Private Information Retrieval with Preprocessing Based on the Approximate GCD Problem

Thomas Vannet(✉) and Noboru Kunihiro

The University of Tokyo, Tokyo, Japan
tvannet@gmail.com

Abstract. PIR protocols allow clients to privately recover data on a server. While many protocols exist, none of them are practical due to their high computation requirement. We explain how preprocessing is a necessity to solve this issue, then we present two independent but related results. First, we show how Goldberg's robust multi-server PIR protocol is compatible with preprocessing techniques. We detail the theoretical computation/memory tradeoff and present practical implementation results. Then, we introduce a new single-server PIR protocol that is reminiscent of Goldberg's protocol in its structure but relies on the unrelated Approximate GCD assumption. We describe its performance and security, along with implementation results.

Keywords: Privacy · Distributed databases · Information-Theoretic Protocols · Sublinear computation · Sublinear communication · Approximate GCD

1 Introduction

Private Information Retrieval was introduced in 1995 [4] as a way to protect a client's privacy when querying public databases. Since there exists a trivial solution consisting in sending the entire database regardless of the query, all efforts were focused on building algorithms sending less data [3]. Eventually algorithms with average communication cost independent of the database [6] were proposed and the problem seemed solved. However a technical report published in 2007 by Sion [14] showed that in practice the algorithms suggested were all performing slower than the trivial one. Essentially, as network speeds increase just as fast as computing power does, the sending of the whole database to a client will never be much more than a constant times slower than the reading of said database by the server. As a consequence of this observation, subsequent PIR research mainly focused instead on improving the running time of the algorithms. Overall, it is fair to measure a PIR algorithm's efficiency as the sum of its computation time (which depends on the CPU) and communication time (which depends on the type of link between client and server). Asymptotically, the overall complexity will be the worse of the two.

O. Dunkelman and L. Keliher (Eds.): SAC 2015, LNCS 9566, pp. 227–240, 2016.
DOI: 10.1007/978-3-319-31301-6_14

It is well known that PIR algorithms must read at least as many bits as the entire database to provide information theoretical client privacy. As such, and regardless of the communication efficiency of the protocol, all those algorithms will asymptotically perform no better than the sending of the entire database with its overall linear complexity.

Clearly, a different approach was needed to achieve practicality. In 2004, Beimel et al. introduced [1] the notion of PIR with preprocessing in multi-server settings, allowing for faster running times. Other approaches include the batching of queries [9–11], where several queries are performed in a single read of the database, or the use of trusted hardware on the server [16,17,19]. All of those schemes come however with clear limitations and may only be used in specific settings. Namely, Beimel et al.'s scheme only works when several non colluding servers host the database, query batching only improves complexity by a constant factor and trusted hardware still requires some degree of trust on the client's side.

We argue that an alternative exists in single-server PIR with preprocessing based on computational assumptions. This type of scheme would have its limitation be the computational assumption, but so do most widely used cryptographic schemes. To the best of our knowledge, there is so far no scheme that manages to answer queries while reading less bits than the entire database size in a completely generic setting.

We point out strong structural similarities between a sublinear protocol by Beimel et al. [1] and a well performing and high-security multi-server protocol by Goldberg [7]. This allows us to combine the best of each protocol and turn Goldberg's protocol into a potentially sublinear one with a noticeably lower computational cost.

However this protocol, like any other information theoretically secure protocol, does not maintain client privacy when every server cooperates. Alternatively, the protocol cannot be used when there is only one server (or a single entity owning all the servers) hosting the database. While Goldberg also suggested a computationally secure protocol that solves this issue, it is particularly inefficient and our preprocessing technique cannot be used on it.

We thus also introduce a brand new single-server PIR protocol. Its computational security is derived from the Approximate GCD assumption [15], well-studied in the field of Fully Homomorphic Encryption. Its low communication rate is obtained using the common construction in Goldberg's and Beimel et al.'s protocols we pointed out earlier. Finally, its low computation complexity is possible using the precomputation techniques we introduce in the first sections of this paper.

