Private Information Retrieval with Guaranteed Output

ABSTRACT

Private Information Retrieval (PIR) allows a client to query a server holding a database D (or a set of non-colluding servers holding the same database) for the value of the database at index i (i.e., $D[i]$). Security warrants that the servers must not learn anything about i. Much research has focused on the concrete efficiency of PIR schemes; however, no scheme *guarantees* that the client will indeed receive $D[i]$ when some threshold number of servers can be malicious.

In this work, we construct the first 3-server PIR scheme where the client is guaranteed to obtain the output even in the presence of 1 malicious server. Our protocol is concretely efficient and enjoys several attractive properties: 1) The client performs no cryptographic operations; 2) All cryptographic operations performed by the servers can be done in an offline pre-processing phase without the involvement of the client; 3) It is based on symmetric key cryptographic primitives. We also demonstrate the practicality of our protocol with a prototype implementation. On commodity hardware, in the online phase, a query to a database with 2^{24} elements of size 8 B each completes in 152 ms with approximately 4.7 KB of communication.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; *Management and querying of encrypted data.*

KEYWORDS

Private Information Retrieval, Guaranteed Output Delivery, Distributed Point Function

ACM Reference Format:

. 2018. Private Information Retrieval with Guaranteed Output. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

One of the most widely studied problems in cryptography is that of Private Information Retrieval (PIR) [22]. In this problem, there is a server $\mathcal{S}$ holding a database D comprising of N elements and a client C with an index i (ranging from 0 to $N-1$). The goal is for the client to learn the i^{th} element of D, i.e. $D[i]$, while ensuring that $\mathcal{S}$ learns nothing about i. A trivial way of solving this problem is for C to download the entire database irrespective of the index being read. However, this solution has a communication complexity of N. Several works have studied the communication and computational complexity of this problem [2, 5, 11, 31, 50–52, 56] and many works have also shown the importance of PIR both in theory [4, 9, 10, 24, 41, 43, 44] and in practice [6, 39, 40, 56]. For the purpose of obtaining efficient solutions, a long line of work has also considered PIR in the multi-server setting [4, 9, 13, 31, 32], where $m \geq 2$ servers $\mathcal{S}_0, \cdots, \mathcal{S}_{m-1}$ hold the copy of the same database and it is assumed that the servers do not collude with each other to learn i.

PIR protocols can be used to achieve privacy in various applications like media streaming [39], ad services [37], location based services [33], online presence detection [12] and anonymous messaging [7, 48, 53] and as such designing efficient and robust PIR protocols is important.

Despite much work in the area, no work has addressed the case where the server(s) could be malicious and the goal of malicious servers is to learn information about i or prevent the client from learning $D[i]$. This notion of security (studied in the context of secure multi-party computation (MPC) in cryptography) is known as *guaranteed output delivery* (GOD) where honest participants are guaranteed to obtain the output of the computation despite the malicious behaviour of (a threshold number of) other participants. In the context of PIR, we will require that an honest client C learns the correct value of $D[i]$ despite the malicious behaviour of $t < m$ servers. Guaranteed output delivery in the presence of malicious entities is of great practical relevance, especially in the client-server models. In multi-server PIR solutions, a malicious server may result in *denial-of-service* attack for all honest clients, or can even choose when to deny service, rendering the application meaningless.

Number of servers. When such stringent security is required, then it can be easily seen that such a setting requires the presence of at least $m = 3$ servers even when at most 1 server may be malicious. To see this, observe that in the case of a single corrupted server, the server can choose not to respond to a client's query. In case where there are only 2 servers, then the malicious server can simply follow the protocol honestly with a corrupted copy of the database D' and the client cannot tell apart the actions of the honest server running the protocol with D and the malicious server running the protocol with D'. Hence, we focus on the minimal setting of 3 servers $\mathcal{S}_0, \mathcal{S}_1$, and $\mathcal{S}_2$ who hold identical copies of the database D where at most one server may be corrupted.

Potential solution and drawback. A potential candidate solution to the above problem is to use a single-server PIR solution that provides privacy against malicious servers. All 2-round single server PIR schemes, e.g., [6, 29, 52, 56], satisfy this requirement of privacy against malicious servers. Now, to build a solution with GOD security, the client can run the single-server PIR protocol with each of the 3 servers separately, obtain $D[i]^0, D[i]^1$, and $D[i]^2$ from $\mathcal{S}_0, \mathcal{S}_1$, and $\mathcal{S}_2$ respectively and take the majority of these 3 values. Since 2 servers are guaranteed to be honest, the majority value will indeed be correct, while since the PIR solution is private against malicious servers, the malicious server will also not learn anything about i. However, single-server PIR schemes are computationally expensive for both the server as well as the client. The servers need to perform many public-key operations and the client is not lightweight. In particular, the client also needs to perform heavy public-key cryptographic operations.

Goal. Our goal is to construct a 3-server PIR solution that has GOD security, where the client performs *no cryptographic operations*. In other words, we want the client to be lightweight and hence, as efficient as possible.

Need for server-to-server interaction. First, observe that even if we only want the weaker security against semi-honest servers, but require low (poly-logarithmic in N) communication complexity and the client to not perform any cryptographic operations, then we would need the servers to interact with each other (or channel these messages through the honest client, thus resulting in a multi-round PIR protocol) unless significant advancements are attained in the field of information-theoretic PIR [61]. This is because any such non-interactive protocol would readily imply an information-theoretic PIR scheme. Hence, any efficient solution for a 3-server PIR with GOD security and non-cryptographic client must allow the servers to interact with each other. Moreover, to achieve total communication that is only poly-logarithmic in the database size, we need to rely on cryptographic assumptions, e.g., one-way functions.

1.1 Our Results

We construct an efficient 3-server PIR protocol with guaranteed output delivery for the client in the presence of one malicious server, in which the client does not perform any cryptographic operations. Our solution relies on the security of one-way functions and is concretely efficient with a total communication of $O(\lambda \log N)$ bits, where λ is the computational security parameter and N is the database size. Further, our solution can be split into an offline pre-processing phase where communication happens only between servers and an online phase where all parties only perform light-weight non-cryptographic operations. That is, all expensive cryptographic operations are pushed to an input-independent offline phase. Moreover, the offline phase only depends on the number of entries in the database and is even independent of the size of each entry. In the offline phase, the total communication is one round of communication of $\approx \lambda \log N$ bits from $\mathcal{S}_2$ to $\mathcal{S}_0$ and $\mathcal{S}_1$ each. Every server performs $\approx N$ pseudo-random generator (PRG) evaluations in the offline phase. In the online phase, our protocol has a total of 3 rounds in the case when the servers are not malicious and a worst-case complexity of 5 when a server behaves maliciously. The total communication between the client and servers is $\approx 11\lambda \log N$.

We build a prototype implementation of our PIR protocol and show it to be concretely efficient (Section 5). On commodity hardware, for a database with 2^{20} elements and 8 bytes database entry, the offline phase of our protocol communicates only 830 bytes with a runtime of 577 ms. In the online phase, the total communication is only 4 KB with a runtime of 11 ms. Of this runtime, the client-side compute only takes 1.1 ms. For database with entries of size 1 KB, the online phase takes 179 ms and 7.9 KB of communication. Furthermore, lowering of bandwidth between the client and servers has negligible impact on performance (due to the low communication of our protocol). Compared to the vanilla solution based on single-server PIR discussed above, our protocols have up to 4893× lower communication and are up to 146× faster (the best single-server PIR schemes either suffer from high communication or latency). Even considering only online time, our protocol communicates up to 1878× lesser data and is up to 37× faster.

1.2 Our Techniques

2-server PIR. Our starting point is a 2-server PIR solution [13] (secure only against semi-honest servers) based on distributed point functions (DPF). A point function $f_{\alpha,\beta}$ is a function that evaluates to 0 everywhere, except at a special point α where it evaluates to β. In a DPF scheme for function $f_{\alpha,\beta}$, a dealer generates keys k_0 and k_1 to be given to 2 parties $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively. $k_b, b \in \{0, 1\}$ individually is guaranteed to hide α and β. $\mathcal{S}_b$ can then locally evaluate the function at any public input point x to obtain arithmetic secret shares of $y_x = f_{\alpha,\beta}(x)$ (i.e., $\mathcal{S}_0$ and $\mathcal{S}_1$ learn random values $z_{x,0}$ and $z_{x,1}$ such that $z_{x,0} + z_{x,1} = z_x$). A DPF scheme can be used to construct a 2-server PIR protocol - the client, with index i, creates a DPF key for $f_{i,1}$ and gives the keys to $\mathcal{S}_0$ and $\mathcal{S}_1$, who evaluate the function on all points $x \in [N]$. Now, both servers can locally take an inner product of the database with the shares they obtain (i.e., S_b computes $z_b = \sum_{j=1}^{N} z_{j,b} \cdot D[j]$). It is easy to see then that the servers hold shares of $D[i]$ (i.e., $z_0 + z_1 = D[i]$). Now, one can add *verifiability* to the above solution (i.e., ensure that the client never accepts an incorrect $D[i]$) using the techniques from [25, 27, 28], that adds an information-theoretic message authentication codes to the DPF outputs. This would give us a 2-server PIR solution with malicious privacy (but no GOD security).

This 2-server PIR has the following overheads: a) Communication between client and servers consists of DPF keys and is $\approx \lambda \log N$ b) Client performs $\approx \log N$ PRG evaluations c) Servers perform $\approx N$ PRG evaluations. Aside from not obtaining GOD security and also requiring the client to perform cryptographic operations, this solution also suffers from the following drawback. It requires the servers to perform $\approx N$ PRG evaluations in the *online* phase (i.e., in the critical PIR query path), since the DPF key is computed and given to the servers by the client.

