from $\mathcal{H}_1$ to $\mathcal{H}_2$ to change the elements one by one from $u_i := H_2(H_1(x_i), v_{x_i})$ to $u_i \xleftarrow{\$} \{0,1\}^{\delta}$. The only way to distinguish between the two intermediate consecutive hybrids is if C makes an H_2 query on $(H_1(x_i), v_{x_i})$. Let q_2 be the number of queries that C makes to H_2. Since v_{x_i} is randomly sampled from $\mathbb{G}$, the probability that C makes such a query is at most q_2/q, which is negligible. This hybrid is exactly the output view of Sim.

A.2 Proof of theorem 2

We construct a simulator Sim that interacts with the malicious client C^* as follows and outputs whatever C^* outputs in the end.

1. Sim builds two tables $\mathcal{T}_1 = \{(x, \phi)\}$ and $\mathcal{T}_2 = \{((h, t), \psi)\}$ to answer the hash queries to H_1 and H_2 respectively. To answer an H_1 query on x that has never been queried before, Sim picks a random $\phi \xleftarrow{\$} \mathbb{G}$, adds an entry (x, ϕ) to $\mathcal{T}_1$, and returns ϕ as $H_1(x)$. To answer an H_2 query on the pair (h, t) which has never been queried before, Sim samples a random $\psi \xleftarrow{\$} \{0,1\}^{\delta}$, adds an entry $((h, t), \psi)$ to $\mathcal{T}_2$, and returns ψ as $H_2(h, t)$. For the queries that have been queried before, maintain consistency and return whatever was returned already.

2. In the pre-processing phase, Sim follows the protocol execution of an honest server with the following exceptions: skip Step 1 in sampling k_{S}, and randomly sample each $u_i \xleftarrow{\$} \{0,1\}^{\delta}$ in Step 3a. Let $U := \{u_i\}_{i \in [N]}$. Sim also answers requests to H_1, H_2 as in item 1 above.

3. In the online phase, upon receiving $\{z_i\}_{i \in [M]}$ in Step 1, Sim first samples $k_{\mathsf{S}} \xleftarrow{\$} \mathbb{Z}_q$ and then sends $\{z'_i\}_{i \in [M]}$ back to C^* where $z'_i := z_i^{k_{\mathsf{S}}}$ for all $i \in [M]$.

4. Sim checks if $\exists ((h, t), \cdot) \in \mathcal{T}_2$ such that $t = h^{k_{\mathsf{S}}}$. If so, abort_1.

5. After sending $\{z'_i\}_{i \in [M]}$, Sim initializes empty sets $Z, V := \emptyset$ and answers H_1, H_2 queries as follows:

   - For the queries that have been queried before, maintain consistency and return whatever was returned already.

- For each query x to H_1 that has not been queried before, randomly sample $\phi \xleftarrow{\$} \mathbb{G}$ and then check if $\exists((h,t),\cdot) \in \mathcal{T}_2$ such that $h = \phi$ and $t = h^{k_\mathsf{S}}$. If so, abort_2; otherwise, add (x,ϕ) to $\mathcal{T}_1$ and return ϕ as $H_1(x)$.
- For each query (h,t) to H_2 that is not yet queried, check if $\exists(x,\phi) \in \mathcal{T}_1$ such that $h = \phi$ and $t = h^{k_\mathsf{S}}$. If not, answer the query as in item 1 above. Otherwise, add x to Z. If $|Z| > M$, abort_3. Otherwise, send x to the ideal functionality. If the functionality returns yes, then Sim picks a random $u \xleftarrow{\$} U \setminus V$, adds u to V, adds $((h,t),u)$ to $\mathcal{T}_2$, and returns u as $H_2(h,t)$. Otherwise, as the ideal functionality returns no, Sim samples a random $\psi \xleftarrow{\$} \{0,1\}^\delta$, adds $((h,t),\psi)$ to $\mathcal{T}_2$, and returns ψ as $H_2(h,t)$.

6. Sim answers the PIR queries following the protocol execution of an honest server. Sim also answers queries to H_1, H_2 as in item 5 above.

Since the ideal-world server gets $\perp$ from the ideal functionality and the real-world server also outputs $\perp$, we only need to argue that C^*'s view in the real-world protocol execution with the honest server S is indistinguishable from its view when interacting with Sim in the ideal world. The only difference between C^*'s views in the real world and ideal world is how H_1 and H_2 queries are answered. Since H_1, H_2 are modeled as random oracles, it is easy to see that the two views only differ when Sim aborts. Next, we argue the three aborts happen with negligible probability. Let q_1, q_2 be the number of queries that C^* makes to H_1, H_2 respectively.

- abort_1 happens if C^* queries H_2 with (h, h^{k_S}) before receiving any information about k_S. Since k_S is sampled randomly from $\mathbb{Z}_q$, this happens with probability at most q_2/q, which is negligible.
- abort_2 happens if C^* queries $H_1(x)$ which returns ϕ, while the entry $((\phi,\phi^{k_\mathsf{S}}),\cdot)$ already exists in $\mathcal{T}_2$. In other words, C^* makes a query $(\phi,\phi^{k_\mathsf{S}})$ for H_2 before knowing that $H_1(x) = \phi$. This happens with probability at most $q_1 \cdot q_2/q$, which is negligible.
- if abort_3 happens, we can construct a reduction Red to break the (q_1, M)-OMGDH assumption with challenges $(g_1, \ldots, g_{q_1})$ as the challenge instance. Red works in the same way as Sim with the following exceptions: (1) Red uses $(g_1, \ldots, g_{q_1})$ to reply to the H_1 queries from C^*; (2) upon receiving $\{z_i\}_{i\in[M]}$ in Step 2 of the online phase, Red queries the $(\cdot)^k$ oracle and gets back $\{z'_i\}_{i\in[M]}$, which it sends back to C^*; (3) whenever Sim needs to check if $t = h^{k_\mathsf{S}}$ for some (h,t), Red calls the DDH oracle to decide whether $t = h^k$. Finally, Red outputs Z if $|Z| > M$. Note that Red decides to add an x to Z only if C^* makes an H_2 query on (h,t) for which $\exists(x,\phi) \in \mathcal{T}_1$ such that $h = \phi$ and $t = h^k$. Hence, the probability that abort_3 happens is bounded by the probability to break the (q_1, M)-OMGDH assumption.