The paper is organized as follows. First, we show how Goldberg's protocol is a perfect fit to allow for some preprocessing. Then we show how to do the actual preprocessing as efficiently as possible. In the second part, we present the Approximate GCD assumption and how it can be combined with Goldberg's approach to result in a new single-server PIR protocol. Then we show how precomputations can be achieved on this new protocol. We suggest and justify parameters to keep the scheme safe and present implementation performance.

1.1 Notations

In Sects. 2 and 3, we work in the usual multi-server PIR setting where several servers are hosting the same copy of a public database of size n. This database is $(b_1, \cdots, b_n)$, $b_i \in \{0, 1\}$. We call $t \geq 2$ the minimum number of servers the client has to query for the protocol to work. In Sect. 4, we work in a single-server setting ($t = 1$).

We use the standard notation $\delta_{x,y} = 1$ if $x = y$, 0 otherwise.

For a variable x and a distribution $\mathcal{D}$, we use the notation $x \xleftarrow{\$} \mathcal{D}$ to indicate that x is picked at random from the distribution.

2 The 2-Dimension Database Construction

2.1 Motivation

In this section, we present the construction which will serve as the central building block for the next sections. It was independently used by Goldberg [7] and to some extent by Beimel et al. [1]. It is also found in Gasarch and Yerukhimovich's proposal [5]. Essentially, it is a very simple structure which can be declined in many different ways. It usually provides a communication complexity of roughly $O(\sqrt{n})$, along with a computational complexity of $O(\sqrt{n})$ on the client side. While this seems very high compared to other schemes achieving $O(n^\epsilon)$ for any $\epsilon > 0$ or even constant rate on average, we argue that this is more than enough. This is due to the fact that a query recovers an entire block of $O(\sqrt{n})$ bits. As it turns out, this value seems to correspond to what real-world systems would need. Indeed, for a medium-sized database of 2^{30} bits which would likely contain text data, a query recovering a couple kilobytes of data makes perfect sense. Similarly, in a large database of 2^{50} bits likely containing media files, recovering a megabyte or more of data seems standard.

Besides, its structure is the perfect candidate for precomputations as we will detail in Sect. 3. Finally, this structure can be used in recursive schemes as in Gasarch and Yerukhimovich's protocol [5].

2.2 Goldberg's Robust Protocol

Here we present Goldberg's version, on which we will be able to add a layer of precomputions in Sect. 3 and replace the security assumption in Sect. 4.

Recall that the database is $(b_1, \cdots, b_n)$, $b_i \in \{0, 1\}$. We assume the database can be split into $\texttt{nb}$ blocks of $\texttt{wpb}$ words of $\texttt{bpw}$ bits each, such that $\texttt{nb}\,\texttt{wpb}\,\texttt{bpw} = n$. Note that $\texttt{nb}$ stands for *number of blocks*, $\texttt{wpb}$ for *words per block* and $\texttt{bpw}$ for *bits per word*. If n cannot be decomposed in such a way, we can easily pad the database with a few extra bits to make it so. We set up the database as a 2-dimensional array of words where every word is characterized by two coordinates. In other words, we rewrite $\{b_i\}_{1 \leq i \leq n}$ as $\{w_{i,j} | 1 \leq i \leq \texttt{nb}, 1 \leq j \leq \texttt{wpb}\}$. We also call block X the set of words $\{w_{X,j} | 1 \leq j \leq \texttt{wpb}\}$. All $t \geq 2$ servers are using the same conventions for this 2-dimensional database.

Suppose the client wants to recover block X while keeping X secret. We set $\mathbb{S}$ a field with at least t elements. The client selects $\mathtt{nb}$ polynomials $P_1, \cdots, P_{\mathtt{nb}}$ of degree $t-1$ with random coefficients in $\mathbb{S}$, except the constant term always set such that $P_X(0) = 1$ and $P_i(0) = 0$ when $i \neq X$. He also selects distinct values $\alpha_1, \cdots, \alpha_t \in \mathbb{S}$. These values can be anything and do not have to be kept secret. For instance $\alpha_k = k$ will work. The protocol has only one round, the client sends a request, the server responds and finally the client deduces the value of every bit in block X from the response.

