Heterogeneous Private Information Retrieval

Hamid Mozaffari
University of Massachusetts Amherst
hamid@cs.umass.edu

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

***Abstract*—Private information retrieval (PIR) enables clients to query and retrieve data from untrusted servers without the untrusted servers learning which data was retrieved. In this paper, we present a new class of multi-server PIR protocols, which we call *heterogeneous PIR (HPIR)*. In such multi-server PIR protocols, the computation and communication overheads imposed on the PIR servers are non-uniform, i.e., some servers handle higher computation/communication burdens than the others. This enables heterogeneous PIR protocols to be suitable for a range of new PIR applications. What enables us to enforce such heterogeneity is a unique PIR-tailored secret sharing algorithm that we leverage in building our PIR protocol.**

We have implemented our HPIR protocol and evaluated its performance in comparison with regular (i.e., homogenous) PIR protocols. Our evaluations demonstrate that a querying client can trade off the computation and communication loads of the (heterogeneous) PIR servers by adjusting some parameters. For example in a two server scenario with a heterogeneity degree of $4/1$, to retrieve a 456KB file from a 0.2GB database, the rich (i.e., resourceful) PIR server will do 1.1 seconds worth of computation compared to 0.3 seconds by the poor (resource-constrained) PIR server; this is while each of the servers would do the same 1 seconds of computation in a homogeneous settings. Also, for this given example, our HPIR protocol will impose a 912KB communication bandwidth on the rich server compared to 228KB on the poor server (by contrast to 456KB overheads on each of the servers for a traditional homogeneous design).

I. Introduction

Private information retrieval (PIR) is a technique to provide query privacy to users when fetching sensitive records from untrusted databases. That is, PIR enables users to query and retrieve specific records from untrusted database(s) in a way that the serving databases can not identify the records retrieved. PIR algorithms have been suggested to be used in various application scenarios involving untrusted database servers [23], [39], [27], [31], [28], [14], [11], [44], [37], from retrieving Tor relay information [39] to privacy-preserving querying of location services [25] to registering Internet domains [23].

There are two major types of PIR protocols. The first type is *computational PIR* (CPIR) [16], [36], [2], [1], [12], [13], [22], [32], [38], [50], [4] in which the security of the protocol relies on the computational difficulty of solving a mathematical problem in polynomial time by the servers, e.g., factorization of large numbers. Most of the CPIR protocols are designed to be run by a single database server, and therefore to minimize privacy leakage they perform their heavy computations on the whole database (even if a single entry has been queried). Consequently, existing CPIR protocols suffer from very high computation overheads. The second major class of PIR is *information-theoretic PIR* (ITPIR) [18], [30], [8], [17], [24], [26], [7], [20]. ITPIR protocols provide information-theoretic security, however, existing designs need to be run on more than one database servers, and they need to assume that the servers do not collude. Existing ITPIR protocols impose lower computation overheads compared to CPIR algorithms, but at the price of requiring the non-collusion assumption for the servers. Therefore, (multi-server) ITPIR protocols are best fit to scenarios involving multiple (potentially competing) data owners who collaborate to run a service privately, therefore colluding is not in their best interest, e.g., [37], [39], [11]. Our work focuses on this class of PIR, i.e., multi-server PIR protocols.

Existing multi-server PIR protocols are homogeneous! The existing body of work on multi-server PIR considers a setting in which *the non-colluding PIR servers have similar computation and communication constraints*. We call such traditional multi-server PIR protocols *homogeneous*. Homogeneous PIR algorithms have been deployed in a wide range of homogeneous applications; this includes registering Internet domains [23], retrieving information of Tor relays [39], private media delivery [27], privacy-preserving e-commerce applications [31], private query in open-access eprint repositories [28], messaging applications [11], private online notification [44], and private file-sharing applications [37]. For instance, in PIR-Tor [39] the servers participating in the protocol are Tor directory servers with similar resources, in DP5 [11] the servers are messaging servers with similar network settings, and in rPIR [37] the servers are p2p file-sharing seeds with equal resources. In all of these applications, the proposed multi-server PIR protocols *impose symmetric computation and communication loads on all of the servers* involved in the multi-server PIR protocol.

Introducing heterogeneous multi-server PIR. In this paper, we introduce a new class of multi-server PIR, which we call *heterogeneous PIR (HPIR)*. An HPIR protocol is a multi-server PIR protocol with *asymmetric computation and communication constraints on its servers*, i.e., some of its servers handle higher computation/communication overheads than the others. We argue that *HPIR algorithms enable new applications for PIR, as well as improve the utility of some of the known applications of PIR*; this is because HPIR allows the participation of low-resource entities in running private services.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24363
www.ndss-symposium.org

Example application scenario for HPIR: Here we present an example application scenario for HPIR protocols: content privacy in content delivery networks (CDNs). Consider a content publisher, say, The New York Times (NYT): https://www.nytimes.com/, who uses a CDN provider (in this case, Fastly Inc.) to host its web service. For the CDN provider (Fastly) to be able to serve NYT's content, NYT will need to expose the activities of its viewers (e.g., what articles they read) to Fastly, which is usually implemented by NYT sharing its TLS private keys with Fastly. Obviously, this exposes the —potentially privacy-sensitive—actions of NYT viewers to Fastly (i.e., the third-party CDN provider).

As a solution to this problem, we suggest using a two-server PIR protocol, in which one PIR server is a CDN edge server (e.g., a Fastly server), and the other one is NYT's origin server. However, one of the main reasons that content publishers (e.g., NYT) use CDNs is to offload their computation and communication loads on CDNs. Therefore, for the presented two-server PIR model to be practical, it needs to impose much lower computation/communication costs on NYT's origin server compared to the costs imposed on CDN (Fastly) edge servers. No existing PIR protocol provides such heterogeneity, and therefore we argue for the design of heterogenous PIR protocols. We further elaborate on potential application scenarios of heterogeneous PIR in Section III.

Our technical approach. In this paper, we design the first HPIR construction. Similar to state-of-the-art multi-server PIR protocols [26], [30], [37], our protocol makes use of a secret sharing algorithm to split queries into shares, where the shares are sent to the multiple PIR servers. The PIR servers then perform some computations on the database based on the query shares they have received, and they send the results of the computation to the querier. Finally, the querier combines the responses from multiple PIR servers to retrieve the data record she had asked for. The main difference between our HPIR protocol and existing multi-server PIR protocols is that, in our scheme, different numbers of query shares are sent to different PIR servers. This results in different (i.e., heterogeneous) computation and communication overheads on different PIR servers.

Note that existing multi-server PIR protocols can *not* be trivially extended to heterogeneous constructions. One can modify a single-query PIR design like Goldberg's PIR [26] to a heterogeneous one by sending more than one query share to some of the PIR servers; however, this will increase the bandwidth/computation overhead on some of the servers (who receive multiple shares) *without reducing the overhead on any of the PIR servers*. The goal of HPIR is to reduce the overhead on resource-constrained servers, through increasing the overhead on resourceful servers. To be able to enforce heterogeneity in our HPIR protocol, we design a specific multi-secret sharing algorithm that enables us to split a query non-uniformly between multiple PIR servers. We call our secret sharing algorithm *PIR-tailored*, as it can only be used as part of a PIR protocol, but has no use in standard applications of secret sharing.

Note that we are not the first to use a multi-secret sharing algorithm for PIR protocols. Henry et al. [30] and Li et al. [37] have used multi-secret sharing algorithms for PIR protocols, with the intent of being able to send multiple PIR queries at each round of the protocol. Unfortunately, their multi-secret sharing algorithms are *ramp schemes*, which are not practical for typical applications of HPIR: a ramp secret sharing scheme requires the number of servers to be proportional to the number of shared secrets. However, most of the application scenarios of HPIR (as introduced in Section III) need to be deployed on two servers, as they comprise two non-colluding parties (e.g., a content publisher and a CDN provider). We therefore design a PIR-tailored multi-secret sharing algorithm that can be deployed on as few as two PIR servers regardless of the number of shares sent to each of the servers.

Therefore, the core of our HPIR algorithm, is our non-ramp, PIR-tailored multi-secret sharing algorithm. In our PIR-tailored secret sharing algorithm, any set of q secrets is shared using a q-degree polynomial, therefore requiring at least $q+1$ shares for reconstruction. Therefore, in each round of our PIR protocol, if the client intends to query for q records, she will need to send $q+1$ query shares to the multiple PIR servers participating in the protocol. The client can do this in a heterogeneous manner by sending different fractions of the $q+1$ shares to different servers, e.g., in a two-server setting, she can send 1 share to the resourceless server (e.g., NYT's origin server) and the other q shares to the resourceful server (e.g., Fastly's edge server). Consequently, the computation and communication overheads on different servers will be heterogeneous as it is proportional to the number of shares they receive and process.

Unlike previous secret-sharing based PIRs who use a single prime number, our protocol makes use of multiple prime numbers. This can increase the bandwidth of our protocol. We therefore introduce several novel techniques to improve the efficiency of our PIR protocol. In particular, unlike prior secret-sharing based PIR schemes, we use *random* x-coordinates in deriving the generated secret sharing polynomials. This reduces the upload and download bandwidth overhead of our protocol by reducing the element sizes of our PIR communications. Also, we reduce the upload bandwidth of the clients by using a pseudo random number generator (PRNG) in sending the client's queries to the PIR servers.

Similar to existing multi-server PIR protocols, our HPIR protocol provides information-theoretic security assuming that at least one server does not collude. However, if a PRNG is used for overhead reduction, our protocol's security will change to computational.

Implementation. We have implemented our HPIR algorithms in C++, wrapped in Rust. We have implemented our code to be compatible with the Percy++ PIR library [43]. We have compared its performance with the most relevant state-of-the-art PIR protocols and in different settings, e.g., for different database sizes, for different numbers of queries, and for various degrees of heterogeneity. We demonstrate that a querying client can trade off the computation and communication loads of the (heterogeneous) PIR servers by adjusting some parameters. For example in a two server scenario with a heterogeneity degree of $4/1$, to retrieve a 1.4MB file from a 2GB database, the rich (i.e., resourceful) PIR server will do 12.5 seconds worth of computation compared to 4.09 seconds by the poor (resource-constrained) PIR server; this is while each of the servers would

Table I: List of PIR notations

ℓ	Number of servers
t	Privacy threshold (max number of colluding servers)
k	Number of server's responses
$\mathbb{D}$	Database matrix
r	Number of rows in the database
s	Number of elements in each record of the database
w	Element size (bits)
N	Total size of the database (bits)

do the same 12.04 seconds of computation in a homogeneous settings. Also, for this given example, our HPIR protocol will impose 2.8MB communication bandwidth on the rich server compared to 724KB on the poor server (by contrast to 1.4MB overheads on each of the servers for a traditional homogeneous design).

Summary of contributions: In summary, we make the following contributions:

- We introduce a new class of multi-server PIR, called HPIR, which imposes non-uniform computation and communication overheads on the PIR servers. We motivate the importance of HPIR for various real-world applications.
- We design the first HPIR protocol which uses a PIR-tailored multi-secret sharing scheme at the core of its protocol.
- We have built a highly optimized implementation of our HPIR protocol in C++ (compatible with the Percy++ PIR library [43]) wrapped in Rust. We have evaluated the performance of our HPIR implementation for different database sizes, for different numbers of queries, and for various degrees of heterogeneity. Our code is available at https://github.com/SPIN-UMass/HPIR.

Organization: The paper is organized as follows. We start by overviewing the preliminaries of PIR and secret sharing algorithms in Section II. In Section III, we discuss potential application scenarios for heterogenous PIR. In Section IV, we present a PIR-tailored secret sharing algorithm, which is the core of our HPIR constructions. We overview the high-level ideas of our HPIR design in Section V, and we present the basic version and the complete version of our HPIR algorithm in Sections VI and VII, respectively. Finally, we present our implementation and evaluation results in Section VIII. The paper is concluded in Section IX.

II. Preliminaries

In this section, we provide background information on private information retrieval and secret sharing.

A. Preliminaries on PIR

In this section, we introduce the main concepts of PIR. Table I shows the notations we use for PIR protocols.