Our Construction. We now present our 3−server protocol with GOD security in which all expensive PRG operations can be performed by the servers in the offline phase. The first idea is to "outsource" the client computation in the above solution to the server. To ensure that the index remains hidden, we must now instead run a secure computation protocol to emulate the client. The work of Doerner and shelat [30] shows how to construct a 2-party secure distributed keygen protocol to generate DPF keys k_0 and k_1 for the DPF function $f_{\alpha,\beta}$, where α and β are secret-shared among the two parties. An attractive feature of their protocol is that the protocol only makes black-box use of PRGs. Unfortunately, their protocol suffers from two drawbacks: first, it is only semi-honest secure; second, it is highly interactive (specifically, parties interact in $\log N$ instances of secure 2-party computation sequentially).

To obtain GOD security against a malicious server, our idea is to leverage the third server and use the technique of "MPC-in-the-head". At a high level, we will require $\mathcal{S}_2$ to internally execute the semi-honest distributed keygen protocol to generate views of both $\mathcal{S}_0$ and $\mathcal{S}_1$. Similar to the idea of replicated secret sharing, we will then also have $\mathcal{S}_0$ and $\mathcal{S}_1$ execute the same protocol with the same randomness, thus enabling verification of correct computation through a simple comparison of views. However, this would still require $\mathcal{S}_0$ and $\mathcal{S}_1$ to interact over $\log N$ rounds of secure computation.

Our third idea is to observe that $\mathcal{S}_0$ and $\mathcal{S}_1$ need not actually interact in the distributed keygen protocol but instead can rely on $\mathcal{S}_2$ to provide the outputs that need to be computed at every step. However, this introduces a new complexity. How does one ensure that the outputs provided by $\mathcal{S}_2$ are indeed correct? Here is where we make use of the client to verify that the computation has been done correctly. In fact, not only are we able to verify correctness of computation, but we can make use of the client to also identify a trusted party in the event that malicious behaviour was detected. Once a trusted party is identified, the client can simply provide its index to this trusted party[1] which will provide the client with the right value. Note that this is indeed the standard approach to obtaining GOD security [21, 58] and doing so does not break security – indeed, trusted or honest parties are allowed to learn private inputs of other parties, as is done in many prior works that achieve GOD security [15, 16, 42, 45].

Our final idea is an observation (from [17]), that one can run the DPF keygen on a random special point, instead of the index being queried and then "rotate" the database appropriately to obtain the queried value. This enables us to push all cryptographic operations to an offline preprocessing phase thereby obtaining a lightweight online phase.

Putting all this together, offline phase only uses symmetric key cryptography and a single message of outputs from keygen from $\mathcal{S}_2$ to $\mathcal{S}_0, \mathcal{S}1$ each. The online phase involves a lightweight verification by the client ($O(\log N)$ complexity) and a simple inner-product by the servers, thereby obtaining a lightweight online phase.

1.3 Other Related Works

The seminal work of Chor *et al.* [22] introduced the problem of PIR and provided the first information-theoretic solution to the problem in the 2-server setting with communication complexity of $O(N^{\frac{1}{3}})$. The work of Efremenko [32] and Itoh and Suzuki [43] considered PIR in the multi-server (> 2) setting with improved communication complexity. It is known that short locally decodable codes (LDCs) yield efficient PIR schemes [44] and vice-versa. Recently, the work of Dvir and Gopi [31] provided a 2-server solution with communication complexity of $O\left(N^{\sqrt{\log\log N/\log N}}\right)$.

Kushilevitz and Ostrovsky [47] first considered the problem of PIR in the computational setting and provided a single server PIR solution with communication complexity $O(\sqrt{N})$. Since then, much progress has been made [18, 19, 35, 60]. Asymptotically, the best known solution [34] has communication of $O(\log N)$. Concretely efficient solutions to this problem have also been explored [6, 29, 51, 52, 56] based on homomorphic encryption. While these solutions have very low communication overhead, they have extremely large computational cost (as an example, in the state-of-the-art scheme of [29], querying a database with 2^{18} elements requires the server to perform $> 17s$ of computation on commodity hardware).

Given the high overheads of single-server PIR, several works have also explored computationally secure PIR solutions in the multi-server (≥ 2) setting. Here, concretely efficient solutions can be obtained based on one-way functions [13, 36] using the technique of distributed point functions (DPFs) instead of relying on public-key assumptions such as homomorphic encryption. These protocols have $O(\log N)$ communication. In terms of computational overhead, the client performs $O(\log N)$ PRG evaluations, while the server must perform $O(N)$ PRG evaluations. These solutions offer security only against semi-honest servers. Using the technique from [13, 14], one can modify the payload of the DPF, to also obtain security with abort against one malicious server. However, these solutions do not guarantee that the client will learn the output.

A recent line of work has also studied PIR in an offline/online setting [24] requiring the servers to do work proportional to N only in an offline preprocessing phase (while interacting with the client) and construct protocols with sub-linear online computation time. Finally, the very recent work of [49] show how to perform an offline preprocessing that is independent of the client and still enable $O(\log^c N)$ online overhead. A similar work of [38] proposes general multi server PIR that acheives sublinear online time and concrete efficiency for large databases. These works only provide privacy against malicious servers and are only efficient for very large database.

Many works on secure multiparty computation (MPC) consider the case of MPC amongst a few parties with the focus on security with abort [8, 20, 54, 57]. In the honest majority setting, the strongest notion of security in MPC is that of guaranteed output delivery. Honest majority is necessary to achieve GOD [23]. Ishai *et al.* [42] give a general transformation to obtain protocols with guaranteed output delivery in honest majority from semi honest secure protocols. Some of the recent works in privacy preserving machine learning with MPC have also considered the notion of GOD specifically for the client-server setting [26, 45, 46].

2 PRELIMINARIES

Let λ denote the security parameter. We assume familiarity with basic cryptographic notations such as negligible functions and computational indistinguishability. We use $a \leftarrow A$ to denote an element a that is uniformly sampled from a set A.

We use two types of two-out-of-two secret sharing schemes. The first one is an arithmetic secret sharing scheme. Here, the secret to be shared is given by a field element $y \in \mathbb{F}$ (where $\mathbb{F}$ is a finite

[1]If indeed, we wish to provide privacy against a single (even honest) server, we can construct a 4-server PIR protocol achieving GOD security using our 3-server PIR protocol in a black-box manner as follows. We run our 3-server solution with every set of 3 out of the 4 servers independently. From the property of our 3-server protocol, we know that every instance will either succeed with the right output or will abort leading to the client identifying a single honest server. Running our 3-server protocol amongst all 4 sets of 3 servers, ensures that at least one execution has no corrupt parties. Hence, the client can take the output of this execution as the final output (in other words, any execution that does not lead to an abort output leads to the right output and we guarantee that at least one execution will have this property). Furthermore, the properties of our 3-server protocol: i.e., constant rounds, no cryptographic operation in online phase, poly log communication etc., all naturally extend to this 4-server protocol.

field). The shares are generated by choosing random elements from $\mathbb{F}$ subject to their sum being equal to y. We denote the shares generated as $(\langle y\rangle_0, \langle y\rangle_1)$. The second type of secret sharing is a Boolean secret sharing scheme. Here, the secret that needs to be shared is a binary string and the shares are uniformly sampled subject to their XOR being equal to the secret. We denote the Boolean share of a string v as $(\langle v\rangle_0^{\mathbf{B}}, \langle v\rangle_1^{\mathbf{B}})$.

2.1 Private Information Retrieval with GOD

We consider the problem of Private Information Retrieval (PIR) between a client C and the three servers $\mathcal{S}_0, \mathcal{S}_1$, and $\mathcal{S}_2$. All the three servers are given a copy of the database $D \in \mathbb{F}^N$ where $\mathbb{F}$ is a field and $N = 2^n$. The client has an index $i \in [N]$ and would like to retrieve the the i-th entry of the database D given by $D[i]$. PIR is a protocol that is run between the clients and the servers that allows the client to retrieve $D[i]$.

For security, we consider the setting where one of the servers might be corrupted by a malicious adversary $\mathcal{A}$. We require that the view of the adversary $\mathcal{A}$ and the output of the honest client in the real protocol to be computationally indistinguishable to the ideal world where the parties have access to an ideal functionality given by $\mathcal{F}$ (Functionality 1). This ideal functionality takes in $i \in [N]$ from the client and databases D_0, D_1, D_2 from the three servers respectively. Note that the honest servers are guaranteed to send the same database D to the oracle whereas the malicious server could send in an arbitrary database. The functionality checks if at least two of the three received databases are consistent (let us denote the consistent database by D) and if it is the case, it delivers $D[i]$ to the client. Note that in the ideal world, the honest client always receives $D[i]$ from the ideal functionality irrespective of the database sent by the malicious server. This ensures that the protocol realizing this functionality satisfies the stronger property of Guaranteed Output Delivery (GOD).

```
Functionality 1: Ideal functionality $\mathcal{F}$ for PIR with GOD
  Input: $C$ inputs $i$ and $\mathcal{S}_b (b \in \{0,1,2\})$ inputs database $D_b$.
1 Atleast two of $D_b$ received will be identical denoted by $D$.
2 Output $D[i]$ to $C$.
```

Definition 2.1 (PIR with Guaranteed Output Delivery). A PIR protocol Π between a client C and three servers $\mathcal{S}_0, \mathcal{S}_1$, and $\mathcal{S}_2$ is said to satisfy guaranteed output delivery if the following properties hold:

- **Security.** For any non-uniform PPT adversary $\mathcal{A}$ that corrupts one of the three servers, there exists an ideal world PPT simulator Sim such that for any client input $i \in [N]$ and any database $D \in \{0,1\}^N$ (where $N = 2^n$), we have:
$$\mathsf{Real}(\Pi, \mathcal{A}, i, D) \approx_c \mathsf{Ideal}(\mathcal{F}, \mathsf{Sim}, i, D)$$
where Real and Ideal refers to the joint distribution of the output of the client C and view of the adversary $\mathcal{A}$ and the view of Sim respectively in the real and ideal experiments.
- **Efficiency:** The communication cost of the protocol Π is $\mathsf{poly}(\lambda, n)$.