During the first step, the client sends a request to every server involved in the protocol. Specifically, for $1 \leq k \leq t$, the client sends $(P_1(\alpha_k), \cdots, P_{\mathtt{nb}}(\alpha_k))$ to server k.

We consider $w_{i,j}$ as an element of S. Now we define, for $1 \leq j \leq \mathtt{wpb}$, the degree $t-1$ polynomial Q_j with coefficients in S.

$$Q_j = \sum_{i=1}^{\mathtt{nb}} P_i w_{i,j}$$

We will keep this notation through the paper. Note that it exclusively depends on the values the client sent and the database contents.

During the second step, every server answers. Specifically, server k computes the following values and sends them to the client.

$$\left\{ Q_j(\alpha_k) = \sum_{i=1}^{\mathtt{nb}} P_i(\alpha_k) w_{i,j} \right\}_{1 \leq j \leq \mathtt{wpb}}$$

After receiving the answer of all t servers, the client knows $Q_j(\alpha_1), \cdots, Q_j(\alpha_t)$ for every $1 \leq j \leq \mathtt{wpb}$. Since Q_j is of degree $t-1$, the client can interpolate all of its coefficients and compute

$$Q_j(0) = \sum_{i=1}^{\mathtt{nb}} P_i(0) w_{i,j} = w_{X,j}$$

Thus the client knows the value of every word in block X.

2.3 Security

Informally, as a variant on Shamir's secret sharing scheme [13], this scheme is information theoretically as secure as possible. It is a well known fact that a protocol cannot be information theoretically secure if every single server cooperates, thus the best we can hope for is information theoretical protection when at least one server does not collaborate, which is what is achieved here. More formally, the scheme is information theoretically secure against a coalition of up to $t-1$ servers. Clearly if it is safe against a coalition of $t-1$ servers, it is safe against any coalition of less than $t-1$ servers.

2.4 Communication Complexity

The client sends $\mathtt{nb}$ elements of $\mathbb{S}$ to each server. Similarly, each server returns $\mathtt{wpb}$ elements of $\mathbb{S}$. Total communication is $t(\mathtt{nb} + \mathtt{wpb}) \log |\mathbb{S}|$, which becomes $2t\sqrt{n} \log |\mathbb{S}| = O(t \log t \sqrt{n})$ when $\mathtt{nb} = \mathtt{wpb} = \sqrt{n}$, $\mathtt{bpw} = 1$ and $\mathbb{S}$ has exactly t elements. Note that the computational complexity is still linear in the size of the database here since every b_i has to be read by the servers. The seemingly high communication compared to protocols with polylogarithmic [3] or constant [6] communication rate is actually a non-issue, since asymptotically (and also in practice [14]), the server's computation time will be much higher than the data transmission time.

3 Adding Precomputations to Goldberg's Robust Protocol

In this section, we will show how we can improve the protocol from Sect. 2 by adding an offline step only executed once. First we use the exact same approach as Beimel et al., which is not practical in our setting, and then we show how to modify it to make it pratical and efficient.

3.1 Using Precomputations

We now show we can improve Goldberg's protocol's running time. To reduce the computational complexity on the server side we add a precomputation step before the algorithm is run. A similar idea was introduced in [1], which used essentially the same 2-dimensional database structure describe in Sect. 2. This step is only run once, has polynomial running time and can be done offline prior to any interaction with clients. Its complexity is thus irrelevant to the actual protocol running time.

Here is how we proceed. First, for every $1 \leq j \leq \mathtt{wpb}$ (every word in a block), we partition the $m = \mathtt{nb}\ \mathtt{bpw}$ bits $b_{1,j}, \cdots, b_{m,j}$ into m/r disjoint sets of r bits each (without loss of generality, we assume r divides m). There are m/r such sets for every j, or n/r sets in total. We call these sets $C_{i,j}$, $1 \leq i \leq m/r$ and $1 \leq j \leq \mathtt{wpb}$. For every $C_{i,j}$, the server computes all $|\mathbb{S}|^r$ possible linear combinations with coefficients in $\mathbb{S}$. That is, every $Pre(C, \Delta) = \sum_{d=1}^{r} \delta_d C[d]$ for any $\Delta = (\delta_1 \cdots \delta_r) \in \mathbb{S}^r$. This requires a total of $\frac{n|\mathbb{S}|^r \log |\mathbb{S}|}{r}$ bits.