Database as a Matrix: In a PIR protocol, one or multiple servers, called *PIR servers*, host a database $\mathbb{D}$, which can be represented as an r-by-s matrix over a finite field $\mathbb{F}$. The goal of a *client* (*querier*) is to retrieve one row of $\mathbb{D}$, called a *data record*, through some interactions with the PIR servers in a way that the PIR servers do not learn which record of $\mathbb{D}$ was retrieved by the client.

$$\mathbb{D} = \begin{bmatrix} \mathbb{D}_{1,1} & \mathbb{D}_{1,2} & \dots & \mathbb{D}_{1,s} \\ \mathbb{D}_{2,1} & \mathbb{D}_{2,2} & \dots & \mathbb{D}_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{D}_{r,1} & \mathbb{D}_{r,2} & \dots & \mathbb{D}_{r,s} \end{bmatrix} \quad (1)$$

Non-private Information Retrieval: Suppose that the client aims at retrieving the jth record of the database. She will create a unit vector $\vec{e}_j$ of size r where all the bits are set to zero except the jth position being set to one:

$$\vec{e}_j = [0 \quad 0 \quad \dots \quad 1 \quad \dots \quad 0 \quad 0] \quad (2)$$

If the client did not care about privacy, she would send $\vec{e}_j$ to the server(s), and the server(s) could generate the client's response by multiplying the vector into the database matrix $\mathbb{D}$:

$$\vec{e}_j.\mathbb{D} = [\mathbb{D}_{j,1} \quad \mathbb{D}_{j,2} \quad \dots \quad \mathbb{D}_{j,s}] \quad (3)$$

Private Information Retrieval: A PIR technique allows the client to obtain this response without revealing $\vec{e}_j$ to the server(s). Existing PIR techniques use two main approaches to obfuscate $\vec{e}_j$: (a) Homomorphic encryption: In such protocols [50], [36], [2], [1], [4], the client encrypts $\vec{e}_j$ element by element before being sent to the servers. During the data recovery phase, the client will extract her intended record by decrypting the components of $\vec{e}_j \times \mathbb{D}$. (b) Secret Sharing: In other PIR protocols [26], [30], [29], [20], [37], the client will use secret sharing to generate different vector shares for $\vec{e}_j$, and she will send the shares to the PIR servers. Most of the existing single-server, CPIR protocols use homomorphic encryption, and most of the existing multi-server, ITPIR protocols use secret sharing.

Key PIR Designs: Here we overview the two key multi-server PIR designs that are the most relevant to our work. We present a more comprehensive overview of other PIR designs in Appendix A.

Goldberg ITPIR using Shamir Secret Sharing. Goldberg's PIR [26] is an ITPIR scheme. The client uses Shamir's secret sharing [48] to split the unit vector $\vec{e}_j$ into ℓ shares (each a vector of size r), where the shares are sent to ℓ servers. Each server will send back the multiplication of its received share into the database matrix $\mathbb{D}$. Finally, the client will interpolate the query responses component-wise at $x = 0$ to extract her interested row of the database.

Henry et al. ITPIR using Ramp Secret Sharing. Henry et al. [30] modify Goldberg's PIR [26] by replacing Shamir's $(t+1, \ell)$-threshold secret sharing with a $(t+1, q, \ell)$-ramp secret sharing [10]. This enables a client to encode q secrets in a $(t+q-1)$ degree polynomial, as opposed to only one secret in a t degree polynomial in [26]; therefore, the protocol is able to query multiple queries from the PIR servers at any round of the PIR protocol. To query the q records, the client will receive $(t+q)$ responses from the PIR servers.

Table II: List of notations used in secret sharing schemes

ℓ	Number of participants
t	Privacy threshold
ρ	Large prime number
q	Number of secrets
$\mathbb{S} = \{s_1, s_2, \ldots, s_q\}$	Secrets

B. Preliminaries on Secret Sharing

The goal of secret sharing is to split a secret into multiple *shares* (e.g., by a trusted *dealer*) such that the secret can be reconstructed by combining a certain number of the shares. The dealer distributes these shares among multiple *shareholders* who participate in the protocol. A $(t+1, \ell)$ *threshold secret sharing scheme* distributes a secret among ℓ shareholders in a way that any coalition of up to t shareholders can not learn anything about the secret, while a coalition of more than t shareholders can fully reconstruct the secret. A scheme is called *multi-secret sharing* [54], [15], [42] if it shares multiple secrets in each round of the protocol.

Notations: Table II lists the notations we use for secret sharing algorithms.

Key Secret Sharing Designs: We introduce two secret sharing schemes that have been the basis of state-of-the-art PIR protocols. Our secret sharing algorithm, introduced later, is built upon these schemes.

Shamir Secret Sharing. Shamir's scheme [48] is a $(t+1, \ell)$-threshold scheme, in which the shares are the points of a polynomial function. Specifically, a secret s is shared as follows:

I The dealer chooses a random polynomial function $f(x) \in_r \mathbb{F}_\rho$ of degree t, where $f(0) = s$ is the secret.
II The dealer chooses ℓ different non-zero x-coordinates $\{x_1, \ldots, x_\ell\}$ uniformly at random, where $x_i \in_r \mathbb{F}_\rho$ for $1 \leq i \leq \ell$.
III The dealer sends $(x_i, f(x_i))$ to the i^{th} shareholder.

Any coalition of $k > t$ shareholders can recover the secret s from their shares by using Lagrange polynomial interpolation. Therefore, given $k > t$ shares $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ the shared secret is reconstructed as

$$s = \sum_{m=1}^{k} y_m \Big(\prod_{n=1, n\neq m}^{k} x_n (x_n - x_m)^{-1} \Big) \; mod(\rho) \qquad (4)$$

On the other hand, for a coalition of $k \leq t$ shareholders, any $s \in \mathbb{F}_\rho$ has the same probability of being the secret.

Ramp Variant of Shamir Secret Sharing. While a Shamir $(t+1, \ell)$ threshold scheme shares only one secret using a t-degree polynomial, a $(t+1, q, \ell)$-ramp secret sharing [10] uses a $(t+q-1)$-degree polynomial to share q secrets simultaneously with the privacy level of t. That is, the dealer shares q secrets $\{s_1, \ldots, s_q\}$ from $\mathbb{F}_\rho$ among ℓ participants in a way that any coalition of $t+q$ or more participants can retrieve *all* of the q secrets, but any set of participants up to t cannot learn anything about the secrets. However, for $t < k < t+q$ participants, the joint distribution of the secrets is not uniform, therefore it leaks information about the secrets. The dealer takes the following steps to share q secrets $\{s_1, \ldots, s_q\}$:

I Chooses $\{y_{q+1}, \ldots, y_{q+t}\}$ randomly from $\mathbb{F}_\rho$
II Finds a polynomial $f(x)$ with degree of at most $t+q-1$ that contains the following points:

$$(1, s_1), \ldots, (q, s_q), (q+1, y_{q+1}), \ldots, (q+t, y_{q+t})$$

III Sends secret share $(x_i, f(x_i))$ to the i^{th} shareholder for $1 \leq i \leq \ell$ (x_is are random, non-zero and different numbers from $\mathbb{F}_\rho$).

To retrieve the secrets, any $k \geq t+q$ shares can give away the secrets using Lagrange interpolation:

$$s_j = \sum_{m=1}^{k} y_m \Big(\prod_{n=1, n\neq m}^{k} (j - x_n)(x_m - x_n)^{-1} \Big) \; mod(\rho) \quad 1 \leq j \leq q \qquad (5)$$

III. Introducing Heterogeneous PIR

A PIR protocol is either single-server or multi-server. The security of multi-server PIR relies on assuming that its multiple PIR servers do not collude; this allows multi-server protocols to impose lower computation overheads than single-server protocols. Consequently, single-server and multi-server protocols are suited to different application scenarios. Note that this is a different classification than computational PIR (CPIR) versus information-theoretic PIR (ITPIR), but all single-server PIR protocols fall in the category of computational PIR (CPIR) [16], [36], [2], [1], [12], [13], [22], [32], [38], [50], [4], as proved by Chor et al. [17].

Existing multi-server PIR constructions [30], [8], [17], [24], [26], [55], [7], [20], [18] impose *uniform* computation and communication overheads on their (non-colluding) multiple PIR servers; therefore, we call them homogenous. In this paper, we introduce *heterogeneous PIR (HPIR)*,[1] which is a subclass of multi-server PIR protocols. An HPIR protocol is a multi-server PIR protocol with *asymmetric computation and communication constraints on its servers*. That is, some of its servers (called *rich servers*) handle higher computation/communication overheads than the others (called *poor servers*).

A. Potential Applications Scenarios

We believe that HPIR protocols will enable new application scenarios for multi-server PIR, as well as improve the usability of some of the known applications of PIR. To support this claim, in this section we present several potential application scenarios for heterogeneous PIR algorithms.

Note that for some of these applications, one could instead use a single-server PIR, however, existing single-server PIRs are too slow for most of the in-the-wild applications. Also, note that we only present the intuitions on why HPIR will fit

[1]Note that some previous work [9] uses HPIR to refer to hybrid PIR, another class of PIR.

these application scenarios; integrating HPIR in each of the following scenarios will require additional engineering efforts (e.g., to synchronize the PIR servers), which is out of our scope.

Finally, note that in our HPIR applications, we assume the HPIR servers to be *non-trusted*. In a given HPIR application, the PIR database itself is not privacy-sensitive, but the users' accesses to the records of the database is privacy-sensitive. Therefore, the PIR database can be replicated on non-trusted servers (like CDN servers). On the other hand, we assume the PIR servers to *not collude* with each other. As discussed later, this is a realistic assumption as, in our applications, the HPIR servers have no motivation to collude.

1) Privacy-preserving Content Delivery: Content publishers increasingly use Content Delivery Networks (CDNs) to improve the security and performance of their services. However, to do so, they have no choice but to expose their clients' activities to the third-party CDN operators they use (normally, by sharing their TLS private keys with their CDN providers). For instance, a CDN provider (say, Fastly) hosting the website of an online newspaper (say, the New York Times) will be able to see which articles are viewed by any particular visitor of that online newspaper. Note that, in this scenario, the privacy-sensitive information is not the hosted web content (e.g., all of the NYT articles are public) but the *metadata* of accessing such (public) content by the users. The problem is applicable to similar CDN-hosted services, e.g., a patents database, a certificate authority, a software updates service, etc.

We suggest to deploy PIR on CDN servers to enable privacy-preserving content delivery. Existing single-server PIR protocols are too slow to be used in this application (and most other proposed applications of PIR); we therefore suggest a heterogeneous 2-servers PIR protocol to be used for this application. This is illustrated in Figure 1: the CDN edge servers act as the "rich" servers and the content publisher's origin servers act as the "poor" servers. Our heterogenous setting is ideal for this application: The CDN edge servers (e.g., Fastly edge servers) are often much closer to the clients and are designed to be capable of handling very large traffic volumes. By contrast, content publisher servers (e.g., www.nytimes.com/'s origin servers) aim to minimize their communication and computation costs (in fact, this is one of the key reasons for using CDNs for content hosting). As mentioned before, *all existing multi-server PIR protocols are homogeneous*. This scenario demonstrates the need for heterogeneous PIR protocols that impose *non-uniform computation and communication loads* on the multiple PIR servers running the protocol.

Note that like standard multi-server PIR protocols, we assume the HPIR servers to be *non-colluding*. This is a realistic assumption, since in our mentioned applications of HPIR, the servers have no motivation to collude! For instance, in the CDN application of HPIR presented here, the origin servers (owned by the New York Times) have no reason to collude with the CDN HPIR servers, as this will compromise the privacy of their users.

2) Private P2P File Sharing: Various popular services such as Spotify, PPTV, and BitTorrent use their clients for content distribution. Recent work [37] has suggested to use PIR to protect privacy in such services, particularly for BitTorrent and Spotify. We argue that an HPIR protocol will significantly improve the usability of PIR for such applications. This is because in these systems, the peers are located in diverse geographic locations, and have different computation and communication resources. Therefore, when a heterogeneous PIR is deployed in this setting, a client can obtain larger traffic volumes from nearby seeding peers compared to distant peers (while protecting privacy through PIR), therefore improving the overall download experience.

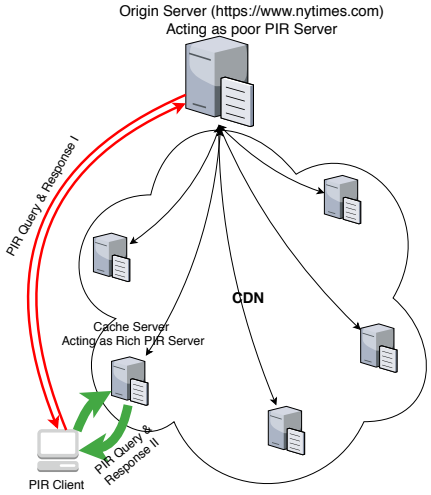


Figure 1: Illustrating how a heterogeneous PIR scheme can enable private content delivery by CDNs.