The committing property. We showed that our protocol is secure against a malicious client for the adaptive PSI functionality. Nevertheless, we can also show that a malicious client C^* cannot change its input set after sending $\{z_i\}_{i\in[M]}$ in the online phase Step 1. In other words, even though the protocol achieves only an adaptive version of the PSI functionality, the adversary C^* is committed to all its inputs in Step 1, and hence it is not clear what advantage C^* could obtain by not making all these queries in later steps, in which case the adaptive functionality is equivalent to the standard functionality. The proof is more involved and we refer the reader to [JL10, Thm 2] for details.

A.3 Proof of theorem 3

We first prove the correctness of the protocol. In the online phase, for each $y_i \in Y$, its corresponding z''_i is computed as $z''_i = H_2(H_1(y_i), ((H_1(y_1)^{k_\mathsf{C}})^{k_\mathsf{S}})^{k_\mathsf{C}^{-1}}) = H_2(H_1(y_i), (H_1(y_1))^{k_\mathsf{S}})$.

Given the fact that H_2 is modeled as a random oracle and the parameter choice of δ, collisions of z_i'' happen with negligible probability. Furthermore, the parameter choice of m guarantees that Cuckoo hashing fails with negligible probability in Step 3c of the online phase. y_i is then put into the hash bin Y_j for some $j \in \{h_1(z_i''), h_2(z_i''), h_3(z_i'')\}$. If $y_i \in X$, then z_i'' is put into all three bins of X_k for each $k \in \{h_1(z_i''), h_2(z_i''), h_3(z_i'')\}$ in the pre-processing phase Step 3b, hence $z_i'' \in X_j$, and z_i'' is put into the $H_{3,j}(z_i'')$-th entry of the table T_j in pre-processing Step 4c. If $y_i \notin X$, then with overwhelming probability $z_i'' \neq H_2(H_1(x), H_1(x)^{k_\mathsf{S}})$ for any $x \in X$ given the fact that H_2 is modeled as a random oracle and the parameter choice of δ, thus z_i'' does not appear in X_j (and hence not in T_j either). In other words, $z_i'' \in T_j[H_{3,j}(z_i'')]$ iff $y_i \in X$ with all but negligible probability. Finally, $z_i'' \in R_j$ in Step 5a iff $z_i'' \in T_j[H_{3,j}(z_i'')]$, which follows from the correctness of the underlying PIR protocol. This concludes the correctness proof.

To prove privacy, we construct a simulator Sim that simulates S's view as follows. Sim runs the honest S's protocol to generate its view, playing the role of an honest client C with the following exceptions. In the online phase Step 1, send randomly sampled group elements to S. In Stp 3e, send PIR queries for the first entry of each database. S's view in the real-world protocol execution is indistinguishable from its view when interacting with Sim in the ideal world. We sketch a hybrid argument below.

$\mathcal{H}_0$ S's view in the real world.

$\mathcal{H}_1$ Same as $\mathcal{H}_0$ except that in the online phase Stp 3e, send PIR queries for the first entry of each database. This is indistinguishable from $\mathcal{H}_0$ because of the security of the underlying PIR protocol.

$\mathcal{H}_2$ Same as $\mathcal{H}_1$ except that for each element $t \in Y$, sample a random $v_t \xleftarrow{\$} \mathbb{G}$ as $H_1(t)^{k_\mathsf{C}}$. This hybrid is exactly the output view of Sim. $\mathcal{H}_2$ is computationally indistinguishable from $\mathcal{H}_1$ because H_1 is modeled as a random oracle and that DDH holds in $\mathbb{G}$. In more detail, we can construct a sequence of hybrids from $\mathcal{H}_1$ to $\mathcal{H}_2$ to change from $H_1(t)^{k_\mathsf{C}}$ to v_t one by one. To argue indistinguishability between every pair of intermediate consecutive hybrids, we can construct a reduction Red to break the DDH assumption of $\mathbb{G}$. On receiving a DDH challenge tuple (g_1, g_2, g_3), Red sets $H_1(t) := g_1, g^{k_\mathsf{C}} := g_2$, and $H_1(t)^{k_\mathsf{C}} := g_3$. Red builds a table $\mathcal{T}_1 = \{(x, \phi)\}$ to answer all the hash queries to H_1, where (t, g_1) is first added to $\mathcal{T}_1$. To answer an H_1 query on x that has never been queried before, Red picks a random $\alpha_x \xleftarrow{\$} \mathbb{Z}_q$, adds an entry $(x, \phi = g^{\alpha_x})$ to $\mathcal{T}_1$, and returns ϕ as $H_1(x)$. Whenever Red needs to compute $H_1(y)^{k_\mathsf{C}}$ for some $y \neq t$, it computes it as $(g_2)^{\alpha_y}$. Distinguishing between $H_1(t)^{k_\mathsf{C}}$ and v_t directly corresponds to distinguishing a DDH tuple from a random tuple for (g_1, g_2, g_3).