If we set $|\mathbb{S}| = t$ (which is the minimal number of distinct elements in $\mathbb{S}$, attained when $\mathtt{bpw}$ is 1) and $r = \epsilon \log n$ for some ϵ, this means a total of $\frac{n^{1+\epsilon \log t} \log t}{\epsilon \log n}$ bits have to be precomputed and stored on the servers. This value is potentially very large if t is too large. We will solve this issue in Sect. 3.2. Computing one $Pre(C, \Delta)$ requires r operations and overall the preprocessing of the whole database requires $O(n^{1+\epsilon \log t})$ operations.

During the online phase, each server only reads m/r elements of $\mathbb{S}$ to compute $Q_j(\alpha_k)$. Over all t servers, only $\frac{tn \log |\mathbb{S}|}{r}$ bits are read. Note that the precomputed

database should be ordered in such a way that, for a fixed i, all $Pre(C_{i,j}, \Delta)$ are stored consecutively. This way Δ, which is sent by the client, only has to be read by the server once. The nature of the algorithm (reusing the same Δ for every $1 \leq i \leq \mathtt{nb}$) is what allows us to achieve sublinearity where different protocols would not benefit from the computations.

Once again, if we set $|\mathbb{S}| = t$, $\mathtt{bpw} = 1$ and $r = \epsilon \log n$ for some ϵ, a total of $\frac{nt \log t}{\epsilon \log n}$ bits are read during the online phase. If $t \log t < \epsilon \log n$, that means the server's computation is sublinear in the size of the database.

Communication complexity and security are exactly the same as detailed in Sect. 2. Indeed, a coallition of servers receives exactly the same information from a client as before. This is interesting however if one notes in particular that the sublinear complexity would allow, from an information theoretical point of view, any individual server to distinguish some bits from the original database which are definitely not requested by the client. This could have potentially lead to an attack from a coallition of servers if it was not for the formal proof of Sect. 2. This shows yet again how fitting Goldberg's scheme is for this type of computational improvements through precomputation.

3.2 Decomposition over Base 2

In this section, we present another approach to make precomputations even more efficient. This potentially improves Goldberg's complexity by several orders of magnitude.

Clearly the most limiting factor in the construction from the previous section (and in Beimel et al.'s approach [1]) is the $|\mathbb{S}|^r$ factor. In some scenarios, we want t (and thus $|\mathbb{S}|$) to be large for redundancy purposes. Using words with more than a single bit can also be interesting. Besides, it is possible to create schemes based on a computational assumption which require very large groups $\mathbb{S}$, as we will show in Sect. 4. Alternatively, some implementations (including Goldberg's `percy++` [8]) set a fixed large $\mathbb{S}$ regardless of how many servers the client decides to use. In all of those situations, the space requirements on the server become prohibitive. We assume here that $\mathbb{S} = \mathbb{Z}_\ell$ for a possibly large ℓ.

As before, in the first step the client sends to server k the following values.

$$(P_1(\alpha_k), \cdots, P_m(\alpha_k))$$

We can however decompose those values as follows, since elements of $\mathbb{S}$ have a standard representation as integers.

$$P_i(\alpha_k) = \sum_{c=0}^{\log \ell} P_{i,c} 2^c \text{ where } P_{i,c} \in \{0, 1\}$$

Then the server's computation can be modified using the following equality.

$$\sum_{i=1}^{\mathtt{nb}} P_i(\alpha_k) w_{i,j} = \sum_{c=0}^{\log \ell} 2^c \sum_{i=1}^{\mathtt{nb}} P_{i,c} w_{i,j}$$

Now once again we partition $\{1, \cdots, m\}$ as defined earlier into m/r disjoint sets $C_{i,j}$ of r elements each, where $1 \leq i \leq m/r$ and $1 \leq j \leq \texttt{wpb}$. For every such set C and $\delta_1, \cdots, \delta_r \in \{0, 1\}$, the server precomputes $\sum_{d=1}^{r} \delta_d C[d]$. Each precomputed value requires only $\log r$ bits for storage.