3) Query Privacy in Cache Networks: A multitude of next-generation network architectures cache content objects to improve the overall utility of the network, e.g., Named Data Networking (NDN) [56], Publish-Subscribe Internet Routing Paradigm (PSIRP) [51], Data Oriented Network Architecture (DONA) [34], Network of Information (NetInf) [3], and MobilityFirst [52]. A key privacy challenge to the design of cache networks like NDN is the privacy of queries against cache routers. That is, a cache router will learn the content names requested by a client in order to be able to serve her. We propose to use PIR as a mechanism to enforce cache privacy in cache networks. Our proposal is to have the cache routers serve as PIR servers, and store named objects into a PIR database. A client interested in a particular named object will need to query the cache routers through a PIR protocol in order to preserve her query privacy. We therefore suggest multi-server PIR protocols to be used for this application. However, existing multi-server designs rely on the assumption that PIR servers should not collude. Therefore, the two (or more) cache routers queried by a client should be under different jurisdictions, i.e., run by competing Internet entities. For instance, in a 2-server setting, the first PIR server queried by the client can be the edge router of the client, and the second router can be a router in a non-peer AS or the content publisher itself (therefore non-colluding with the edge router).

As can be seen, in this setting, the edge router (e.g., the client's default gateway) can tolerate much higher bandwidth

and computation burden than the distant router/publisher—in fact, the whole purpose of information centric networks is to reduce transition loads by caching content on local routers. Therefore, the deployed PIR protocol needs to be heterogenous.

IV. Our PIR-Tailored Secret Sharing Algorithm

In this section, we define and design a PIR-tailored secret sharing scheme that we use in the design of our PIR protocol.

Why a new scheme? Similar to the state-of-the-art multi-server PIR schemes [26], [30], [37], our proposed HPIR scheme uses secret sharing to split the query vector between PIR servers. To enable heterogeneity, an HPIR protocol needs to use a *multi-secret sharing* algorithm, as introduced earlier. This will allow an HPIR protocol to split PIR computations and communications unevenly between PIR servers. Note that some prior PIR protocols by Henry et al. [30] and Li et al. [37] have used multi-secret sharing algorithms. However, a ramp secret sharing algorithm is not suitable for heterogeneous PIR: in a ramp secret sharing scheme, the number of required servers increases with the number of shared secrets; therefore, for an HPIR protocol based on a ramp secret sharing scheme, the number of PIR servers increases with the degree of heterogeneity. However, most of the practical application scenarios of HPIR (as introduced in Section III) need to be deployed on two servers, as they comprise two non-colluding parties (e.g., a content publisher and a CDN provider). Therefore, *we design a PIR-tailored multi-secret sharing algorithm in which the number of shareholders does not increase with the number of shared secrets*. This allows our HPIR protocol that uses this PIR-tailored secret sharing to be run by as few as only two servers, regardless of the degree of heterogeneity.

A. Introducing PIR-tailored Secret Sharing

A PIR-tailored secret sharing algorithm is one that can be used as part of a PIR protocol, but has no use in traditional applications of secret sharing algorithms. In particular, unlike standard secret sharing algorithms, in a PIR-tailored secret sharing, the shareholders can *not* recover the secrets even if they all collude; the shareholders can only use their shares to perform some computation (i.e., on the PIR database) and send the results to the dealer. Also, unlike standard secret sharing algorithms, our PIR-tailored secret sharing only shares secrets from $\{0,1\}$; this is because in PIR applications, the client's (secret) query for each database record has one of the $\{0,1\}$ values, where 1 implies interest in retrieving that database record. Finally, in PIR-tailored secret sharing, only one of the secrets in each PIR-tailored polynomial can be 1, and the rest are 0s.

Before presenting our PIR-tailored secret sharing algorithm, we summarize the main differences of our PIR-tailored secret sharing algorithm from standard secret sharing algorithms:

1) In standard secret sharing, a dealer shares value(s) from $\mathbb{F}_\rho$ where ρ is a prime number. However, in PIR-tailored secret sharing, the dealer is sharing value(s) from $\{0,1\}$.
2) In a standard multi-secret sharing algorithm, the dealer has no constraint on choosing the values of the secrets. However, in PIR-tailored secret sharing, only one of the secrets can be 1, and the rest should be 0s.
3) In standard secret sharing, the x-coordinates of the points used for constructing the secrets sharing polynomial is known to the public as $X = \{1, 2, \dots\}$; however, in PIR-tailored secret sharing these values are secret and only known to the dealer.
4) In standard secret sharing, the x-coordinates used in generating the share for shareholder i is known to that specific shareholder, i.e., the ith shareholder knows $(x'_i, f(x'_i))$; however, in PIR-tailored secret sharing these values (x'_i) are secret from all shareholders, and only known to the dealer.

B. Algorithm Details

Parameters Suppose the dealer plans to share q secrets, $\mathbb{S} = \{s_1, s_2, \dots, s_q\}$, among ℓ participants with a threshold $t \leq \ell$. All the secrets are from $\{0,1\}$, and only one of the secrets could be 1. At the first step, the dealer generates q prime numbers $\mathbb{P} = \{p_1, p_2, \dots, p_q\}$ at random. Then, the dealer calculates $n = p_1 \times p_2 \times \cdots \times p_q$. Note that our shares are about q times of prior works since all the calculations are in $mod(n)$.

Initial phase: At the start of the protocol, the dealer creates the $\mathbb{P}$ and n parameters and announces value of n publicly.

Sharing Secrets: In each round of the protocol, the dealer shares q secrets with the shareholders by taking the following steps (we first describe this for $q \leq t$, and then will discuss the modifications for $q > t$). Our scheme is based on the ramp secret sharing introduced in Section II-B, so we discuss the modifications with respect to that scheme. For simplicity, we use $\mathbb{X} = \{1, \dots, t+1\}$ as the x-coordinates of the root points, but they can be chosen randomly from $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid gcd(n,x) = 1\}$. Like other systems working based on Lagrange interpolation, we have the constraint that $gcd(x_i - x_j, n) = 1$ for $x_i, x_j \in \mathbb{X}$ and $i \neq j$.

I In contrast to the ramp scheme that uses points (i, s_i), our scheme uses the points $(i, r_i \times p_i + s_i)$ to build our secret sharing polynomial function. $r_i \in \mathbb{Z}_n^*$ are random numbers that increase the degree of freedom of the secrets. Therefore, the dealer uses Lagrange polynomial interpolation to find a polynomial $f(x)$ (with a degree of at most t) that contains these $t+1$ points:

$$(1, (r_1 \times p_1) + s_1 \ mod(n)), \dots, (q, (r_q \times p_q) + s_q \ mod(n)) \dots$$

$$(q+1, r_{q+1} \ mod(n)), \dots, (t+1, r_{t+1} \ mod(n))$$

II The dealer will send the secret share $(f(x_i))$ to the ith shareholder for $1 \leq i \leq \ell$. ($x_i \in \mathbb{X}'$s are random numbers from $\mathbb{Z}_n^*$ with Lagrange constraint $gcd(x'_i - x'_j, n) = 1$ for $x'_i, x'_j \in \mathbb{X}'$ and $i \neq j$ and a constraint that $gcd(x_i - x'_j, n) = 1$ for $x_i \in \mathbb{X}$ and $x'_j \in \mathbb{X}'$.) $\mathbb{X}'$ is the set of x-coordinates used to generate shares.

Secret reconstruction: To retrieve the secrets, any $k \geq t+1$ shares can give away the secrets using Lagrange interpolation:

1) The combiner will construct the polynomial $f(x)$ using the $k \geq t+1$ points of $(x'_j, f(x'_j))$ for $x'_j \in \mathbb{X}'$ with Lagrange polynomial interpolation.
2) The combiner uses $f(x)$ to obtain the secrets. Suppose the combiner wants to recover the ith secret, s_i. As $(i, (r_i \times p_i) + s_i \ mod(n))$ is a point of the known $f(x)$, she can extract s_i using the p_i:

$$\begin{aligned} s_i &= f(i) \ mod(p_i) \\ &= \sum_{m=1}^{k} f(x'_m)(\prod_{n=1, n\neq m}^{k} (i - x'_n)(x'_m - x'_n)^{-1}) \ mod(p_i) \end{aligned} \quad (6)$$

Extension to $q > t$: The protocol will operate with small changes. $f(x)$ will have a degree of at most q. Therefore, in the first step of the sharing protocol, the dealer will use the points $(1, (r_1 \times p_1) + s_1 \ mod(n)), \ldots, (q, (r_q \times p_q) + s_q \ mod(n)), (q+1, r_{q+1} \ mod(n))$ to construct $f(x)$. The dealer releases $q - t$ random points of $f(x)$ publicly to make the scheme a $(t+1, \ell)$ threshold secret sharing algorithm. The rest of the protocol is the same.

C. Security Analysis

Degree of freedom of secrets The degree of freedom of shared secrets demonstrates how many *independent variables* are used in generating the secret sharing function. A multi-secret sharing scheme is secure if the q shared secrets $\{s_1, \ldots, s_q\}$ have a degree of freedom of q or higher when up to t shares are available. Each secret share known to the adversary reduces the degree of freedom by one, as it provides the adversary with a new equation for the main polynomial. Therefore, if there are d independent variables in $f(x)$, the degree of freedom given t secret shares will be $d - t$.

Increasing the degree of freedom by introducing new random variables Our PIR-tailored secret sharing scheme shares q secrets using a t degree polynomial. We add q random variables, r_is, to the polynomial points we share, i.e., $(1, (r_1 \times p_1) + s_1 \ mod(n)), \ldots, (q, (r_q \times p_q) + s_q \ mod(n))$. Therefore, the adversary's degree of freedom to reconstruct the secrets will be $t + q + 1$. Assuming that the adversary is provided with t shares, the degree of freedom will reduce to $q + 1$, which is still larger than the number of secrets, q.

The $q > t$ state of the PIR-tailored secret sharing is used as the core of our HPIR since we want to deploy our HPIR with minimum number of PIR servers (e.g., two servers with $t = 1$). Therefore, when $q > t$, we claim that any coalition with up to q shares can not learn *anything* about the secrets. Note that this is unlike the ramp secret sharing which leaks some information if the number of shares is between $t+1$ and $t + q - 1$. The complete proof is provided in Appendix B.

V. Sketch of our HPIR Protocol

In this section we present the core design of our HPIR protocol. The core of our HPIR construction is the PIR-tailored multi-secret sharing algorithm introduced in Section IV.

The high-level ideas of our HPIR protocol Our HPIR protocol has the same high-level architecture as Henry et al.'s multi-server PIR [30]. The querying client will act as the secret sharing dealer, and the PIR servers act as the shareholders. The client will use the PIR-tailored secret sharing algorithm of Section IV to split queries into shares, which are then sent to the servers. The servers will make some computation using the query shares and will send the results back to the querying client. Finally, the client recovers her requested records by combing the responses from the PIR servers. Like previous information-theoretic PIR systems, we assume that all the PIR servers are not colluding, so they cannot reconstruct the secret sharing polynomials to recover the secrets.

Our HPIR protocol is a multi-query protocol, i.e., the client queries multiple (q) records in each round of the protocol. The client will generate a polynomial $f_i(x)$ for *each record* of the database. Each polynomial is used to share q secrets with the possible values of 0 or 1. A value of 1 means that the client is asking for the record corresponding to the index of that polynomial. To query for q records in a given round of the protocol, the client will send $q + 1$ vectors of size r elements to the PIR servers to retrieve q records of the database.

Note that the key enabler of heterogeneity in our HPIR is the PIR-tailored secret sharing scheme of Section IV. To enforce heterogeneity, the client simply sends a different fraction of query shares to different servers based on their bandwidth and computational capabilities. For instance, consider a three-server setting, where the three servers plan to handle 30%, 10%, and 60% of the overall communication/computation overheads, respectively. Therefore, the *normalized* resources of the three servers are 3, 1, 6, respectively (we normalize by dividing by the smallest number). For this setting, the client will choose $q = (3 + 1 + 6) - 1 = 9$. Then, to retrieve $q = 9$ records of the database, the client will send 3 (out of total $q + 1 = 10$) of her query shares to the first server, 1 (out of total $q + 1 = 10$) of the shares to the second server, and the rest of them to the third server. The non-ramp property of our PIR-tailored secret sharing scheme enables us to design multi-query PIR algorithms that can operate using as few as two PIR servers.

Two Versions To better present the technical details of our HPIR protocol, we present a basic version and a complete version for our HPIR protocol. In our basic HPIR (Section VI), there are q prime numbers involved in generating the queries, so the communication cost will increase with the number of queries linearly; we address this in our complete version (Section VII) by introducing additional parameters.

VI. Our HPIR Algorithm (Basic Version)

In the following, we present the steps of our HPIR protocol. For clarity of presentation, we present our protocol for a 2-server PIR setting (composed of a rich server and a poor server). Please refer to Table I for the notations.