In simulation based security for multi party computation, it is allowed for client to identify an honest server and send it's index to the honest server in clear to receive output and achieve guaranteed output delivery. This is because the simulation based security definition only requires simulation of view of adversary and not of honest party. In fact, in most MPC protocols secure against malicious adversary, a malicious party can send it's state to an honest party which would allow the honest party to learn private inputs of other parties and this is not prevented by a secure protocol[2]. Similar to us, many notable works in the literature achieve guaranteed output delivery by identifying honest parties and running computation with them in the clear [15, 16, 21, 42, 45].

3 DISTRIBUTED POINT FUNCTIONS

We now recall the definition of distributed point functions from [36].

Definition 3.1 (Point Function). A point function $f_{\alpha,\beta}$ is defined using two parameters $(\alpha, \beta) \in [N] \times \mathbb{F}$ (where $N = 2^n$ and $\mathbb{F}$ is a finite field)

$$f_{\alpha,\beta} : [N] \to \mathbb{F}$$
$$f_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

We call β as the payload.

A two-party Distributed Point Function (DPF) scheme is a pair of PPT algorithms $(\Pi_{\mathsf{Gen}}, \Pi_{\mathsf{Eval}})$ with the following syntax:

- $\Pi_{\mathsf{Gen}}(1^\lambda, \alpha, \beta)$ is a key generation algorithm which on input $1^\lambda, \alpha \in [N]$ and $\beta \in \mathbb{F}$ outputs pair of keys (k_0, k_1).
- $\Pi_{\mathsf{Eval}}(b, k_b, x)$ is an evaluation algorithm that takes in $b \in \{0,1\}$, the key k_b, the evaluation point $x \in [N]$ and outputs $y_b^x \in \mathbb{F}$.

Definition 3.2 (Distributed Point Functions [36]). We require a two-party DPF to satisfy the following properties:

(1) **Correctness**: For all point functions $f_{\alpha,\beta}$ and every x in domain of $f_{\alpha,\beta}$,
$$Pr\Big[(k_0, k_1) \leftarrow \Pi_{\mathsf{Gen}}(1^\lambda, \alpha, \beta) \implies$$
$$y_0^x + y_1^x = f_{\alpha,\beta}(x) : \{y_b^x = \Pi_{\mathsf{Eval}}(k_b, x)\}_{b\in\{0,1\}}\Big] = 1$$
(2) **Privacy**: For every $b \in \{0,1\}$, there exists a simulator Sim such that for any point function $f_{\alpha,\beta}$, we have:
$$\Big\{k_b : (k_0, k_1) \leftarrow \Pi_{\mathsf{Gen}}(1^\lambda, \alpha, \beta)\Big\} \approx_c \Big\{\mathsf{Sim}(1^\lambda)\Big\}$$

3.1 Doerner-shelat DPF Key Generation Protocol

We use the Doerner-shelat DPF key generation protocol [30] and its extension to the case of arithmetic payloads [1] (henceforth, called the Ds protocol) as one of the key tools in our PIR construction. We abstract out the key properties that we need from this protocol below and later show that the Ds protocol (Algorithm 1) satisfies all these properties.

[2]To address this issue, a stronger security definition, Friends-and-Foes, was introduced recently [3]. In our work, we do not provide this security guarantee.

A DPF key generation protocol Φ is a two-party protocol between $\mathcal{S}_0$ and $\mathcal{S}_1$ with private inputs $(\langle\alpha\rangle_0^{\mathbf{B}}, \langle\beta\rangle_0) \in [N] \times \mathbb{F}$ and $(\langle\alpha\rangle_1^{\mathbf{B}}, \langle\beta\rangle_1) \in [N] \times \mathbb{F}$ respectively. The output of the party $\mathcal{S}_b$ (for each $b \in \{0,1\}$) at the end of the protocol is given by k_b where $(k_0, k_1) \leftarrow \Pi_{\mathsf{Gen}}(1^\lambda, \alpha, \beta)$, $\alpha = \langle\alpha\rangle_0^{\mathbf{B}} \oplus \langle\alpha\rangle_1^{\mathbf{B}}$ and $\beta = \langle\beta\rangle_0 + \langle\beta\rangle_1$.

Definition 3.3 (PIR-compatible DPF Key Gen Protocol). We say that a DPF key generation protocol Φ is PIR-compatible if it satisfies the following properties:

- **Perfect Correctness:** Φ computes the Π_{Gen} functionality with perfect correctness.
- **Security:** Φ securely implements the Π_{Gen} functionality of the DPF protocol against semi-honest adversaries.
- **Black-Box:** Φ makes black-box use of a PRG.
- **Interaction via Oracles:** There exists a specific two-party *non-cryptographic* functionality $\mathcal{G}$ such that the only interaction between the parties P_0 and P_1 in the DPF protocol is to securely compute this functionality $\mathcal{G}$. Therefore, in the $\mathcal{G}$-oracle model, the protocol is non-interactive.[3]
- **Efficiency:** The communication complexity of the protocol is $\mathsf{poly}(\lambda, n)$ (where $n = \log N$) and the computational complexity is $\mathsf{poly}(\lambda, N)$.

We define the view of a party in a PIR compatible DPF key generation protocol.

Definition 3.4 (View of a party in a PIR-Compatible DPF Key Gen). The view of a party in a PIR compatible DPF key gen protocol is given by its private input, private randomness, and the (input,output) pair for each $\mathcal{G}$-oracle call.

We make three observations about the view of a party in a PIR compatible DPF key generation protocol.

Proposition 3.5.

(1) *The size of the view of each party is at most* $\mathsf{poly}(\lambda, n)$.
(2) *Given a partial view that comprises of the input, private randomness and the outputs of each $\mathcal{G}$-oracle call, there is an algorithm that runs in time* $\mathsf{poly}(\lambda, N)$ *and outputs the full view.*
(3) *Given the view, the output obtained by the party in the DPF key generation protocol is computable by a deterministic algorithm that runs in time* $\mathsf{poly}(\lambda, N)$.

Proof.

(1) The size of view of $\mathcal{S}_b$ is sum of size of private input, randomness, and the (input,output) pair for each $\mathcal{G}$-oracle call. Private input $(\langle\alpha\rangle_b^{\mathbf{B}}, \langle\beta\rangle_b)$ is of size $\mathsf{poly}(n)$ and randomness is of size $poly(\lambda)$. The total size of (input, output) pairs for all $\mathcal{G}$-oracle call is $\mathsf{poly}(\lambda, n)$. This follows from communication efficiency of Φ and the fact that only communication that happens is to compute $\mathcal{G}$ (Defn. 3.3). Therefore, size of view is $\mathsf{poly}(\lambda, n)$.
(2) Given the partial view that comprises of the private input x, randomness r and the outputs of the $\mathcal{G}$-oracle $\{\mathsf{out}_i\}_{i\in[L]}$, the algorithm does the following.
    (a) It initializes an empty list C.
    (b) For each $i \in [L]$,
        (i) It computes $\mathsf{inp}_i = \mathsf{NextMsgFunc}(x, r, C)$ where $\mathsf{NextMsgFunc}$ denotes the next message function of the party.
        (ii) It adds $(\mathsf{inp}_i, \mathsf{out}_i)$ to C.

    It is easy to observe that the above procedure generates the full view and the efficiency follows from the computational complexity of the PIR-compatible protocol.
(3) It follows from correctness of the protocol and the bound on the computational complexity.

□

[3]By a non-interactive protocol in the $\mathcal{G}$-oracle model, we mean that the only interaction with the other party in the protocol is via the $\mathcal{G}$ oracle. That is, the parties only make sequential calls to the $\mathcal{G}$-oracle without any interaction with the other party.

We give a brief description of Ds protocol (Algorithm 1) and prove that the Ds protocol is a PIR compatible DPF key generation protocol by showing that it satisfies Definition 3.3.

Description of Ds protocol. Party P_b has private inputs $\langle r\rangle_b^{\mathbf{B}}, \langle v\rangle_b$ and obtains DPF key k_b corresponding to point function $f_{r,v}$. Prg is a pseudorandom generator mapping λ bit strings to $2\lambda+2$ bit strings. Since r is not known in clear, both parties compute Prg outputs for all possible nodes at level $j \in [1, n]$ in step 3. The correction words of DPF key $\{\sigma^j, \tau^{j,0}, \tau^{j,1}\}$ are computed by secure 2PC in step 5. This secure computation is modeled by non-cryptographic functionality $\mathcal{G}$ with $\left(\langle r_j\rangle_b^{\mathbf{B}}, \langle z^{j,0}\rangle_b^{\mathbf{B}}, \langle \hat{z}^{j,0}\rangle_b^{\mathbf{B}}, \langle z^{j,1}\rangle_b^{\mathbf{B}}, \langle \hat{z}^{j,1}\rangle_b^{\mathbf{B}}\right)$ as input and $(\sigma^j, \tau^{j,0}, \tau^{j,1})$ as output of $\mathcal{G}$-oracle call for party P_b.

Since the payload is over $\mathbb{F}$, P_0, P_1 compute W_0, W_1 using a pseudorandom function Convert that maps λ bit strings to elements in $\mathbb{F}$. The final secure computation in step 10 is modeled by $\mathcal{G}$ with (W_b, T_b) as input and γ as output of $\mathcal{G}$-oracle call for party P_b. The parties interact to compute the $\mathcal{G}$-oracle outputs only. The protocol has communication complexity $\mathsf{poly}(\lambda, n)$, computational complexity $\mathsf{poly}(\lambda, N)$ and it is secure against semi-honest adversary. The corresponding Π_{Eval} protocol is given in appendix A for completeness. For more details, refer to [1, 30].

Lemma 3.6. *The Ds protocol satisfies Definition 3.3.*

Proof.

(1) **Perfect Correctness:** We give description of Π_{Eval} in appendix A. The Π_{Eval} procedure of the scheme remains the same as that of the Ds protocol with Boolean payloads presented in [30], along with an additional step to account for the arithmetic conversion in step 10 of Algorithm 1. Borrowing notation and assuming honestly generated keys k_b, P_b performs the final correction operation as follows to obtain shares of output for input $x \in [N]$

$$(-1)^b \left(\mathsf{Convert}\left(\hat{S}_b^{n,x}\right) + \hat{t}_b^{n,x} \cdot \gamma\right)$$

Substituting the values from step 10, we get

$$\begin{aligned}\mathsf{output} &= \Pi_{\mathsf{Eval}}(0, k_0, x) + \Pi_{\mathsf{Eval}}(1, k_1, x)\\ &= (c_0^{n,x} - c_1^{n,x}) + (\hat{t}_0^{n,x} - \hat{t}_1^{n,x}) \cdot \gamma\end{aligned}$$

where $c_b^{n,x} = \mathsf{Convert}\left(\hat{S}_b^{n,x}\right)$.
Case 1: $x \neq r$. If $x \neq r$ then $c_0^{n,x} = c_1^{n,x}$, $\hat{t}_0^{n,x} = \hat{t}_1^{n,x}$ and output is 0.

```
Algorithm 1: Distributed Keygen with Arithmetic Shares
  Input: Each party $P_b$ for $b \in \{0,1\}$ holds additive shares
         $\langle r \rangle_b^{\mathbf{B}}$ of $r \in \{0,1\}^n$ and $\langle v \rangle_b$ of $v \in \mathbb{F}$.
  Output: DPF keys for $f_{r,v}$.
  Each party $P_b$ performs the following.
1 Sample $\hat{S}_b^{0,0} \in \{0,1\}^\lambda$ and set $\hat{t}_b^{0,0} = b$.
2 for $j = 1$ to $n$ do
3     For all $\ell \in [0, 2^{j-1})$, compute
      $\left\{\left(S_b^{j,2\ell} \| t_b^{j,2\ell} \| S_b^{j,2\ell+1} \| t_b^{j,2\ell+1}\right)\right\} = \left\{\mathsf{Prg}\left(\hat{S}_b^{j-1,\ell}\right)\right\}$.
4     Compute $\left(\langle z^{j,0}\rangle_b^{\mathbf{B}}, \langle \hat{z}^{j,0}\rangle_b^{\mathbf{B}}, \langle z^{j,1}\rangle_b^{\mathbf{B}}, \langle \hat{z}^{j,1}\rangle_b^{\mathbf{B}}\right) =$
      $\bigoplus_\ell \left(S_b^{j,2\ell} \| t_b^{j,2\ell} \| S_b^{j,2\ell+1} \| t_b^{j,2\ell+1}\right)$.
5     Secure Computation:
      - Inputs: Boolean sharing of $r_j$ and $\left(z^{j,0}, \hat{z}^{j,0}, z^{j,1}, \hat{z}^{j,1}\right)$.
      - Compute:
            $\sigma^j \leftarrow \langle z^{j,\overline{r_j}}\rangle_0^{\mathbf{B}} \oplus \langle z^{j,\overline{r_j}}\rangle_1^{\mathbf{B}}$
            $\langle \tau^{j,0}\rangle_b^{\mathbf{B}} \leftarrow \langle \hat{z}^{j,0}\rangle_b^{\mathbf{B}} \oplus \langle r_j\rangle_b^{\mathbf{B}} \oplus b$
            $\langle \tau^{j,1}\rangle_b^{\mathbf{B}} \leftarrow \langle \hat{z}^{j,1}\rangle_b^{\mathbf{B}} \oplus \langle r_j\rangle_b^{\mathbf{B}}$
      - Output $CW^j = \left(\sigma^j, \tau^{j,0}, \tau^{j,1}\right)$ to both.
6     Compute $\left\{\hat{S}_b^{j,\ell}\right\}_\ell = \left\{S_b^{j,\ell} \oplus \hat{t}_b^{j-1,\lfloor \ell/2 \rfloor} \cdot \sigma^j\right\}_\ell$.
7     Compute $\left\{\hat{t}_b^{j,\ell}\right\}_\ell = \left\{t_b^{j,\ell} \oplus \hat{t}_b^{j-1,\lfloor \ell/2 \rfloor} \cdot \tau^{j,\mathrm{Lsb}(\ell)}\right\}_\ell$.
8 end
9 Compute $W_b = \sum_{\ell \in [0,2^n)} \mathsf{Convert}\left(\hat{S}_b^{n,\ell}\right), T_b = \sum_{\ell \in [0,2^n)} \hat{t}_b^{n,\ell}$
10 Secure Computation:
  - Inputs: $T_b, W_b$ and arithmetic sharing of $v$.
  - Compute:
        $q = 1$ if $T_1 > T_0$, $0$ otherwise.
        $\gamma \leftarrow (-1)^q (v - W_0 + W_1)$
  - Output $\gamma$ to both.
11 Output $k_b \leftarrow \left(\hat{S}_b^{0,0}, \left\{CW^j\right\}_{j \in [1,n]}, \gamma\right)$.
```

Case 2: $x = r$. WLOG assume that $\hat{t}_0^{n,x} = 1$ and $\hat{t}_1^{n,x} = 0$. Now we have

$$\text{output} = (c_0^{n,x} - c_1^{n,x}) + (v - W_0 + W_1)$$

Since $W_1 - W_0 = \sum_{\ell \in [0,2^n)} (c_1^{n,\ell} - c_0^{n,\ell}) = (c_1^{n,x} - c_0^{n,x})$ as $c_0^{n,\ell} = c_1^{n,\ell}$ for all $\ell \neq x$, we get output $= v$. Hence the Ds protocol computes Π_{Gen} with perfect correctness.

(2) **Security:** The parties do not gain any additional information other than the DPF key. In semi honest setting, the view of each party can be simulated using the final output key and the protocol is secure. [1, 30]
(3) **Black-Box:** The parties P_0 and P_1 only use black-box oracle calls to Prg and Convert.
(4) **Interaction Via Oracles:** The only interaction that occurs between P_0 and P_1 in the protocol is in the secure computations of step 5 and step 10. Both these computations involve only arithmetic and XOR operations which can be modeled as non-cryptographic functionality $\mathcal{G}$ (for a formal description of $\mathcal{G}$, see Appendix B). Hence, all communication occurs only in the secure computation of functionality $\mathcal{G}$.
(5) **Efficiency.** The total number of PRG calls required is $O(N)$, since the local computation requires XOR to be performed over all nodes in each layer of DPF tree. There are $n+1$ secure computations each requiring communication of $\mathsf{poly}(\lambda)$. It follows that the communication cost and computational cost of the protocol are $\mathsf{poly}(n, \lambda)$ and $\mathsf{poly}(N, \lambda)$ respectively, with N being the database size and $n = \log N$.

We conclude that the Ds protocol is PIR-compatible DPF key gen protocol. □

4 PROTOCOL FOR PIR WITH GOD

In this section, we make use of a PIR compatible DPF Key Gen protocol Φ to construct a PIR protocol between a client and three servers that satisfies guaranteed output delivery. We then instantiate Φ with Ds protocol and report concrete costs of our PIR protocol.

Description of the Protocol. We give the formal description of the protocol in Algorithm 2. Our protocol uses PIR-compatible DPF key generation from Ds [30] as defined in Definition 3.3. Let D be the database held by the servers $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$ with $N = 2^n$ entries. Here we consider the database entries in a field $\mathbb{F}$ of size 2^κ for statistical security parameter κ. Later we show in Section 4.3 how our scheme can be naturally extended to databases with arbitrarily large entries.

Offline Phase. First, $\mathcal{S}_2$ picks a random $r \in [N]$ and $\alpha \in \mathbb{F}$. We would be working with a DPF for function $f_{r,(1,\alpha)}$. $\mathcal{S}_2$ generates secret shares of r and $v = (1, \alpha)$ that would be the private inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$ in the distributed key generation. Next, $\mathcal{S}_2$ runs the protocol Φ locally on these inputs to generate the two views view_0 and view_1 corresponding to $\mathcal{S}_0$ and $\mathcal{S}_1$. Now, $\mathcal{S}_2$ sends the partial view in Φ consisting of the inputs, private randomness, and outputs to $\mathcal{G}$-oracle calls in view_b to $\mathcal{S}_b$ for $b \in \{0,1\}$.

Now given the partial view, $\mathcal{S}_b$ creates the full view, denoted by $\overline{\mathsf{view}_b}$, locally (see Proposition 3.5 (2)). And given the full view, $\mathcal{S}_b$ computes its share of the function key k_b (see Proposition 3.5 (3)). Next, $\mathcal{S}_b$ evaluates the function key k_b on the whole domain. That is, it obtains $(\langle z^x \rangle_b \| \langle y^x \rangle_b) = \Pi_{\mathsf{Eval}}(k_b, x)$ for all $x \in [N]$.

As is clear, the offline phase has a single round of messages from $\mathcal{S}_2$ to $\mathcal{S}_0, \mathcal{S}_1$ of size $O(\lambda n)$. The computational complexity of all the servers is $O(N)$ PRG calls. Note that the offline phase is independent of contents of D.