A. Client Generates r Polynomials

Suppose that the client wants to query q records of the database with indices $\beta = \{\beta_1, \ldots, \beta_q\}$; she takes the following steps to generate r polynomials (one for each row of the database):

1) The client will choose $q > 2$ different prime numbers $P = \{p_1, p_2, \ldots, p_q\}$ greater than 2^w, where w is the element size in the database. The client will calculate $n = p_1 \times p_2 \times \cdots \times p_q$, and will send it to the PIR servers (note that all the calculations in this protocol are in $mod(n)$).
2) The client will construct r polynomials of degree q based on our PIR-tailored secret sharing algorithm (see Section IV-B). In generating each of the polynomials, the client will choose $q+1$ points with random distinct x-coordinates $\mathbb{X} = \{x_1, x_2, \ldots, x_{q+1}\}$ (used for all the polynomials) and y-coordinates given by (for $1 \leq i \leq r$):

$$y_{i,j} = f_i(x_j) = \begin{cases} (r_{i,j} \times p_j) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q \\ r_{i,j} \ mod(n) & \text{for } j = q+1 \end{cases} \tag{7}$$

where $r_{i,j}$s are random numbers from $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid gcd(n, x) = 1\}$, and the secrets are:

$$\delta_{ij} = \begin{cases} 1 & i = \beta_j \\ 0 & \text{o.w.} \end{cases} \tag{8}$$

where $\beta = \{\beta_1, \ldots, \beta_q\}$ are the indices of the data records being queried by the client.
3) Finally, after choosing these points, the client uses Lagrange interpolation to find the r polynomials of degree q that contain these points.

Constraints All the members of $\mathbb{X}$ are chosen from $\mathbb{Z}_n^*$ at random. Like other systems working based on Lagrange interpolation, we have the constraint that $gcd(x_i - x_j, n) = 1$ for $x_i, x_j \in \mathbb{X}$, $i \neq j$.

Example: Suppose we have five records in our database ($r = 5$), and the client wants to retrieve records with indices $\beta = \{1, 3, 4\}$. Each row of the following matrix shows the y-coordinates of each of the $r = 5$ polynomials ($r_{i,j} \in_r \mathbb{Z}_n^*$ for $1 \leq i \leq r, 1 \leq j \leq q+1$):

$$\mathbb{Y} = \begin{bmatrix} (p_1 \times r_{1,1}) + 1 & p_2 \times r_{1,2} & p_3 \times r_{1,3} & r_{1,4} \\ p_1 \times r_{2,1} & p_2 \times r_{2,2} & p_3 \times r_{2,3} & r_{2,4} \\ p_1 \times r_{3,1} & (p_2 \times r_{3,2}) + 1 & p_3 \times r_{3,3} & r_{3,4} \\ p_1 \times r_{4,1} & p_2 \times r_{4,2} & (p_3 \times r_{4,3}) + 1 & r_{4,4} \\ p_1 \times r_{5,1} & p_2 \times r_{5,2} & p_3 \times r_{5,3} & r_{5,4} \end{bmatrix} \tag{9}$$

B. Client Generates Queries

Using the r polynomials generated above, the client will generate secret shares for her q queries. To do so, as described in Section IV-B, the client will pick $(q+1)$ random x-coordinates of $\mathbb{X}' = \{x'_1, x'_2, \ldots, x'_{q+1}\}$ (different from $\mathbb{X}$, with the same constraint), which are kept secret from the server, to generate the query matrices. $\mathbb{Q}_c$ is the query matrix for the rich (resourceful) server with q rows and r columns, and $\mathbb{Q}_r$ is the query matrix for the poor (low-resource) server with one row and r columns:

$$\mathbb{Q}_c = \begin{bmatrix} \vec{F}(x'_1) \\ \vec{F}(x'_2) \\ \vdots \\ \vec{F}(x'_q) \end{bmatrix} = \begin{bmatrix} f_1(x'_1) & f_2(x'_1) & \ldots & f_r(x'_1) \\ f_1(x'_2) & f_2(x'_2) & \ldots & f_r(x'_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x'_q) & f_2(x'_q) & \ldots & f_r(x'_q) \end{bmatrix} \tag{10}$$

$$\mathbb{Q}_r = \left[\vec{F}(x'_{q+1})\right] = \begin{bmatrix} f_1(x'_{q+1}) & f_2(x'_{q+1}) & \ldots & f_r(x'_{q+1}) \end{bmatrix} \tag{11}$$

where $f_i()$ is the polynomial function corresponding to the record i. The client sends the query matrices $\mathbb{Q}_r$ and $\mathbb{Q}_c$ to the servers.

C. The Servers Respond

After receiving the query matrices, each server will calculate the multiplication of $\mathbb{R}_{c/r} = \mathbb{Q}_{c/r} \times \mathbb{D} \ mod(n)$, and it will return the results to the client. Note that the servers do not know the values of r_{ij}, $\mathbb{X}'$, $\mathbb{X}$, and the prime numbers (p_js).

D. Reconstructing the Records by the Client

Using the responses received from the servers, $\mathbb{R} = \mathbb{R}_c || \mathbb{R}_r$, the client will construct s polynomials $\phi_k(x)$. Each ϕ_k is a q-degree polynomial that produces the kth elements of the q queried records.

Each column k of $\mathbb{R}$ contains $q+1$ points for a polynomial $\phi_k(x)$, i.e., the points $(x'_j, \mathbb{R}_{j,k})$ for $1 \leq j \leq (q+1)$. Each $\mathbb{R}_{j,k}$ (for $1 \leq j \leq q+1$ and $1 \leq k \leq s$) is given by:

$$\begin{aligned} \mathbb{R}_{j,k} &= \sum_{i=1}^{r} f_i(x'_j) \times D_{i,k} \ mod(n) \\ &= \sum_{i=1}^{r} (D_{i,k} \times \sum_{v=1}^{q+1} (a_{i,v} \times x_j'^{v-1})) \ mod(n) \\ &= \sum_{v=1}^{q+1} (x_j'^{v-1} \times \sum_{i=1}^{r} (a_{i,v} \times D_{i,k})) \ mod(n) \\ &= \phi_k(x'_j) \ mod(n) \end{aligned} \tag{12}$$

where $\phi_k()$ is a polynomial of degree q, and $a_{i,v}$ is the v^{th} coefficient of the polynomial $f_i(x)$. ϕ_k has a degree of q, and therefore the client can derive it using Lagrange interpolation by using the $q+1$ points $(x'_j, \mathbb{R}_{j,k})$ for $1 \leq j \leq (q+1)$.

Finally, the client retrieves the responses to her q queries using the derived $\phi.(\cdot)$ polynomials by feeding the x-coordinates $\mathbb{X} = \{x_1, \ldots, x_q\}$ into $\phi_k(x_j)$. Specifically, the client derives the kth element of the jth queried record, $\mathbb{D}_{\beta_j,k}$, as:

$$D_{\beta_j,k} = \phi_k(x_j) \ mod(p_j) \text{ for } 1 \leq k \leq s, 1 \leq j \leq q \tag{13}$$

E. Communication Overhead

Since the client uses q prime numbers (one for each query), the upload and download overheads are linear with the number of queries. To retrieve q records, the client should send $q^2 \times r \times w$ bits to the rich server, and $q \times r \times w$ bits to the poor server. The rich server will send back $q^2 \times s \times w$ bits, and the poor server will send back $q \times s \times w$ bits.

F. Security

This PIR protocol is built on our PIR-tailored secret sharing scheme of Section IV, so its security is based on the underlying PIR-tailored secret sharing scheme. Our HPIR protocol uses r PIR-tailored secret sharing functions (one for each database record), and retrieves q secrets at each round. To provide

information-theoretic security, the set of q secrets should have a degree of freedom more than $q \times r$. In a two-servers setting, the rich server will know $q \times r$ points of these PIR-tailored secret sharing schemes, so the number of independent variables in the system should be more than $2 \times q \times r$. There would be $(2q+1) \times r + 2q + 2$ random variables inside the system: $(q+1) \times r$ variables of r_{ij}, $q \times r$ secret values of δ_{ij}, $q+1$ random variables of $\mathbb{X}$, and $q+1$ random variables of $\mathbb{X}'$. If the poor server does not collude with the rich server, the degree of freedom of secrets will be more than the $q \times r$ threshold. Therefore, our HPIR is information-theoretic secure if at least one of the PIR servers does not collude with the others.

VII. Our HPIR Algorithm (Complete Version)

Why extending the design As mentioned above, the basic version of our HPIR protocol has high communication costs due to using q prime numbers (one for each query). Our extended protocol uses only two prime numbers $\mathbb{P} = \{p_1, p_2\}$ for query construction, therefore reduces communication costs significantly. To preserve its information-theoretic security, we add multiple parameters to its polynomials. To do so, we make the x-coordinates of the PIR-tailored secret sharing polynomials unique for each row of the database, and unknown to the servers.

Improving efficiency by introducing unique x-coordinates for each PIR-tailored secret sharing function Recall from Section VI-D that the constructed PIR-tailored secret sharing polynomials, $\phi_k(\cdot)$s, are functions of client polynomials $f_i(x)$ and database elements, i.e., $\phi_k(x)\ mod(n) = \sum_{i=1}^{r} f_i(x) \times \mathbb{D}_{i,k}\ mod(n)$ for $1 \leq k \leq s$. To be able to extract the kth element of the desired rows (β) using servers' responses, the querying client should remove the effect of the undesired records ($\mathbb{D}_{i,k}(x)$, for $i \notin \beta$) in $\phi_k(x)$. To do so, the client should construct the polynomials in a way that for any inputs sample $x \in \mathbb{X}$, the functions $f_i(x)$ output zero for $i \notin \beta$, and output a non-zero value for $i \in \beta$. Recall that each client polynomial $f_i(x)$ can be represented as:

$$\begin{aligned} f_i(x) &= \sum_{m=1}^{q+1} y_m \times \ell_m(x) \quad mod(n) \\ \ell_m(x) &= \prod_{1 \leq v \leq q+1, v \neq m} \frac{(x - x_v)}{(x_m - x_v)} \quad mod(n) \end{aligned} \tag{14}$$

where (x_m, y_m)'s are the points used to generate the polynomials. As can be seen from the above equation, there are two approaches for making each polynomial ($f_i(x)$) zero for $i \notin \beta$. The first approach is making y_m zero in (14) for input $x = x_m$, so $(x_m, 0)$ is one of the points used for interpolation of all the polynomials ($f_i(x)$) for $i \notin \beta$. This requires a fix set of x-coordinates $\mathbb{X} = \{x_1, \ldots, x_{q+1}\}$ be used across all of the r PIR-tailored secret sharing polynomials. This is what has been done by prior PIR protocols [30], [37], [26], [20].

The second approach to make $f_i(x)$ zero for $i \notin \beta$ is to make $\ell_m(x)$ in (14) zero by choosing x and x_v in a way that $x - x_v$ becomes zero. In $mod(p_j)$, if $p_j | (x - x_v)$ or $p_j | y_m$ (where $|$ means division), then $f_i(x)\ mod(p_j)$ will produce zero. In this approach, the x-coordinates used for generating functions, x_ms, can be different from the input x-coordinate, x. This approach enables us to choose different random x-coordinates for each polynomial. In our complete HPIR protocol, we combine these two approaches, i.e., we use different x-coordinates and y-coordinates for the PIR-tailored secret sharing polynomials. Combining these two approaches enables us to use just two prime numbers in constructing the PIR-tailored secret sharing polynomials, which reduces the communication overhead as explained below. This is while the basic version of our protocol (Section VI) needs q prime numbers. Specifically, in our basic protocol, all of the calculations are in $mod(n)$, and n is the multiplication of q prime numbers, so size of sending and receiving elements linearly depends on the number of queries. Using just two prime numbers in our PIR-tailored secret sharing constructions will keep the size of each element in query and response vectors to be fixed and independent of the number of queries. This will improve the efficiency of our complete HPIR in upload and download bandwidth consumption by sending and receiving smaller elements.

Preserving the degree of freedom of secrets using two prime numbers In the basic version of the protocol, the x-coordinates are fixed for all of the secret sharing polynomials; however, for the complete version we use different x-coordinates for constructing each polynomial. We add q new random variables r'_{ij} in x-coordinates of each polynomial, which doubles the degree of freedom of the secrets compared to the basic version (Section VI). If an adversarial server removes the effect of half of the prime numbers, the secrets will still have a high enough degree of freedom. To do so, the client will create half of the PIR-tailored secret sharing points using p_1 (i.e., $(p_1 \times random + secret)$), and the other half using p_2. Algorithm VII.1 summarizes our complete HPIR protocol.