Online Phase. A client comes in with input $i \in [N]$. Since one of the servers can be malicious, the first step is to ensure that $\mathcal{S}_0$ and $\mathcal{S}_1$ are indeed holding a DPF key for a special point r with payload $(1, \alpha)$ for some α. We want the client to verify this without doing any cryptographic operations. For this, the client checks for consistency of the two views as reported by $\mathcal{S}_0, \mathcal{S}_1$ with the one claimed by $\mathcal{S}_2$. If there is an inconsistency between reported views of $\mathcal{S}_2$ and $\mathcal{S}_b$ with $b \in \{0,1\}$, it is clear that one of them is cheating and hence, $\mathcal{S}_{1\oplus b}$ is the honest party that client can rely upon. However, it does not suffice that the views match. A malicious $\mathcal{S}_2$ can compute $\mathcal{G}$-oracle calls maliciously resulting in incorrect

Algorithm 2: PIR with Guaranteed Output Delivery

```
Input: $C$ holds $i \in [N]$ and $\mathcal{S}_b$ holds $D \in \mathbb{F}^N$ where
    $b \in \{0,1,2\}, N = 2^n$.
Output: $C$ outputs $D[i] \in \mathbb{F}$.
Offline phase
1  $\mathcal{S}_2$ samples random $r \leftarrow [N]$, $\alpha \leftarrow \mathbb{F}$.
2  $\mathcal{S}_2$ generates $(\langle r\rangle_0^{\mathbf{B}}, \langle r\rangle_1^{\mathbf{B}})$ and $(\langle v\rangle_0, \langle v\rangle_1)$ as Boolean shares
     of $r$ and arithmetic shares of $(1,\alpha) \in \mathbb{F}^2$ respectively.
3  $\mathcal{S}_2$ runs the protocol $\Phi$ on inputs $(\langle r\rangle_0^{\mathbf{B}}, \langle v\rangle_0)$ and
     $(\langle r\rangle_1^{\mathbf{B}}, \langle v\rangle_1)$ in its "head" and generates the view of $P_0$
     given by $\mathsf{view}_0$ and the view of $P_1$ given by $\mathsf{view}_1$ in the
     protocol.
4  For each $b \in \{0,1\}$, $\mathcal{S}_2$ sends the partial view comprising of
     the private input, the private randomness and the outputs
     of each $\mathcal{G}$-oracle call in $\mathsf{view}_b$ to $\mathcal{S}_b$. Note that $\mathcal{S}_2$ does
     not send the inputs to each $\mathcal{G}$-oracle call.
5  From the partial view obtained from $\mathcal{S}_2$, for each $b \in \{0,1\}$,
     $\mathcal{S}_b$ generates the rest of the view. Let us call the updated
     full view by $\overline{\mathsf{view}_b}$ and let $k_b$ be the output obtained in
     $\overline{\mathsf{view}_b}$.
6  for $b \in \{0,1\}$ do
7      for $x \in [N]$ do
8          $\mathcal{S}_b$ computes $\langle z^x\rangle_b \in \{0,1\}, \langle y^x\rangle_b \in \mathbb{F}$ as
             $(\langle z^x\rangle_b \| \langle y^x\rangle_b) = \Pi_{\mathsf{Eval}}(k_b, x)$.
9      end
10 end
Online phase
11 $\mathcal{S}_2$ sends $(\mathsf{view}_0, \mathsf{view}_1)$ to $C$ and for each $b \in \{0,1\}$, $\mathcal{S}_b$
     sends $\overline{\mathsf{view}_b}$ to $C$.
12 On receving the views from $\{\mathcal{S}_j\}_{j\in\{0,1,2\}}$, $C$ collects the set
     $\{(\mathsf{inp}_\ell, \mathsf{out}_\ell)\}_{\ell\in[L]}$ as the sequence of (input,output)
     values to each $\mathcal{G}$-oracle call in $(\mathsf{view}_0, \mathsf{view}_1)$.
13 $C$ identifies honest party $\mathcal{S}_T$ as follows:
         if $\exists b \in \{0,1\}$ s.t. $\mathsf{view}_b \neq \overline{\mathsf{view}_b}$, then $\mathcal{S}_T = \mathcal{S}_{b\oplus 1}$
         else if $\exists \ell \in [L]$ s.t. $\mathsf{out}_\ell \neq \mathcal{G}(\mathsf{inp}_\ell)$, then $\mathcal{S}_T = \mathcal{S}_0$
     If $\mathcal{S}_T$ is identified, go to step 18, continue otherwise.
14 $C$ computes $(\alpha, r)$ from $(\mathsf{view}_0, \mathsf{view}_1)$ and sends $(i-r)$ to
     party $\mathcal{S}_0, \mathcal{S}_1$.
15 For each $b \in \{0,1\}$, $\mathcal{S}_b$ computes :
         $\langle z\rangle_b = \sum_{x\in[N]} \langle z^x\rangle_b \cdot D[x+(i-r)]$
         $\langle y\rangle_b = \sum_{x\in[N]} \langle y^x\rangle_b \cdot D[x+(i-r)]$
     and sends $\langle z\rangle_b, \langle y\rangle_b$ to $C$.
16 $C$ computes $z = \langle z\rangle_0 + \langle z\rangle_1$ and $y = \langle y\rangle_0 + \langle y\rangle_1$. It then
     checks if $\alpha \cdot z = y$.
17 If the check passes, $C$ outputs $D[i] = z$; else, it identifies
     trusted party $\mathcal{S}_T = \mathcal{S}_2$ and goes to step 18.
18 $C$ sends $i$ to $\mathcal{S}_T$ and receives $D[i]$.
```

keys while ensuring that the consistency check passes. To catch this attack, the client also checks the consistency of inputs and outputs of $\mathcal{G}$-oracle calls as reported by $\mathcal{S}_2$. If this check fails, it is clear that $\mathcal{S}_2$ is malicious and hence, $\mathcal{S}_0, \mathcal{S}_1$ are honest. Note that since $\mathcal{G}$ is non-cryptographic, the client does not need to do any cryptographic operations.

The client proceeds to the next step if all the above consistency checks pass. Next, the client sends $(i-r)$ to $\mathcal{S}_0, \mathcal{S}_1$, where r is computed from $\mathsf{view}_0, \mathsf{view}_1$. Now the servers rotate the database based on $(i-r)$ and take an inner-product with $\{(\langle z^x\rangle_b \| \langle y^x\rangle_b)\}_{x\in[N]}$, to get $(\langle z\rangle_b, \langle y\rangle_b)$ that is sent back to the client. The client checks the validity of MAC, i.e., whether $z \cdot \alpha = y$. If the check fails, then one of $\mathcal{S}_0, \mathcal{S}_1$ is cheating and the client identifies $\mathcal{S}_2$ as the honest party.

Hence, in our protocol, either all checks pass and the client outputs z or it has identified an honest party. In the latter case, it sends i to the honest server to learn correct $D[i]$ reliably. Thus, achieving guaranteed output delivery with one malicious server. Recall that it is allowed for client to send index to identified honest party in clear and this does not violate the definition of simulation based security for MPC protocols (Section 2.1).

Remark: The Ds protocol is such that the full view creation from the partial view itself provides the evaluation of the function key on all points. That is, as an optimization, the evaluation phase need not be done separately. Hence, the PRG invocations done during key generation suffice.

We prove the following theorem.

THEOREM 4.1. *Assuming that Φ is a PIR-compatible DPF key generation protocol (see Definition 3.3), the construction given in Algorithm 2 satisfies Definition 2.1. The communication in offline phase is $2n(\lambda+3)+\lambda+8\log|\mathbb{F}|$ bits and 1 round. The communication in online phase is $11n\lambda+28n+4\lambda+26\log|\mathbb{F}|$ and 3 rounds.*

4.1 Proof of Theorem 4.1

The efficiency requirements of the protocol follow directly from Proposition 3.5. We now prove the security property. We consider two cases.

Case-1: $\mathcal{S}_2$ is the corrupted server. In this case, we first observe that if any of the checks described in Step 13 do not pass, then one of $\mathcal{S}_0$ or $\mathcal{S}_1$ is designated as the trusted server $\mathcal{S}_T$. In this case, the output of the real and the ideal experiments are identically distributed. Hence, let us assume that these two checks pass.

This means that for each $b \in \{0,1\}$, $\mathsf{view}_b = \overline{\mathsf{view}_b}$ and for each $\ell \in [L]$, $\mathsf{out}_\ell = \mathcal{G}(\mathsf{inp}_\ell)$. Since $\mathcal{S}_0$ and $\mathcal{S}_1$ are honest servers, it follows via a standard induction on the number of calls to $\mathcal{G}$ that $(\mathsf{view}_0, \mathsf{view}_1)$ is correctly computed by $\mathcal{S}_2$. In other words, it is consistent with the inputs, randomness and the outputs of correctly computed $\mathcal{G}$-oracle calls. This implies that check done in Step 16 passes with probability 1 (this follows from the perfect correctness of the protocol Φ). Thus, the client correctly obtains $D[i]$ and never designates $\mathcal{S}_2$ as the trusted server $\mathcal{S}_T$. We note that the view of the server $\mathcal{S}_2$ is independent of i and hence, it is trivially simulatable.

Case-2: $\mathcal{S}_b$ for some $b \in \{0,1\}$ is the corrupted server. In this case, $\mathcal{S}_2$ is honest and hence, $(\mathsf{view}_0, \mathsf{view}_1)$ are correctly computed.

Therefore, for each $\ell \in [L]$, it follows that $\mathsf{out}_\ell = \mathcal{G}(\mathsf{inp}_\ell)$. If there exists a $b \in \{0,1\}$ such that $\mathsf{view}_b \neq \overline{\mathsf{view}}_b$, then it follows that $\mathcal{S}_b$ is the corrupt server. In this case, we designate $\mathcal{S}_{1\oplus b}$ as the honest server and the real and ideal experiments are identically distributed. Hence, let us assume that the checks described in Step 13 pass. For this case, we prove security by a sequence of hybrids.

- Hyb_0 : This corresponds to the real execution of the protocol.
- Hyb_1 : In this hybrid, instead of performing the checks described in Step 16, we check if

$$\langle z\rangle_b = \sum_{x\in[N]} \langle z^x\rangle_b \cdot D[x+(i-r)]$$

$$\langle y\rangle_b = \sum_{x\in[N]} \langle y^x\rangle_b \cdot D[x+(i-r)]$$

  If not, we abort. Note that if the above check passes, then it follows from the perfect correctness of Φ that check described in Step 16 also passes. We now argue that the converse holds except with negligible probability.

  We first observe that there is an unique value of α such that the check described in Step 16 passes, but the above check does not pass. In particular, if $(\langle z\rangle_b, \langle y\rangle_b)$ is the output of the correct computation and $(\langle z'\rangle_b, \langle y'\rangle_b)$ is the incorrect output produced by $\mathcal{S}_b$ that passes the check in Step 16, then $\alpha = \frac{\langle y\rangle_b - \langle y'\rangle_b}{\langle z\rangle_b - \langle z'\rangle_b}$. In other words, we can compute the value of α. However, it follows from the semi-honest security of Φ that α cannot be computed except with probability $1/|\mathbb{F}| + \mathsf{negl}(\lambda)$. To see why this is the case, we first generate the partial view sent to $\mathcal{S}_b$ using the simulator for Φ. It follows from the security of Φ that except with negligible probability, $\mathcal{S}_b$ still produces an incorrect evaluation that passes the check in Step 16 and hence, we would still be able to compute the value of α. Now, in the modified distribution α is randomly chosen and independent of view of $\mathcal{S}_b$. Hence, the probability of computing α in this modified distribution is at most $1/|\mathbb{F}|$. Thus, Hyb_1 and Hyb_2 are computationally indistinguishable.
- Hyb_2 : In this hybrid, we generate the partial view sent by $\mathcal{S}_2$ to $\mathcal{S}_b$ using the simulator for Φ instead of computing it honestly as per the protocol specification. It follows from the semi-honest security of the protocol Φ that Hyb_1 and Hyb_2 are computationally indistinguishable. Note that the view of $\mathcal{S}_b$ in Hyb_2 is independent of i as r is uniformly distributed.

If the check described in Hyb_1 passes, then it is easy to see that the client obtains the correct $D[i]$ as $S_{1\oplus b}$ is honest. This shows that the real and the ideal experiments are computationally indistinguishable with Hyb_2 as the ideal experiment.

4.2 Concrete Efficiency

Instantiating keygen protocol Φ with Ds protocol, we give the concrete efficiency of our PIR protocol.

Computation: In offline phase, there are a total of $N + N\left\lceil \frac{\log|\mathbb{F}|}{\lambda+1} \right\rceil - 1$ PRG invocations by each server $\mathcal{S}_j,\ j \in \{0,1,2\}$. When we set $\log \mathbb{F} \approx 64$, we get $2N-1$ PRG calls. According to description of algorithm 2, P_b computes output $(\langle z^x\rangle_b \| \langle y^x\rangle_b) = \Pi_{\mathsf{Eval}}(k_b, x)$ for $b \in \{0,1\}, x \in [N]$. This would require more AES calls however the output can be directly computed from the view $\overline{\mathsf{view}_b}$ and no extra AES calls are required for Π_{Eval}.

In the online phase, $\mathcal{S}_0, \mathcal{S}_1$ compute inner product of shares of point function output with database requiring $O(N)$ multiplications and additions over $\mathbb{F}$. There are no cryptographic operations in online phase.

Lightweight client: The computational complexity of client is $O(n)$. The client only computes $\mathcal{G}$ which is non-cryptographic (Definition 3.3) making it extremely lightweight.

Communication: In offline phase, $\mathcal{S}_2$ sends partial view of size $n(\lambda+3)+\lambda+4\log|\mathbb{F}|$ bits to both $\mathcal{S}_0, \mathcal{S}_1$. There is only server-server communication in offline phase.

In the online phase $\mathcal{S}_2$ sends $(\mathsf{view}_0, \mathsf{view}_1)$ and $\{\mathcal{S}_b\}_{b\in\{0,1\}}$ sends $\overline{\mathsf{view}_b}$ to C where $|\mathsf{view}_b| = n(3\lambda+7)+\lambda+6\log|\mathbb{F}|$ bits. Outputs of $\mathcal{G}$, $\{out_\ell\}_{\ell\in[L]}$ are common in $\mathsf{view}_0, \mathsf{view}_1$ so $\mathcal{S}_2$ can send just one copy of $\{out_\ell\}_{\ell\in[L]}$. C sends $(i-r)$ to $\mathcal{S}_0, \mathcal{S}_1$ and $\mathcal{S}_b$ sends $(\langle z\rangle_b, \langle y\rangle_b)_{b\in\{0,1\}}$ to C. The total communication is $11n\lambda + 28n + 4\lambda + 26\log|\mathbb{F}|$ bits and 3 rounds. If $\mathcal{S}_0$ or $\mathcal{S}_1$ behaves maliciously during online phase, C queries $D[i]$ from $\mathcal{S}_2$ in clear increasing the rounds to 5. There is only client-server communication in online phase.

4.3 Extension to large database entries

The previous protocol was over databases with entries in $\mathbb{F}$. This protocol can be naturally extended to PIR over database with arbitrary size entries. Statistical security and computational security parameters are same as before. Database D with entry size k bits can be divided into $m = \lceil k/\kappa \rceil$ databases of entry size κ bits on which servers run online phase of protocol. Each of these databases have entries in $\mathbb{F}$ (size of $\mathbb{F}$ is 2^κ) and servers compute inner product on each of these databases individually and send shares for client to reconstruct the correct entry.

(1) The offline phase remains the same as described in Algorithm 2.
(2) Next in the online phase, servers $S_b,\ b \in \{0,1\}$ compute (for each database $D_j,\ j \in [m]$)

$$\langle z\rangle_b^j = \sum_{x\in[N]} \langle z^x\rangle_b \cdot D_j[x+(i-r)]$$

$$\langle y\rangle_b^j = \sum_{x\in[N]} \langle y^x\rangle_b \cdot D_j[x+(i-r)]$$

(3) The servers S_b send $\{\langle z\rangle_b^j, \langle y\rangle_b^j\}_{j\in[m]}$ to the client.
(4) The client reconstructs z^j, y^j for all j and identifies $\mathcal{S}_2$ as honest party if for any j, $\alpha \cdot z^j \neq y^j$.
(5) If the check above passes, client outputs $z = z^0 \| \ldots \| z^{m-1} \in \mathbb{Z}_{2^k}$ i.e. concatenation of binary $\{z^j\}_{j\in m}$.

Security. Security follows directly from the base protocol. The extended protocol can be seen as PIR on $\lceil k/\kappa \rceil$ databases of entry size κ bits. The security follows from security of PIR on each database which follows from security of our main protocol.

Concrete Cost. Here we summarize the cost of this modified protocol in a similar manner as above.

Computation cost: The offline computation is dominated by $2N-1$ PRG calls same as before. In the online phase, servers compute $O(mN)$ additions and multiplications over $\mathbb{F}$ where $m = \lceil k/\kappa \rceil$.

Communication cost: In the offline phase, $\mathcal{S}_2$ sends a partial view of size $n(\lambda+3)+\lambda+4\log|\mathbb{F}|$ bits to both $\mathcal{S}_0, \mathcal{S}_1$ each.

In the online phase, servers send their view to client of size $n(3\lambda+7)+\lambda+6\log|\mathbb{F}|$ bits. Client sends $(i-r)$ to $\mathcal{S}_0, \mathcal{S}_1 : 2n$ bits. $\mathcal{S}_b$ sends $\{\langle z\rangle_b^j, \langle y\rangle_b^j\}_{j\in[n]}$ communicating $2k$ bits each. Total communication is $11n\lambda+28n+4\lambda+22\log|\mathbb{F}|+4k$ bits.

5 IMPLEMENTATION AND EVALUATION

In this section, we discuss the performance of our PIR protocol. We implement both our protocol and its extension to larger database[4]. We also compare our protocol to baseline 3-server PIR protocols with GOD, described in Section 1, that can be obtained based on state-of-the-art single server PIR protocols.

Protocol Parameters: We set the computational security parameter $\lambda = 128$. We choose $\mathbb{F} = \mathbb{Z}_p$ where p is a 64 bit prime. We implement our protocol for database with entry size 8 B and protocol extension to larger database for database with entry size 1KB. For database with 1KB entry, we divide database entries into chunks of 8 B. We benchmark online, offline execution time and communication. We also benchmark client runtime i.e. the time it takes for client to verify views sent by server.

Implementation Details: We implement the PIR protocol in C++. We use 128-bit block size and AES − 128 as our PRG function. We use the implementation of GroupElement class from [55] for inner product computations in the online phase. We use OpenMP to manage multi-threading and local AES computations were done using the cryptoTools [59] library. We use multi-threading in local AES calls during the offline phase and inner product with database in the online phase.

Experimental Setup: We ran all 4 parties (client and three servers) on a single machine with 32 core Intel Xeon 2.6GHz CPU with 64GB RAM. Each of the 3 servers use 8 threads for multithreading while the client uses a single thread. We configure a LAN connection with bandwidth 800 Mbps and a WAN connection with bandwidth 20 Mbps using linux tc tool. Since there are only 4 rounds and < 10KB communication (in total), the performance of our protocol is the same in both LAN and WAN setting.

5.1 Performance

Table 1 reports the communication cost of our protocol for different database configurations. Table 2 reports our end-to-end performance of both offline and online phase for our PIR protocol in the LAN and WAN setting for databases with entry sizes 8B and 1KB. Since the offline phase is independent of database size, we do not report performance of offline phase for 8B-entry and 1KB-entry database separately. All the experiments are run 10 times and average values are reported.

Offline Phase: The offline phase is independent of database entry size. The communication in the offline phase (see Table 1) is only one way, from $\mathcal{S}_2$ to $\mathcal{S}_0, \mathcal{S}_1$, for sending the partial view of DPF key generation protocol which is poly$(\log N, \lambda)$ bits in size.

The computation in the offline phase is dominated by $O(N)$ PRG calls by servers. In our implementation, $\mathcal{S}_2$ computes view of $\mathcal{S}_0, \mathcal{S}_1$ and then sends the partial view to the respective party who then compute their complete view. This generation of views by $\mathcal{S}_2$ and $\mathcal{S}_0/\mathcal{S}_1$ can further be parallelized if $\mathcal{S}_2$ sends parts of the partial view as it computes the view instead of computing view entirely and then sending it to $\mathcal{S}_0, \mathcal{S}_1$. This optimization would further reduce the offline time by 1.5×.