Correctness: In extracting the requested records (step C2 in Algorithm VII.1), when j is an even number ($j\%2 == 0$), the client will use p_1, otherwise (j is an odd number) she will use p_2, where j is the index of the query. Below we demonstrate the correctness of our HPIR protocol (when j is even) by showing that the client can always reconstruct her queried records using our HPIR protocol.

$$\begin{aligned} &\phi_k(\alpha_j) \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = (\sum_{i=1}^{r} f_i(\alpha_j) \times \mathbb{D}_{i,k}) \\ &\times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = (\sum_{i=1}^{r}(\sum_{m=1}^{q+1} y_{i,m} \times \prod_{v=1, v\neq m}^{q+1} \frac{\alpha_j - x_{i,v}}{x_{i,m} - x_{i,v}}) \\ &\times \mathbb{D}_{i,k}) \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = f_{\beta_j}(\alpha_j) \times \mathbb{D}_{\beta_j,k} \\ &\times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = \mathbb{D}_{\beta_j,k} \end{aligned} \tag{15}$$

where $\mathbb{D}_{\beta_j,k}$ is the kth element of the jth query. As can be seen, the effect of undesired records ($i \notin \beta$) will be cancelled in the calculation of $\phi_k(\alpha_j) \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1)$ since $f_i(\alpha_j)$ will produce zero for them in $mod(p_1)$.

A. Communication Costs

To retrieve q records, the client should send $2 \times q \times r \times w$ bits to the rich server, and $2 \times r \times w$ to the poor server. The rich server will send back $2 \times q \times s \times w$ bits, and the poor server will send back $2 \times s \times w$ bits.

Client (querying for block numbers $\beta = \{\beta_1, \ldots, \beta_q\}$) :
P1. Choose two prime numbers $\mathbb{P} = \{p_1, p_2\}$ with more than w bits ($p_i > 2^w$).
P2. Calculate $n = p_1 \times p_2$, and release it to the servers.
P3. Choose q random distinct $\{\alpha_1, \ldots, \alpha_q\}$ from $\mathbb{Z}_n^*$.
P4. Construct r polynomials of degree q. For generating the ith function, the client will take the following steps:
P4(a). Construct $(q+1)$ x-coordinates $\mathbb{X} = \{x_{i,1}, \ldots, x_{i,q+1}\}$ as follows for $1 \leq i \leq r$ ($r'_{i,j}$ and r' are random numbers from $\mathbb{Z}_n^*$ such that $gcd(x_{i,j} - x_{i,k}, n) = 1$ for a specific i and different j and ks:

$$x_{i,j} = \begin{cases} (r'_{i,j} \times p_1) + \alpha_j \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 0 \text{ (even)} \\ (r'_{i,j} \times p_2) + \alpha_j \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 1 \text{ (odd)} \\ r' \ mod(n) & \text{for } j = q+1 \end{cases}$$

P4(b). Construct $(q+1)$ y-coordinates $\mathbb{Y} = \{y_{i,1}, \ldots, y_{i,q+1}\}$ as follows for $1 \leq i \leq r$ ($r_{i,j}$ is a random number from $\mathbb{Z}_n^*$):

$$y_{i,j} = \begin{cases} (r_{i,j} \times p_1) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 0 \text{ (even)} \\ (r_{i,j} \times p_2) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 1 \text{ (odd)} \\ r_{i,j} \ mod(n) & \text{for } j = q+1 \end{cases}$$

P4(c). Use Lagrange polynomial interpolation to find $f_i(x)$ of degree q that satisfies $(x_{i,j}, y_{i,j})$ for $1 \leq i \leq r, 1 \leq j \leq q+1$.
P5. Choose q random distinct x-coordinates $\mathbb{X}' = \{x'_1, \ldots, x'_q\}$.
P6. Send matrix $\mathbb{Q}_c$ to the rich server ($\mathbb{Q}_c[j][i] = f_i(x'_j)$ for $1 \leq i \leq r, 1 \leq j \leq q$).
P7. Send the the query matrix $\mathbb{Q}_r$ ($\mathbb{Q}_r[0][i] = f_i(r') = r_{i,j}$) for $1 \leq i \leq r, j = q+1$) to the poor server.
Each Server:
S1. Multiply the $\mathbb{Q}_c$ and $\mathbb{Q}_r$ matrices to the database matrix, and return the results ($\mathbb{R}_c = \mathbb{Q}_c * \mathbb{D}$ and $\mathbb{R}_r = \mathbb{Q}_r * \mathbb{D}$) to the client.
Client:
C1. Construct polynomials $\phi_k(x)$ (for $1 \leq k \leq s$) that satisfy $(x'_j, R_c[j][k])$ for $1 \leq j \leq q$ and $(r', \mathbb{R}_r[0][k])$ using Lagrange polynomial interpolation.
C2. extract the items of queried records (for $1 \leq j \leq q, 1 \leq k \leq s$):

$$\mathbb{D}_{\beta_j,k} = \begin{cases} \phi_k(\alpha_j) \times f_{\beta_j}(\alpha_j)^{-1} \ mod(p_1) & j\%2 == 0 \text{ (even)} \\ \phi_k(\alpha_i) \times f_{\beta_j}(\alpha_j)^{-1} \ mod(p_2) & j\%2 == 1 \text{ (odd)} \end{cases}$$

Algorithm VII.1: Our HPIR Protocol (Complete Version)

Pseudo-random number generator for coordinates We can further improve the communication overhead of the poor server by having the client use a pseudo-random number generator to generate the query vectors of the poor server. For our protocol, instead of random $r_{i,q+1}$ for $1 \leq i \leq r$, client will generate r numbers $\{g_1, g_2, \ldots, g_r\}$ in $mod(n)$ using a random seed. Now, the client in our basic HPIR will construct the polynomials as follows:

$$y_{i,j} = \begin{cases} (r_{i,j} \times p_j) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q \\ g_i \ mod(n) & \text{for } j = q+1 \end{cases} \quad (16)$$

and in the complete version, client will construct the polynomials as follows:

$$y_{i,j} = \begin{cases} (r_{i,j} \times p_1) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 0 \\ (r_{i,j} \times p_2) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 1 \\ g_i \ mod(n) & \text{for } j = q+1 \end{cases} \quad (17)$$

Therefore, instead of sending r elements to the poor server, she will just send the seed of the pseudo-random number generator, and the poor server can reproduce these r numbers $\{g_1, g_2, \ldots, g_r\}$ and construct the query vector. Note that using a PRNG will downgrade our security guarantee from information-theoretic to computational security; this is because in the presence of PRNG, the rich server is able to perform a (computationally intensive) exhaustive search to find the seed of the PRNG.

B. Security

Information-theoretic security If all the PIR servers do not collude, they can not learn anything about client's queries. There are $(3q+1) \times r + 2q + 1$ variables in the system that are only known to the querying client: $(q+1) \times r$ variables of r_{ij}, $q \times r$ variables of r'_{ij}, $q \times r$ secrets δ_{ij}, q variables of α_j, q variables of $\mathbb{X}'$, and one variable r'. Half of the random variables r_{ij} and r'_{ij} are obfuscated with p_1 and the other half are obfuscated with p_2. Even if the PIR server knows the values of these two prime numbers, it can only remove half of the variables by calculating the shares in $mod(p_1)$ or $mod(p_2)$. The server knows at most $q \times r$ points of the polynomials, and it can remove half of r_{ij} and r'_{ij}, which results in a more than $q \times r$ degree of freedom for the secrets.

Robustness against colluding servers If all the PIR servers collude, they can determine the client's query since they can factorize n and find the prime numbers used in the scheme. However, if we use larger prime numbers, the security of the protocol will reduce to computationally secure against all colluding servers based on the factorization problem, which is an NP problem. To compromise a client, the servers first need to factorize n. This trades off between performance and security: increasing the size of the prime numbers improves the computational security, but the matrix multiplication will take longer. Therefore, **our protocol is information-theoretically secure if up to t servers collude; with large prime num-**

bers, our protocol is computationally secure even if all the servers collude (given large prime numbers). This is because, if all the servers collude, they still need to obtain the prime factors of n to cancel the effect of random numbers. So, if the prime numbers are large enough, e.g., n has 2048 bits or more, the adversarial servers should solve the factorization problem, which is an NP problem.

C. Overhead Comparison to Prior Work

1) Communication Cost: Table III compares the communication costs of our protocol (Complete Version) with different protocols and in different settings. We also compare a homogeneous version of our HPIR protocol with state-of-the-art homogenous protocols (in the homogenous version of our protocol, we send equal number of shares to the servers). We compare the protocols for the same volume of retrieved traffic. Therefore, we amplify Goldberg's [26] communications by q as it is a single-query protocol.

Also, note that our element size is two times of prior works since our calculations are in $mod(n)$, where n is a multiplication of two prime numbers. Each prime number has about w bits, so our elements are about $2 \times w$ bits. However, in prior works the calculations are in $mod(p)$, where p is a w bits prime number.

Homogeneous version of our protocol Compared to Goldberg's PIR [26], homogeneous version of our protocol is slightly higher in upload and download bandwidth consumption. Comparing to Henry et al. PIR [30], the number of exchanged elements are the same, but each element of our protocol is two times as their elements. However, since Henry et al. PIR protocol is based on ramp secret sharing, they need $t + q$ servers for their protocol. It means that by increasing the number of queries, they need more non-colluding servers, while our protocol can be run on as few as two servers.

Heterogeneous version of our protocol We also compare the heterogeneous version of our protocol (complete version) with the heterogeneous versions of Goldberg's PIR and Henry et al. PIR. We create a heterogeneous version for these prior works by making the client send more queries to one of the PIR servers (the rich server). However, to maintain the privacy level (the maximum number of colluding servers without compromising client's privacy), we need to increase the degree of the polynomials used in their schemes. This results in increasing the bandwidth/computation overhead on the rich server of these prior works without reducing the burden on the poor server. Therefore, *the heterogeneous variant of these prior works has no advantage over their homogenous versions*. Also, we see that using a PRNG in our protocol reduces the upload bandwidth to the poor server at the cost of downgrading our information-theoretic security to a computational security.

2) Computation Cost: Table IV compares the computation cost of our protocol (Complete Version) with other protocols. These numbers are based on standard matrix multiplication, which take $O(n^3)$ operations. For large number of queries (q), using the Strassen's algorithm for matrix multiplication will further reduce the order of matrix multiplication to $O(n^{2.8})$ operations.

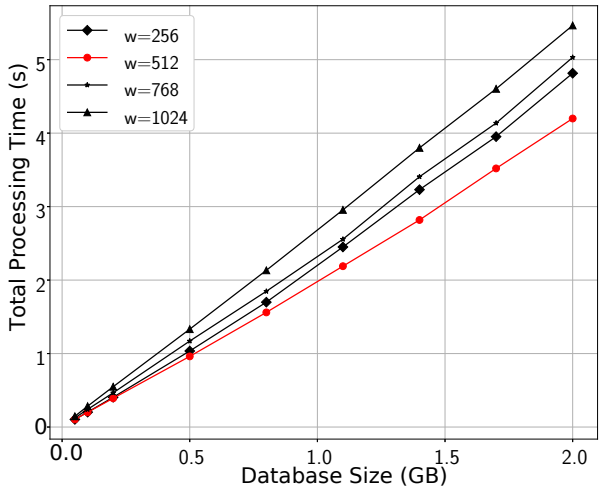


Figure 2: Total computation time (s) (i.e., server and client running times) vs. database sizes (GB) of protocol (complete version) in retrieving one record with different element sizes. $w = 512$ provides the least overhead.

VIII. Implementation

Implementation Setup We have implemented our HPIR protocol in C++, wrapped in Rust. We have implemented our code to be compatible with the Percy++ PIR library [43]. We use the NTL library [49] for handling big number operations similar to the Percy++ [43] PIR suite. Our code is available at https://github.com/SPIN-UMass/HPIR.

We measure the performance of our algorithm on a desktop computer with a quad-core i7 CPU @ 3.6 GHz and 32 GB of RAM, running Ubuntu 18.04. All of our experiments are single-threaded, though our most expensive operation, which is server computation, is highly parallelizable. In all of the experiments, we load the database into the RAM before measuring the time, so the measured times do not include the I/O times. Note that the computation time of our protocol depends mostly on the size of the database, not its dimensions, so in all of the experiments we choose $s = r = \sqrt{N/w}$, which is the communication-optimal block size derived by Goldberg et al. [26] (N is the size of database in bits and w is the element size in bits). All of our experiments are performed in a two servers scenario (that suits most of the real world applications): a rich server with high communication/computation resources and a poor server with lower resources.

We compare our protocol with Goldberg PIR [26] design in a two servers scenario with privacy level $t = 1$. To ensure a fair comparison, we integrate Goldberg's PIR protocol from Percy++ [43] into our test framework, which is wrapped in Rust. We compare our design with this paper since Henry et al. PIR [30] with single query ($q = 1$) is a variant of Goldberg's PIR.