Online Phase: The communication in the online phase (see Table 1) is higher than in the offline phase and scales linearly with database entry size. There is only client-server communication in the online phase. For larger database entries, the communication is dominated by $\langle z\rangle_b, \langle y\rangle_b$ sent by $\mathcal{S}_0, \mathcal{S}_1$ to C which is 4× the database entry size.

Our implementation results in a very fast online phase in which $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$ send their views to C for the consistency check and compute the inner product of DPF outputs with the *rotated* database. For database with entry size 1KB, we use multi-threading to compute inner product with 8B chunks of database entries in parallel.

Table 1: Communication (KB) of PIR protocol for database with N entries of 8B and 1KB.

N	Offline	Online	
		8B entry	1KB entry
2^{16}	0.67	3.19	7.16
2^{17}	0.70	3.39	7.36
2^{18}	0.74	3.58	7.54
2^{19}	0.77	3.76	7.73
2^{20}	0.81	3.95	7.91
2^{21}	0.84	4.13	8.10
2^{22}	0.88	4.32	8.28
2^{23}	0.91	4.50	8.47
2^{24}	0.95	4.69	8.66

Table 2: Runtime (ms) of PIR protocol for database with N entries of size 8B and 1KB in LAN/WAN setting.

N	Offline	Online		Client Time
		8B entry	1KB entry	
2^{16}	41	2	19	1
2^{17}	80	3	32	1
2^{18}	152	4	63	1
2^{19}	295	6	99	1.1
2^{20}	577	11	179	1.1
2^{21}	1, 128	21	329	1.2
2^{22}	2, 215	40	643	1.2
2^{23}	4, 489	78	1, 309	1.2
2^{24}	8, 879	152	2, 618	1.3

[4]Our code will be made available upon publication of the paper.

Client Time: This measures compute time on the client side separately, i.e., the time taken by client to receive (inp, view) from $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$ and check for consistency before providing masked index i. This client computation is independent of database entry size and lightweight ($\approx$ 1ms for all $2^{16} \leq N \leq 2^{24}$).

5.2 Comparison to other works

We compare our work to the baseline 3-server PIR solution that can be obtained based on single server PIR as outlined earlier. State-of-the-art single server PIR protocols like FrodoPIR [29] and SPIRAL [52] provide privacy against malicious server but do not guarantee correctness of output. As described in section 1, we can obtain GOD by running 3 instances of single server PIR on 3 separate servers tolerating 1 malicious corruption. We refer to this protocol as *three server PIR with GOD based on single server PIR*. We run the implementation of FrodoPIR[5] and SPIRAL[6] available online, in both LAN and WAN setting. Based on this single server PIR, in order to extrapolate the performance of a 3-server PIR with GOD security, we ignore any additional computational cost on the client side, assume that the 3-servers can completely run all of their computation in parallel at the same time (thus causing no further overhead) and only account for 3× communication required to be sent by the 3 servers to the client (which will affect performance based on the network setting).

We compare our solution with those based on FrodoPIR and SPIRAL on three different database sizes - 2^{20}×256B (256 MB), 2^{18}× 30KB (7.5 GB) and 2^{14}× 100KB (1.6 GB) and present our results in Table 3. The total communication of our protocol is 306 − 4893 × lower than SPIRAL and 2242−4230 × lower than FrodoPIR, while the total run time is 2.3−146 × faster than SPIRAL and 443−6036 × faster than FrodoPIR across network settings. Even when comparing only online time, our protocol is 1.5 − 1878 × more communication efficient and 2.2 − 37 × faster than the state-of-the-art. Hence our protocol achieves GOD with high efficiency and the performance doesn't deteriorate even with bandwidth constrained client.

Table 3: Comparison of this work to PIR protocols with GOD based on single server PIR. Runtime is in seconds and communication is in KB. The best values are marked in bold.

LAN Setting					
Database	Protocol	Offline		Online	
		Time	Comm.	Time	Comm.
2^{20}×256B	SPIRAL	0.75	43,008	1.00	102
	FrodoPIR	328.00	4,731	0.30	15,024
	This work	**0.58**	**0.81**	**0.16**	**8**
2^{18}×30KB	SPIRAL	0.81	55,296	16.80	300
	FrodoPIR	7,708.00	509,952	4.30	3,360
	This work	**0.15**	**0.74**	**1.16**	**124**
2^{14}×100KB	SPIRAL	2.1	144,384	4.03	606
	FrodoPIR	1,621.62	1,701,888	0.90	1,152
	This work	**0.01**	**0.6**	**0.41**	**402**

WAN Setting					
Database	Protocol	Offline		Online	
		Time	Comm.	Time	Comm.
2^{20}×256B	SPIRAL	17.13	43,008	1.03	102
	FrodoPIR	328.84	4,731	6.03	15,024
	This work	**0.58**	**0.81**	**0.16**	**8**
2^{18}×30KB	SPIRAL	21.87	55,296	16.92	300
	FrodoPIR	7,902.20	509,952	5.58	3,360
	This work	**0.15**	**0.74**	**1.16**	**124**
2^{14}×100KB	SPIRAL	57.09	144,384	4.26	606
	FrodoPIR	2,269.80	1,701,888	1.34	1,152
	This work	**0.01**	**0.6**	**0.41**	**402**

REFERENCES

[1] Amit Agarwal, Stanislav Peceny, Mariana Raykova, Phillipp Schoppmann, and Karn Seth. 2022. Communication Efficient Secure Logistic Regression. *IACR Cryptol. ePrint Arch.* (2022).

[2] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo. 2021. Communication–Computation Trade-offs in PIR. In *USENIX Security Symposium.* 1811–1828. https://eprint.iacr.org/2019/1483.pdf

[3] Bar Alon, Eran Omri, and Anat Paskin-Cherniavsky. 2020. MPC with Friends and Foes. In *Advances in Cryptology – CRYPTO 2020*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer International Publishing, Cham, 677–706.

[4] Andris Ambainis. 1997. Upper Bound on Communication Complexity of Private Information Retrieval. In *International Colloquium on Automata, Languages and Programming.*

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. *2018 IEEE Symposium on Security and Privacy (SP)* (2018), 962–979.

[6] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. *2018 IEEE Symposium on Security and Privacy (SP)* (2018), 962–979.

[7] Sebastian Angel and Srinath T. V. Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *USENIX Symposium on Operating Systems Design and Implementation.*

[8] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. 2017. Optimized Honest-Majority MPC for Malicious Adversaries — Breaking the 1 Billion-Gate Per Second Barrier. In *2017 IEEE Symposium on Security and Privacy (SP).* 843–862. https://doi.org/10.1109/SP.2017.15

[9] Amos Beimel and Yuval Ishai. 2001. Information-Theoretic Private Information Retrieval: A Unified Construction. *Electron. Colloquium Comput. Complex.* TR01 (2001).

[10] Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. 2005. General constructions for information-theoretic private information retrieval. *J. Comput. Syst. Sci.* 71 (2005), 213–247.

[11] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Ilan Orlov. 2012. Share Conversion and Private Information Retrieval. *2012 IEEE 27th Conference on Computational Complexity* (2012), 258–268.

[12] Nikita Borisov, George Danezis, and Ian Goldberg. 2015. DP5: A Private Presence Service. *Proceedings on Privacy Enhancing Technologies* 2015 (2015), 24 – 4.

[13] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions *(CCS '16).* Association for Computing Machinery, New York, NY, USA, 1292–1303. https://doi.org/10.1145/2976749.2978429

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure Computation with Preprocessing via Function Secret Sharing. In *Theory of Cryptography*, Dennis Hofheinz and Alon Rosen (Eds.). Springer International Publishing, Cham, 341–371.

[15] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. 2019. Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19).* Association for Computing Machinery, New York, NY, USA, 869–886. https://doi.org/10.1145/3319535.3363227

[16] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. 2020. Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs. In *Advances in Cryptology – ASIACRYPT 2020*, Shiho Moriai and Huaxiong Wang (Eds.). Springer International Publishing, Cham, 244–276.

[17] Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. 2020. Efficient 3-Party Distributed ORAM. In *Security and Cryptography for Networks: 12th International Conference, SCN 2020, Amalfi, Italy, September 14–16, 2020, Proceedings* (Amalfi, Italy). Springer-Verlag, Berlin, Heidelberg, 215–232. https://doi.org/10.1007/978-3-030-57990-6_11

[5] https://github.com/brave-experiments/frodo-pir
[6] https://github.com/menonsamir/spiral

[18] Christian Cachin, Silvio Micali, and Markus Stadler. 1999. Computationally Private Information Retrieval with Polylogarithmic Communication. In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 402–414.

[19] Yan-Cheng Chang. 2004. Single Database Private Information Retrieval with Logarithmic Communication. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings (Lecture Notes in Computer Science, Vol. 3108)*, Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan (Eds.). Springer, 50–61. https://doi.org/10.1007/978-3-540-27800-9_5

[20] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop* (London, United Kingdom) *(CCSW'19)*. Association for Computing Machinery, New York, NY, USA, 81–92. https://doi.org/10.1145/3338466.3358922

[21] David Chaum, Claude Crépeau, and Ivan Damgard. 1988. Multiparty Unconditionally Secure Protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 11–19. https://doi.org/10.1145/62212.62214

[22] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. 1995. Private Information Retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*. IEEE Computer Society, USA, 41.

[23] R Cleve. 1986. Limits on the Security of Coin Flips When Half the Processors Are Faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing* (Berkeley, California, USA) *(STOC '86)*. Association for Computing Machinery, New York, NY, USA, 364–369. https://doi.org/10.1145/12130.12168

[24] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *IACR Cryptology ePrint Archive*.

[25] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD Z2k: Efficient MPC Mod for Dishonest Majority. Springer-Verlag, Berlin, Heidelberg, 769–798. https://doi.org/10.1007/978-3-319-96881-0_26

[26] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 2183–2200. https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov

[27] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In *Computer Security – ESORICS 2013*, Jason Crampton, Sushil Jajodia, and Keith Mayes (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[28] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*. Springer-Verlag, Berlin, Heidelberg, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[29] Alex Davidson, Gonçalo Pestana, and Sofía Celi. 2023. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *Proc. Priv. Enhancing Technol.* 2023 (2023), 365–383.

[30] Jack Doerner and Abhi Shelat. 2017. Scaling ORAM for Secure Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 523–535. https://doi.org/10.1145/3133956.3133967

[31] Zeev Dvir and Sivakanth Gopi. 2016. 2-Server PIR with Subpolynomial Communication. *J. ACM* 63, 4, Article 39 (sep 2016), 15 pages. https://doi.org/10.1145/2968443

[32] Klim Efremenko. 2009. 3-Query Locally Decodable Codes of Subexponential Length *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 39–44. https://doi.org/10.1145/1536414.1536422

[33] Eric Fung, Georgios Kellaris, and Dimitris Papadias. 2015. Combining Differential Privacy and PIR for Efficient Strong Location Privacy. In *International Symposium on Spatial and Temporal Databases*.

[34] Craig Gentry and Shai Halevi. 2019. Compressible FHE with Applications to PIR. In *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11892)*, Dennis Hofheinz and Alon Rosen (Eds.). Springer, 438–464. https://doi.org/10.1007/978-3-030-36033-7_17

[35] Craig Gentry and Zulfikar Ramzan. 2005. Single-Database Private Information Retrieval with Constant Communication Rate. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming* (Lisbon, Portugal) *(ICALP'05)*. Springer-Verlag, Berlin, Heidelberg, 803–815. https://doi.org/10.1007/11523468_65

[36] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8441)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer, 640–658. https://doi.org/10.1007/978-3-642-55220-5_35

[37] Matthew Green, Watson Ladd, and Ian Miers. 2016. A Protocol for Privately Reporting Ad Impressions at Scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1591–1601. https://doi.org/10.1145/2976749.2978407

[38] Daniel Günther, Maurice Heymann, Benny Pinkas, and Thomas Schneider. 2022. GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 1759–1776. https://www.usenix.org/conference/usenixsecurity22/presentation/gunther

[39] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. 2016. Scalable and Private Media Consumption with Popcorn *(NSDI'16)*. USENIX Association, USA, 91–107.

[40] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. USENIX Security Symposium. (2023).

[41] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2005. Sufficient Conditions for Collision-Resistant Hashing. In *Theory of Cryptography Conference*.

[42] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. 2016. Secure Protocol Transformations. In *Advances in Cryptology – CRYPTO 2016*, Matthew Robshaw and Jonathan Katz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 430–458.

[43] Toshiya Itoh and Yasuhiro Suzuki. 2008. New Constructions for Query-Efficient Locally Decodable Codes of Subexponential Length. *ArXiv* abs/0810.4576 (2008).

[44] Jonathan Katz and Luca Trevisan. 2000. On the Efficiency of Local Decoding Procedures for Error-Correcting Codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing* (Portland, Oregon, USA) *(STOC '00)*. Association for Computing Machinery, New York, NY, USA, 80–86. https://doi.org/10.1145/335305.335315

[45] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 2651–2668. https://www.usenix.org/conference/usenixsecurity21/presentation/koti

[46] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. 2021. Tetrad: Actively Secure 4PC for Secure Training and Inference. *IACR Cryptol. ePrint Arch.* (2021), 755. https://eprint.iacr.org/2021/755

[47] E. Kushilevitz and R. Ostrovsky. 1997. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. 364–373. https://doi.org/10.1109/SFCS.1997.646125

[48] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2015. Riffle: An Efficient Communication System With Strong Anonymity. *Proceedings on Privacy Enhancing Technologies* 2016 (08 2015). https://doi.org/10.1515/popets-2016-0008

[49] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. 2022. Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE. *IACR Cryptol. ePrint Arch.* (2022), 1703. https://eprint.iacr.org/2022/1703

[50] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. 2022. Incremental Offline/Online PIR. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 1741–1758. https://www.usenix.org/conference/usenixsecurity22/presentation/ma

[51] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR : Private Information Retrieval for Everyone. *Proceedings on Privacy Enhancing Technologies* 2016 (2016), 155 – 174.

[52] Samir Menon and David J. Wu. 2022. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. *2022 IEEE Symposium on Security and Privacy (SP)* (2022), 930–947.

[53] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. 2011. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *Proceedings of the 20th USENIX Conference on Security* (San Francisco, CA) *(SEC'11)*. USENIX Association, USA, 31.

[54] Payman Mohassel, Mike Rosulek, and Ye Zhang. 2015. Fast and Secure Three-Party Computation: The Garbled Circuit Approach *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 591–602. https://doi.org/10.1145/2810103.2813705

[55] mpc msri. 2022. EzPC. https://github.com/mpc-msri/EzPC.

[56] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021).

[57] Peter Sebastian Nordholt and Meilof Veeningen. 2018. Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification. In *Applied Cryptography and Network Security*, Bart Preneel and Frederik Vercauteren (Eds.). Springer International Publishing, Cham, 321–339.

[58] T. Rabin and M. Ben-Or. 1989. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (Seattle, Washington, USA) *(STOC '89)*. Association for

Functionality 2: $\mathcal{G}$-oracle for Ds protocol

```
**Input:** $P_b, b \in \{0,1\}$, inputs $(i, \mathsf{inp}_b)$ where $i \in [1, n+1]$.
**Output:** $P_0, P_1$ learn $CW_i$ for $i \in [1,n]$ or $\gamma$ for $i = n+1$.
1 if $i \in [1,n]$ then
2     Parse $\mathsf{inp}_b = \left\{\langle r_i\rangle_b^{\mathsf{B}}, \langle z^{i,0}\rangle_b^{\mathsf{B}}, \langle z^{i,1}\rangle_b^{\mathsf{B}}, \langle \hat{z}^{i,0}\rangle_b^{\mathsf{B}}, \langle \hat{z}^{i,1}\rangle_b^{\mathsf{B}}\right\}$ and
        reconstruct the values from shares.
3     Compute $\sigma^i, \tau^{i,0}, \tau^{i,1}$ as follows:
          $\sigma^i \leftarrow z^{i,\overline{r_i}}$
          $\tau^{i,0} \leftarrow \hat{z}^{i,0} \oplus r_i \oplus 1$
          $\tau^{i,1} \leftarrow \hat{z}^{i,1} \oplus r_i$
4     Output $CW_i = \left\{\sigma^i, \tau^{i,0}, \tau^{i,1}\right\}$ to $P_0, P_1$.
5 else if $i = n+1$ then
6     Parse $\mathsf{inp}_b = \{T_b, W_b, \langle v\rangle\}$.
7     Let $q = 1$ if $T_1 > T_0$, $q = 0$ otherwise.
8     Compute $\gamma \leftarrow (-1)^q(v - W_0 + W_1)$.
9     Output $\gamma$ to $P_0, P_1$.
```

Computing Machinery, New York, NY, USA, 73–85. https://doi.org/10.1145/73007.73014

[59] Peter Rindal. 2022. cryptoTools. https://github.com/ladnir/cryptoTools.

[60] Julien P. Stern. 1998. A New Efficient All-Or-Nothing Disclosure of Secrets Protocol *(ASIACRYPT '98)*. Springer-Verlag, Berlin, Heidelberg, 357–371.

[61] Sergey Yekhanin. 2010. Private Information Retrieval. *Commun. ACM* 53, 4 (apr 2010), 68–73. https://doi.org/10.1145/1721654.1721674

A DPF EVAL PROTOCOL

For completeness, we give the Π_{Eval} protocol (Algorithm 3) from [13, 30] and use it to prove perfect correctness of Ds protocol in Lemma 3.6.

Parties $P_b, b \in \{0,1\}$ has DPF key k_b generated by Ds protocol corresponding to point function $f_{r,v}(r \in [N], v \in \mathbb{F})$ and evaluate the key on $x \in [N]$, $n = \log N$. Prg and Convert are pseduorandom functions defined in Section 3.1

Algorithm 3: $\Pi_{\mathsf{Eval}}(b, k_b, x)$

```
**Input:** Party $P_b$ has DPF key $k_b$ and evaluation point
       $x \in [N]$
**Output:** $P_b$ outputs $\langle f_{r,v}(x)\rangle_b \in \mathbb{F}$ i.e. arithmetic secret
       share of function value at $x$.
1 Parse $k_b = \left(\hat{S}_b^0, \hat{t}_b^0, \left\{\sigma^j, \tau^{j,0}, \tau^{j,1}\right\}_{j\in[1,n]}, \gamma\right)$.
2 for $j = 1$ to $n$ do
3     $(S_b^{j,0} \| t_b^{j,0} \| S_b^{j,1} \| t_b^{j,1}) = \mathsf{Prg}(\hat{S}_b^{j-1})$.
4     $\hat{S}_b^j = S_b^{j,x_j} \oplus (\hat{t}_b^{j-1} \cdot \sigma^j)$.
5     $\hat{t}_b^j = t^{j,x_j} \oplus (\hat{t}_b^{j-1} \cdot \tau^{j,x_j})$.
6 end
7 Output $\langle f_{r,v}(x)\rangle_b = (-1)^b[\mathsf{Convert}(\hat{S}_b^n) + \hat{t}_b^n \cdot \gamma]$.
```

B $\mathcal{G}$-ORACLE FOR DOERNER-SHELAT KEY GENERATION

The secure computation in step 5, 10 in Ds protocol (Algorithm 1) can be modeled as $\mathcal{G}$ (Functionality 2). To distinguish between computing correction words $\{CW_i\}_{i\in[1,n]}$ and γ, parties input $i \in [1, n+1]$. $\mathcal{G}$ outputs CW_i for $i \in [1,n]$ and γ for $i = n+1$. The notation for variables is same as that used in Ds protocol.

As is clear from the description, the only interaction between parties in Ds protocol is to compute outputs of $\mathcal{G}$ which is a two party non-cryptographic functionality.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009