Degree of Heterogeneity (DH) We define a *degree of heterogeneity (DH)* parameter to represent the heterogeneity of our protocol. For a two-server setting, we define DH to

Table III: Communication cost comparison (bits)

PIR Protocol		Upload BW for Each Server	Download BW for Each Server	Total Upload BW	Total Download BW	Minimum Number of PIR Servers (ℓ)
Goldbergs's PIR [26]		$q \times r \times w$	$q \times s \times w$	$\ell \times q \times r \times w$	$\ell \times q \times s \times w$	$t+1$
Henry et al. PIR [30]		$r \times w$	$s \times w$	$\ell \times r \times w$	$\ell \times s \times w$	$t+q$
Homogeneous Version of Our Protocol (Complete Version) ($q+1$ servers, $t=q$)		$2 \times r \times w$	$2 \times s \times w$	$2 \times \ell \times r \times w$	$2 \times \ell \times s \times w$	$q+1$
Homogeneous Version of Our Protocol (Complete Version) (2 servers, $t=1$)		$r \times (q+1) \times w$	$s \times (q+1) \times w$	$2 \times (q+1) \times r \times w$	$2 \times (q+1) \times s \times w$	2
Heterogeneous version of Goldberg's PIR [26]	Rich Server: Poor Server:	$q \times r \times t \times w$ $q \times r \times w$	$q \times s \times t \times w$ $s \times t \times w$	$(\ell+t) \times q \times r \times w$	$(\ell+t) \times q \times s \times w$	$t+1$
Heterogeneous version of Henry et al. PIR [30]	Rich Server: Poor Server:	$r \times t \times w$ $r \times w$	$s \times t \times w$ $s \times w$	$(\ell+t) \times r \times w$	$(\ell+t) \times s \times w$	$t+q$
Our Heterogeneous Protocol (Complete Version), $t=1$	Rich Server: Poor Server:	$2 \times q \times r \times w$ $2 \times r \times w$	$2 \times q \times s \times w$ $2 \times s \times w$	$2 \times (q+1) \times r \times w$	$2 \times (q+1) \times s \times w$	2
Our Heterogeneous Protocol (Complete Version) Using PRNG, $t=1$	Rich Server: Poor Server:	$2 \times q \times r \times w$ $2 \times w$	$2 \times q \times s \times w$ $2 \times s \times w$	$(2 \times q \times r + 1) \times w$	$2 \times (q+1) \times s \times w$	2

Table IV: Computation cost comparison

PIR Protocol		Computation Cost for Each Server	Total Computation Cost	Minimum Number of PIR Servers (ℓ)
Goldbergs's PIR [26]		$\mathcal{O}(q \times r^2 \times s)$	$\mathcal{O}(\ell \times q \times r^2 \times s)$	$t+1$
Henry et al. PIR [30]		$\mathcal{O}(r^2 \times s)$	$\mathcal{O}(\ell \times r^2 \times s)$	$t+q$
Our Heterogeneous Protocol (Complete Version)	Rich Server: Poor Server:	$\mathcal{O}(q \times r^2 \times s)$ $\mathcal{O}(r^2 \times s)$	$\mathcal{O}((q+1) \times r^2 \times s)$	2

be the ratio of the number of query shares the client sends to the rich PIR server divided by the number of query shares she sends to the poor PIR server. This metric represents the bandwidth/computation ratio of the PIR servers. Note that in our protocol, when retrieving q records, the maximum DH is $q/1$, as the client will have $q+1$ secret shares to send to the servers.

Tuning the element size (w) parameter First, we measure the computation overhead performance of our protocol for different element sizes to find the optimal value of w. Figure 2 shows the total computation times (server and client side) for various database sizes, and for different values of w. The plot suggests using $w = 512$ bits as the most efficient value, which we use for the rest of our experiments.

Note that a $w = 512$ bits results in n to have a size of 1024. In case of the collusion of the PIR servers, the security of the protocol will be tied to factorizing n. Therefore, increasing w will improve the security of the protocol in case of collusion at the cost of higher processing times.

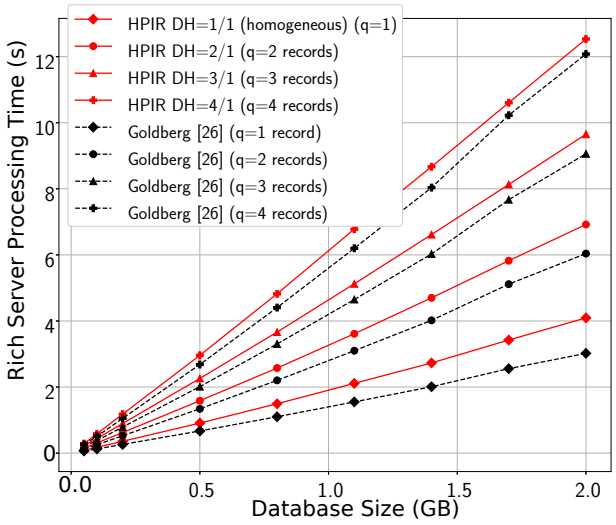
A. Server Computation Overhead

The major computation performed by the servers in our HPIR protocol is matrix multiplication (i.e., multiplying the query matrix, $\mathbb{Q}$, into the database matrix, $\mathbb{D}$). Figure 3 shows the server processing time for both the rich and poor servers for different database sizes and different number of queries (q) for a fixed $w = 512$. As can be seen, the server processing time is linear with the size of the database. The figure also compares the rich and poor server processing times with that of Goldberg's ITPIR [26] for various number of queries. As can be seen, the rich server processing time of our HPIR protocol is *very close to, but slightly larger than* Godlberg's server processing time (for the same number of records q being retrieved). On the other hand, the processing time of the poor server is *significantly smaller* than Goldberg's homogeneous server. As an example, to retrieve a 1.4 MB file (4 records) from a 2 GB database, the rich and poor HPIR servers will take 12.5 and 4.09 seconds, respectively, whereas the two servers of Goldberg will take 12.04 seconds each. That is, **our HPIR protocol significantly reduces the computation overhead on the poor server by slightly increasing the computation on the rich (resourceful) server**. We see that the computation gain of our HPIR further increases by increasing the degree of heterogeneity. As shown in below, increasing the degree of heterogeneity slightly increases the client's computation.

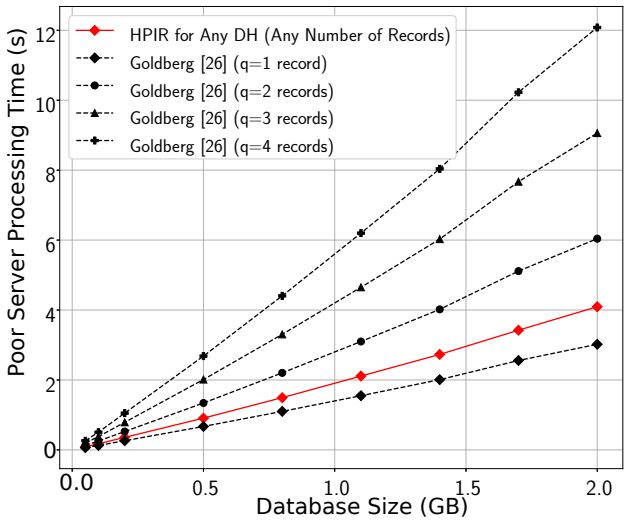
B. Client Computation Overhead

The client computation overhead has two parts: client preparation time, which includes constructing the r polynomials and generating the query matrices, and client data extraction time, which includes the time of constructing s functions $\phi_k(x)$ and extracting the elements of queried records. Figure 4 shows the client processing time of our HPIR protocol for different database sizes and different number of queries (for $w = 512$). As can be seen, client processing time has a sub-linear (square-root) relation with the database size, which is in agreement with Goldberg's results [26].

Also, as Figure 4 shows, our client processing time is *larger* than Goldberg's homogeneous algorithm. However, we see that even the increased client computation times are highly practical for typical clients, e.g., the computation time for $q = 4$ records in a 1.5GB database is around 500ms for HPIR, compared to 200ms for Goldberg's. Also, note that *server computation times are the practical bottleneck in PIR protocols* since they are an order of magnitude larger that client computation times



(a) Server processing time of the rich server vs. Database size



(b) Server processing time of the poor server vs. Database size

Figure 3: Server processing time (for a degree of heterogeneity of $q/1$)

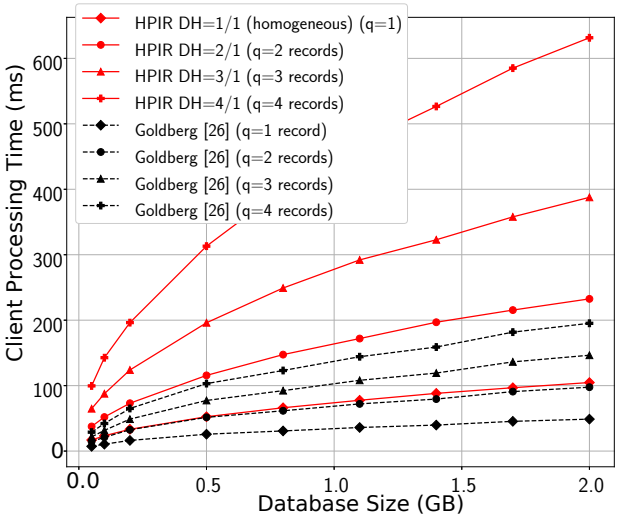


Figure 4: Client processing time vs. Database size

(e.g., 0.2s client computation time compared to 9s server computation time in Goldberg's). Therefore, we believe that HPIR improves a client's overall experience by offloading the bulk of computations to the resourceful server, which can reduce the client's overall retrieving time.

C. Communication Overhead

Recall that, in HPIR the client can control the DH parameter by splitting the $q+1$ query vectors non-uniformly among the PIR servers, e.g., q and 1 vectors to the rich and poor servers, respectively. Figure 5 shows the download and upload bandwidth overheads of our heterogeneous protocol for retrieving $q = 31$ records from a 2 GB database, with different degrees of heterogeneity. As can be seen, increasing the degree of heterogeneity trades off the communication overhead of the poor and rich servers. For instance, for a $DH = 16/16$ (which represents a homogeneous setting), the download/upload bandwidth of the rich and poor servers are 11.3MB each. By increasing DH to $31/1$, the bandwidth of the rich and poor servers will be 21.9MB and 724KB, respectively. Therefore, we see that **HPIR reduces the communication overhead of the poor server by increasing the communication overhead on the rich server.** We also see that the homogeneous version of our HPIR protocol (i.e., for $DH = 16/16$) imposes computation overheads very close to that of Goldberg's homogeneous protocol. Finally, the figure shows that the bandwidth of our poor server when we are using a PRNG is always fixed regardless of the value of DH, since the client sends only one element (the seed of the PRNG).

Note that for the above results, we set $s = r$ [26], which is the optimal value as described in our implementation setup section (so, $s = 5792$ results in records of size 0.35MB). The client can also control DH by changing the value of s (the number of elements in each database record), therefore changing the required number of queries q for a given PIR transaction. We demonstrate this in Figure 6; The figure shows the download and upload bandwidth overheads (normalized by the size of the queried file) of our heterogeneous protocol for retrieving a 10.95MB file from a 2GB database for different record sizes. We can see that there is a *trade-off between the upload bandwidth of the rich server and the download bandwidth of the poor server*; the client can adjust this by changing s.

D. Comparison With State-Of-The-Art PIR Protocols

Here we compare our HPIR design with state-of-the-art (homogenous) two-server PIR designs of PIR-PSI [19] and RAID-PIR [18], as well as the state-of-the-art single-server SealPIR [4], in terms of computation and communication costs. Note that not all PIR protocols can be converted into an HPIR format, so we compare with their regular (homogenous) versions. In particular, RAID-PIR [18] is based on XOR, and

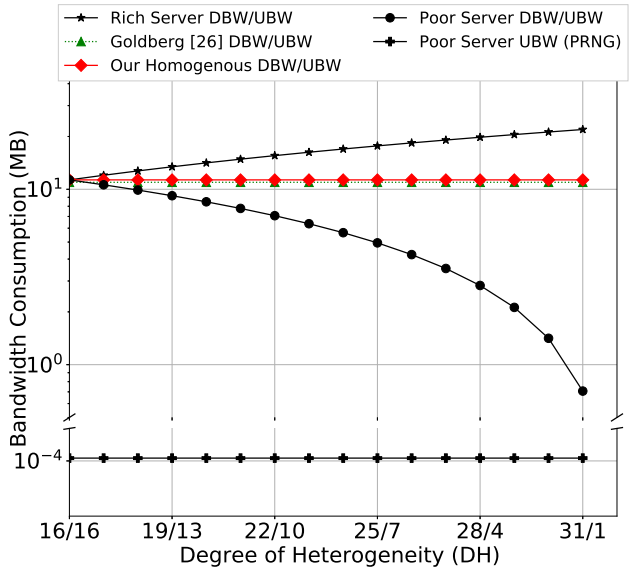


Figure 5: The upload and download overheads for our HPIR (complete version). We download a 10.95MB file from a 2GB database with $q = 31$.

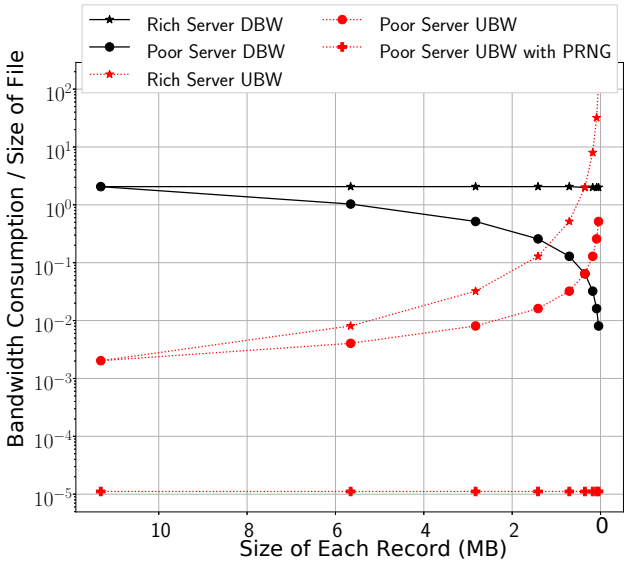


Figure 6: The upload and download overheads (normalized by size of the requested file) for our HPIR. We download a 10.95MB file from a 2GB database.

PIR-PSI [19] is based on the distributed point function (DPF) rather than secret sharing; therefore there is no trivial way to convert them into an HPIR setting.

The two-server protocol of PIR-PSI [19] and the single-server SealPIR [4] provide computational security, so we compare them with our computational HPIR protocol (with PRNG). For fair comparisons, we use the original implementation of PIR-PSI [45] and SealPIR [47], and we use the original values for retrieved file sizes used in their implementations. As shown in Table V, our poor HPIR server (with DH=2/1) has smaller server computation overhead compared to these two PIR designs, despite the fact that HPIR retrieves a much larger file (i.e., 270kB versus 4B and 288B). Also, the upload bandwidth of HPIR is better than these two designs, and its download bandwidth is close to them despite retrieving much larger data.

Table V: Comparison of the computation and communication costs between HPIR and state-of-the-art PIR designs in a (database size is 288MB)

PIR Protocol	Retrieved file	Server computation (s)	UBW (MB)	DBW (MB)
PIR-PSI [19]	4 B	0.528	0.02	0.03
SealPIR [4]	288 B	3.00	0.06	0.25
Computational HPIR (DH=2/1)	270 KB	Rich: 1.14 Poor: 0.39	Rich: 0.52 Poor: 0.001	Rich: 0.52 Poor: 0.26
RAID-PIR [18]	540 KB	0.51	0.003	0.52
IT HPIR (DH=4/1)	540 KB	Rich: 1.16 Poor: 0.40	Rich: 1.05 Poor: 0.26	Rich: 1.05 Poor: 0.26

We also compare RAID-PIR [18], which is an ITPIR algorithm, with our IT HPIR protocol. We use the original implementation of RAID-PIR [46]. As shown in Table V, in retrieving a file of 540 KB, the poor HPIR server achieves lower computation and download bandwidth overheads. Increasing the DH metric will further reduce the load on our poor HPIR server.

IX. Conclusions

We introduced a new class of multi-server PIR protocols, called heterogeneous PIR (HPIR), in which the PIR servers running the protocol undertake different computation and communication overheads. We argue that HPIR algorithms enable new applications for PIR by allowing the participation of low-resource parties in running private services, as well as improve the utility of some of the existing applications of PIR.

We designed the first HPIR protocol that is based on a novel PIR-tailored secret sharing construction, and deployed an efficient implementation of it compatible with the Percy++ PIR library [43]. We extensively evaluated the performance of our implemented HPIR protocol in different settings, e.g., for different degrees of heterogeneity.

Acknowledgements

We would like to thank Christina Poepper for shepherding our paper, Adam O'Neill and Mukul Kulkarni for extremely helpful discussions, and anonymous reviewers for their feedback. This research was funded by the NSF award #1719386 and the Intel Corporation award #34627511.

References

[1] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "XPIR: Private information retrieval for everyone," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 2, pp. 155–174, 2016.

[2] C. Aguilar-Melchor and P. Gaborit, "A lattice-based computationally-efficient private information retrieval protocol," in *Western European Workshop on Research in Cryptology*. Citeseer, 2007.

[3] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, "Design considerations for a network of information," in *Proceedings of the 2008 ACM CoNEXT Conference*, 2008, pp. 1–6.

[4] S. Angel, H. Chen, K. Laine, and S. Setty, "PIR with compressed queries and amortized query processing," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 962–979.

[5] A. Beimel and Y. Ishai, "Information-theoretic private information retrieval: A unified construction," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2001, pp. 912–926.

[6] A. Beimel, Y. Ishai, E. Kushilevitz, and I. Orlov, "Share conversion and private information retrieval," in *2012 IEEE 27th Conference on Computational Complexity*. IEEE, 2012, pp. 258–268.

[7] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, "Breaking the o (n/sup 1/(2k-1)/) barrier for information-theoretic private information retrieval," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*. IEEE, 2002, pp. 261–270.

[8] A. Beimel and Y. Stahl, "Robust information-theoretic private information retrieval," in *International Conference on Security in Communication Networks*. Springer, 2002, pp. 326–341.

[9] R. Bhat and N. Sunitha, "A novel hybrid private information retrieval with non-trivial communication cost," in *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*. IEEE, 2018, pp. 1–7.

[10] G. R. Blakley and C. Meadows, "Security of ramp schemes," in *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1984, pp. 242–268.

[11] N. Borisov, G. Danezis, and I. Goldberg, "DP5: A private presence service," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 4–24, 2015.

[12] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[13] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 402–414.

[14] J. Cappos, "Avoiding theoretical optimality to efficiently and privately retrieve security updates," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 386–394.

[15] H.-Y. Chien, J.-K. Jan, and Y.-M. Tseng, "A practical (t, n) multi-secret sharing scheme," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 83, no. 12, pp. 2762–2765, 2000.

[16] B. Chor and N. Gilboa, "Computationally private information retrieval," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997, pp. 304–313.

[17] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE, 1995, pp. 41–50.

[18] D. Demmler, A. Herzberg, and T. Schneider, "RAID-PIR: Practical multi-server PIR," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*. ACM, 2014, pp. 45–56.

[19] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "PIR-PSI: Scaling private contact discovery," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 159–178, 2018.

[20] C. Devet, I. Goldberg, and N. Heninger, "Optimally robust private information retrieval." in *USENIX Security Symposium*, 2012, pp. 269–283.

[21] P. Dingyi, S. Arto, and D. Cunsheng, "Chinese remainder theorem: applications in computing, coding, cryptography." World Scientific, 1996.

[22] C. Dong and L. Chen, "A fast single server private information retrieval protocol with low communication cost," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 380–399.

[23] F.Olumofin and I. Goldberg, "Privacy-preserving queries over relational databases," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 75–92.

[24] Y. Gertner, S. Goldwasser, and T. Malkin, "A random server model for private information retrieval (or how to achieve information theoretic PIR avoiding data replication)." *IACR Cryptology ePrint Archive*, vol. 1998, p. 13, 1998.

[25] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 121–132.

[26] I. Goldberg, "Improving the robustness of private information retrieval," in *2007 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2007, pp. 131–148.

[27] T. Gupta, N. Crooks, W. Mulhern, S. T. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with popcorn." in *USENIX Symposium on Networked Systems Design and Implementation*, 2016, pp. 91–107.

[28] S. M. Hafiz and R. Henry, "Querying for queries: Indexes of queries for efficient and expressive IT-PIR," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1361–1373.

[29] R. Henry, "Polynomial batch codes for efficient IT-PIR," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 202–218, 2016.

[30] R. Henry, Y. Huang, and I. Goldberg, "One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries." in *Network and Distributed System Security Symposium*, 2013.

[31] R. Henry, F. Olumofin, and I. Goldberg, "Practical PIR for electronic commerce," in *Proceedings of the 2011 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2011.

[32] A. Kiayias, N. Leonardos, H. Lipmaa, K. Pavlyk, and Q. Tang, "Optimal rate private information retrieval from homomorphic encryption," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 222–243, 2015.

[33] O. Knill, "A multivariable chinese remainder theorem," *arXiv preprint arXiv:1206.5114*, 2012.

[34] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, 2007, pp. 181–192.

[35] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka, "A fast (k, l, n)-threshold ramp secret sharing scheme," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 8, pp. 1808–1821, 2009.

[36] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: Single database, computationally-private information retrieval," in *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*. IEEE, 1997, pp. 364–373.

[37] L. Li, M. Militzer, and A. Datta, "rPIR: ramp secret sharing-based communication-efficient private information retrieval," *International Journal of Information Security*, vol. 16, no. 6, pp. 603–625, 2017.

[38] H. Lipmaa and K. Pavlyk, "A simpler rate-optimal CPIR protocol," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 621–638.

[39] P. Mittal, F. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, "PIR-Tor: Scalable anonymous communication using private information retrieval." in *USENIX Security Symposium*, 2011, p. 31.

[40] F. Olumofin and I. Goldberg, "Revisiting the computational practicality of private information retrieval," in *International Conference on Financial Cryptography and Data Security*. Springer, 2011, pp. 158–172.

[41] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[42] L.-J. Pang and Y.-M. Wang, "A new (t, n) multi-secret sharing scheme based on shamir's secret sharing," *Applied Mathematics and Computation*, vol. 167, no. 2, pp. 840–848, 2005.

[43] "Percy++ project on sourceforge," *Available at http://percy.sourceforge.net/*.

[44] A. M. Piotrowska, J. Hayes, N. Gelernter, G. Danezis, and A. Herzberg, "AnNotify: A private notification service," in *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. ACM, 2017, pp. 5–15.

[45] "PIR-PSI implementation github repository," *Available at https://github.com/osu-crypto/libPSI*.

[46] "RAID-PIR implementation github repository," *Available at https://github.com/encryptogroup/RAID-PIR*.

[47] "SealPIR implementation github repository," *Available at https://github.com/microsoft/sealpir*.

[48] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[49] V. Shoup, "Number theory library (NTL) for c++," *Available at Shoup's homepage https://shoup.net/ntl*, 2010.

[50] J. P. Stern, "A new and efficient all-or-nothing disclosure of secrets protocol," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 1998, pp. 357–371.

[51] S. Tarkoma, M. Ain, and K. Visala, "The publish/subscribe internet routing paradigm (psirp): Designing the future internet architecture." in *Future Internet Assembly*, 2009, pp. 102–111.

[52] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee, "Mobilityfirst: a mobility-centric and trustworthy internet architecture," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 74–80, 2014.

[53] H. Yamamoto, "Secret sharing system using (k, l, n) threshold scheme," *Electronics and Communications in Japan (Part I: Communications)*, vol. 69, no. 9, pp. 46–54, 1986.

[54] C.-C. Yang, T.-Y. Chang, and M.-S. Hwang, "A (t, n) multi-secret sharing scheme," *Applied Mathematics and Computation*, vol. 151, no. 2, pp. 483–490, 2004.

[55] S. Yekhanin, "New locally decodable codes and private information retrieval schemes," in *Electronic Colloquium on Computational Complexity*, vol. 127, 2006, p. 2006.

[56] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

APPENDIX A
AN OVERVIEW OF PIR DESIGNS

Information-Theoretic PIR (ITPIR) protocols ITPIR protocols require more than one server, and there is an assumption that these servers are not colluding. These protocols have two main advantages, first they are fast since they do not use complicated cryptography operations. Second, the query is information-theoretic private i.e. the adversary can not learn anything about the queries even though she has unlimited computation power. This kind of PIR requires low computational resources compared to the CPIR protocols.

Chor ITPIR Chor et al. [17] introduced a very basic ITPIR which uses exclusive OR as the main operation. One advantage of XOR is that it can cancel the effect of repeated elements, so the client makes her queries in a way that all the records have an even number of repeats, so at the end, she can cancel their effects. In this protocol the client create ℓ queries that $\ell - 1$ of them are totally random and the last one is the result of XOR of the first $\ell - 1$ vectors and $\vec{e}_j$. Then it will send them to PIR servers, each server multiplies these vectors to the database in GF(2). This dot production in GF(2) is simple XOR, which "1" in position j^{th} of $\vec{e}_j$ means XOR this record, and "0" means do not XOR this record. At the end, each server sends back the result of the XOR to the client, and the client XOR all the responses.

Although this scheme is the first ITPIR protocol, it is still widely cited and proposed for different applications. This amount of citations comes from the fact that this scheme is very fast compared to all other PIR protocols.

Robustness Most of the PIR protocols use "Honest-but-Curious" adversarial model. This model assumed that all the servers are honest, it means they always respond a correct answer, but they try to infer which record has been fetched. One of the main issues with ITPIR is that how the client should deal with servers that do not respond at all or send an incorrect answer that makes the client's result incorrect. A t-private ℓ-server PIR is a private information retrieval protocol which information-theoretically protects the privacy of the client's queries when less than $t+1$ servers collude. Beimel et al. [8] explore the situation in which some servers cannot respond, but the client still can retrieve the data. They define a t-private k-out-of-ℓ PIR as a PIR in which the client only needs k answers out of ℓ servers to recover her record, and if up to t servers collude then the client still has privacy. They also examine what happens if v servers reply incorrect answers, and how many correct answers the client needs to recover the record successfully. They defined t-private v-byzantine-robust k-out-of-ℓ PIR as a PIR that can handle a v number of byzantine servers which sends incorrect answers to corrupt the client's result.

PIR using secret sharing [6] and [5] show that secret sharing and secret conversion can be used to construct a private information retrieval. Li et al. [37] propose four different multi-query ITPIR protocols based on ramp secret sharing schemes [35], [53], and they call them ramp secret sharing-based PIR (rPIR). [29] proposes new techniques that increase efficiency of multi-server ITPIR protocols based on ramp secret sharing. This paper shows ramp secret sharing can help in encoding the data similar to encoding the query. They encode each record of the database into multi shares of a secret, and the client can recover the record by sending multi queries for these shares.

Computational PIR (CPIR) protocols Most of CPIR protocols use a single server which is computationally bounded for retrieving data. It means the security of these protocols is based on a very difficult mathematical problem, and if the adversary finds a solution for the problem, or if she has enough time and computation resources, some data will be leaked.

Stern et al. CPIR Stern et al. [50] propose a CPIR protocol that its algorithm is based on additively homomorphic cryptosystem which has these functions: (i) Gen: function of generating public and private keys pk, sk and system parameters , (ii) Enc: Encryption function with public key ,and (iii) Dec: Decryption function. The important point in this scheme is that the cryptosystem is non-deterministic i.e. the Enc is a randomize function that encrypts the same input to different output each time. Dec function will cancel the effect of the random variable that was used in Enc function. The most famous non-deterministic cryptosystem that is used for CPIR is Paillier [41].

XPIR [1] proposes a new CPIR based on this protocol by looking at each record of the database as a polynomial (the elements of the databases are encoded as coefficients of the polynomial). However, the bandwidth consumption is not very good, so SealPIR [4] introduce a way to compress the queries in this system.

Kushilevitz and Ostrovsky CPIR One of the merits of

CPIR is that the client can run the protocol recursively to reduce the bandwidth consumption. This idea was used by Kushilevitz and Ostrovsky [36] in their scheme for improving the communication cost. First, they split the database into several virtual blocks, and each one of these blocks contains some of the real blocks, then the client sends her query for a specific virtual block, then server will calculate the result of that query on the database and will look at the result as a new database, the next query will be applied to this temporary database, and this process continues until one vector of size s will be sent to the client, and client can recover her requested index out of this result.

Aguilar-Mechor et al. CPIR Aguilar-Mechor et al. [2] propose a lattice-based PIR, and their security is based on the differential hidden lattice problem which is an NP problem. Olumofin and Goldberg [40] show that this design is an order of magnitude faster than trivial download which downloads the entire database.

APPENDIX B
SECURITY PROOF

Theorem 1: In our PIR-tailored secret sharing (when $q > t$), regardless of the number of secrets being shared, the participants can not learn anything about the secrets with up to q shares.

Proof: To prove the security of this scheme, we should show that with less than $q+1$ shares, there is no information about the secrets (when $q > t$). For proving security, we should prove that the adversary given q shares *can not* differentiate that a polynomial with one secret with value of 1 generates these q shares or a polynomial with all secrets with value 0. By this proof, we can show that in the PIR protocol based on this scheme, the PIR server cannot differentiate that the PIR client wants a record ($s_i = 1$) or not ($s_i = 0$).

Suppose that the polynomial $f(x)$ is sharing one secret with value 1 and $q-1$ secrets with value 0 (with degree q), and there is another polynomial $f'(x)$ that is sharing 0 as the values of secrets with the same degree of q. We can show that both of these polynomial functions can generate the q shares $(x'_j, f(x'_j))$ the adversary has, i.e., both of them can generate the same q shares ($f'(x'_j) = f(x_j)$ for $1 \leq j \leq q$). We assume that at the worst case, the adversary knows the x-coordinates used for generating secret sharing polynomial $\mathbb{X} = \{x_1, \ldots, x_{q+1}\}$ and the x-coordinates used for generating shares ($\mathbb{X}' = \{x'_1, \ldots, x'_{q+1}\}$).

For generating the polynomial $f(x)$ which shares one secret 1, we use the following points:

$$(1, (r_1 \times p_1)+1 \ mod(n)), \ldots, (q, (r_q \times p_q) \ mod(n)), (q+1, r_{q+1} \ mod(n))$$

For generating the polynomial $f'(x)$ which shares 0s, we use the following points:

$$(1, (r'_1 \times p_1)+0 \ mod(n)), \ldots, (q, (r'_q \times p_q) \ mod(n)), (q+1, r'_{q+1} \ mod(n))$$

Suppose that the adversary has q shares (x'_j, y_j) for $1 \leq j \leq q$, so he has q equations based on Lagrange interpolation. First we prove that the value of the first secret (s_1) is indistinguishable, and then the same proof can be used for other secrets too. If the adversary assumes that this secret sharing is sharing value of $s_1 = 0$:

$$p_1 r_1 L_1(x'_1) + p_2 r_2 L_2(x'_1) + \cdots + r_{q+1} L_{q+1}(x'_1) = y_1 \ mod(n)$$

$$p_1 r_1 L_1(x'_2) + p_2 r_2 L_2(x'_2) + \cdots + r_{q+1} L_{q+1}(x'_2) = y_2 \ mod(n)$$

$$\vdots$$

$$p_1 r_1 L_1(x'_q) + p_2 r_2 L_2(x'_q) + \cdots + r_{q+1} L_{q+1}(x'_q) = y_q \ mod(n)$$

where $\{p_1, p_2, \ldots, p_q\}$ are q different prime numbers and $n = p_1 p_2 \ldots p_q$.

If the adversary assumes that this secret sharing is sharing value of $s_1 = 1$:

$$(p_1 r_1 + 1) L_1(x'_1) + p_2 r_2 L_2(x'_1) + \cdots + r_{q+1} L_{q+1}(x'_1) = y_1 \ mod(n)$$

$$(p_1 r_1 + 1) L_1(x'_2) + p_2 r_2 L_2(x'_2) + \cdots + r_{q+1} L_{q+1}(x'_2) = y_2 \ mod(n)$$

$$\vdots$$

$$(p_1 r_1 + 1) L_1(x'_q) + p_2 r_2 L_2(x'_q) + \cdots + r_{q+1} L_{q+1}(x'_q) = y_q \ mod(n)$$

where $L_m(x'_j)$ is the Lagrange function for specific values of $\mathbb{X} = \{x_1, x_2, \ldots, x_{q+1}\}$:

$$L_m(x'_j) = \prod_{n=1, n \neq m}^{q+1} (x'_j - x_n)(x_m - x_n)^{-1}) \ mod(n)$$

If we show that there is at least one solution for both of these set of equations, we can show that the adversary cannot differentiate that the secret was zero or one. So for set of unknowns $\{r_1, r_2, \ldots, r_{q+1}\}$ and $\{r'_1, r'_2, \ldots, r'_{q+1}\}$ we should show that:

$$(p_1 r_1 + 1) L_1(x'_1) + p_2 r_2 L_2(x'_1) + \cdots + p_q r_q L_q(x'_1) + r_{q+1} L_{q+1}(x'_1) = p_1 r'_1 L_1(x'_1) + p_2 r'_2 L_2(x'_1) + \cdots + p_q r'_q L_q(x'_1) + r'_{q+1} L_{q+1}(x'_1) \ mod(n)$$

$$(p_1 r_1 + 1) L_1(x'_2) + p_2 r_2 L_2(x'_2) + \cdots + p_q r_q L_q(x'_2) + r_{q+1} L_{q+1}(x'_2) = p_1 r'_1 L_1(x'_2) + p_2 r'_2 L_2(x'_2) + \cdots + p_q r'_q L_q(x'_2) + r'_{q+1} L_{q+1}(x'_2) \ mod(n)$$

$$\vdots$$

$$(p_1r_1+1)L_1(x'_q)+p_2r_2L_2(x'_q)+\cdots+p_qr_qL_q(x'_q)+r_{q+1}L_{q+1}(x'_q)=$$
$$p_1r'_1L_1(x'_q)+p_2r'_2L_2(x'_q)+\cdots+p_qr'_qL_q(x'_q)+r'_{q+1}L_{q+1}(x'_q)\ mod(n)$$

We can write:

$$r'_1=r_1+k_1$$
$$r'_2=r_2+k_2$$
$$\vdots$$
$$r'_q=r_q+k_q$$
$$r'_{q+1}=r_{q+1}+k_{q+1}$$

By putting the above solution in the equations we will have:

$$L_1(x'_1)=p_1k_1L_1(x'_1)+\cdots+p_qk_qL_q(x'_1)+k_{q+1}L_{q+1}(x'_1)\ mod(n)$$

$$L_1(x'_2)=p_1k_1L_1(x'_2)+\cdots+p_qk_qL_q(x'_2)+k_{q+1}L_{q+1}(x'_2)\ mod(n)$$

$$\vdots$$

$$L_1(x'_q)=p_1k_1L_1(x'_q)+\cdots+p_qk_qL_q(x'_q)+k_{q+1}L_{q+1}(x'_q)\ mod(n)$$

We know that if an equation has an answer in $mod(p_i)$, it will have answer in $mod(n)$ too where n is a multiple of p_i. So we can apply $mod(p_i)$ on the ith equation of above system, then we have the following equations:

$$L_1(x'_1)=p_2k_2L_2(x'_1)+\cdots+p_qk_qL_q(x'_1)+k_{q+1}L_{q+1}(x'_1)\ mod(p_1)$$

$$L_1(x'_2)=p_1k_1L_1(x'_2)+\cdots+p_qk_qL_q(x'_2)+k_{q+1}L_{q+1}(x'_2)\ mod(p_2)$$

$$\vdots$$

$$L_1(x'_q)=p_1k_1L_1(hx'_q)+p_2k_2L_2(h_q)+\cdots+k_{q+1}L_{q+1}(x'_q)\ mod(p_q)$$

Therefore, based on Multivariable Chinese Reminder Theorem (Appendix D), since all the p_is are co-prime to each other and in each equation we have the $p_iL_i(h)$s that are co-prime in $mod(p_j)$ where $j\neq i$ (we used $\mathbb{X}$ and $\mathbb{X}'$ in Section IV-B that $gcd(x_i-x'_j,n)=1$ for $x_i\in\mathbb{X}$ and $x'_j\in\mathbb{X}'$, there is at least one solution for $\{k_1,k_2,\ldots,k_{q+1}\}$. Therefore the adversary given q shares *cannot* differentiate what secret was shared.

■

APPENDIX C
CHINESE REMAINDER THEOREM (CRT)

Chinese Remainder Theorem is one of the most useful tools in number theory [21]. This theorem shows the existence of solution for following q equations:

$$x=a_i\ \ mod(p_i)\quad \text{for } 1\leq i\leq q$$

This theorem says that if p_is are co-prime to each other, then there is one and only one value for $x\ mod(n)$ where n is $\prod_{i=1}^{q}p_i$.

APPENDIX D
MULTIVARIABLE CHINESE REMAINDER THEOREM

This theorem [33] says that for a linear systems of equations $A\vec{x}=\vec{b}\ \ mod(\vec{P})$ (each equation is $a_{i,1}x_1+\cdots+a_{i,n}x_n=b_i\ \ mod(p_i)$) has solutions for all $\vec{b}$ if the $p_i\in\vec{P}$ are co-prime to each other and there is at least one element in ith row (all rows) of the matrix A that is co-prime with p_i.