If $r = \epsilon \log n$, the following holds regarding the precomputed database size.

$$\frac{m}{r} \texttt{wpb} 2^r \log r = n^{1+\epsilon} \frac{\log(\epsilon \log n)}{\epsilon \log n} < n^{1+\epsilon}$$

Thus less than $n^{1+\epsilon}$ bits are precomputed and stored in total. This value is independent of the size of $\mathbb{S}$ and thus indirectly independent of t. The one-time preprocessing requires $O(n^{1+\epsilon})$ elementary operations.

Now during the online phase, Q_j can be computed in $\frac{m}{r} \log l$ operations. Overall, the t servers will return their results in $n \frac{t \log \ell}{r} = n \frac{t \log \ell}{\epsilon \log n}$ operations, which is sublinear if $t \log \ell < \epsilon \log n$. Note that for real world values, $\log r = \log(\epsilon \log n)$ is small and summing m/r values should give a number that fits in 64 bits. This means that the online phase of the protocol can be efficiently implemented without using any large integer libraries for its most expensive step. Then only $\log \ell$ operations have to be performed using large integers. Because of this feature, the algorithm remains highly efficient even when $t \log l > \epsilon \log n$ as is the case with very large l, t or small ϵ (to save space).

3.3 A Note on Goldberg's Computationally Secure Scheme

If all t servers collaborate, it is very easy for them to recover the secret X. Furthermore, we know it is impossible to make an information theoretically secure protocol with sublinear communication when every server cooperates. However it is still possible to make a computationally secure protocol in this situation as shown in [7]. Note that because all of the servers can collaborate, we might as well consider that all t servers are the same, and that gives us a single server PIR protocol (although in some scenarios having several servers can also be convenient).

The basic idea behind the protocol is to encrypt the shares sent to the servers with an additively homomorphic scheme and to have the servers perform the computation on the ciphertexts. In practice, the Paillier cryptosystem is used. It has the interesting property that the product of ciphertexts is a ciphertext associated to the sum of the original plaintexts. It is not possible to add a precomputing step in this situation because the parameters (the modulus) must be chosen privately by the client. As such, all hypothetical precomputations would be done in $\mathbb{Z}$ instead of some $\mathbb{Z}/\ell\mathbb{Z}$ and reading a hypothetical precomputed product would require reading as many bits as reading the individual components of the product.

Instead, we can choose to use another additive homomorphic scheme like one based on the Approximate GCD problem. The scheme has however to be changed in several ways which we describe in the next section.

4 Single Server PIR Using the Approximate GCD Assumption

4.1 Computational Assumptions

In this section we reuse the 2-dimensional construction of the database, but instead of securing it through Shamir's secret sharing scheme, we rely on the Approximate GCD assumption. The assumption, which has been the subject of widespread study thanks to its importance in the field of Fully Homomorphic Encryption, states that it is computationally hard to solve the Approximate GCD problem for sufficiently large parameters. This problem can be formulated as in Definition 1.

For any bit length λ_q and odd number p, we call $\mathcal{D}(\lambda_q, p)$ the random distribution of values $pq + \epsilon$ where q has λ_q bits and $\epsilon \ll p$. In addition, for a bit $b \in \{0, 1\}$, we call $\mathcal{D}(\lambda_q, p, b)$ the random distribution of values $pq + 2\epsilon + b$ where q has λ_q bits and $\epsilon \ll p$.

Definition 1 *(Approximate GCD).* Let $\{z_i\}_i \xleftarrow{\$} \mathcal{D}(\lambda_q, p)$ be a polynomially large collection of integers. Given this collection, output p.

It was introduced in 2010 by Van Dijk et al. [15]. In the same paper, they also showed that the following problem can be reduced to the Approximate GCD problem.

Definition 2 *(Somewhat Homomorphic Encryption).* Let $\{z_i\}_i \xleftarrow{\$} \mathcal{D}(\lambda_q, p)$ be a polynomially large collection of integers, $b \xleftarrow{\$} \{0, 1\}$ a secret random bit and $z \xleftarrow{\$} \mathcal{D}(\lambda_q, p, b)$. Given $\{z_i\}_i$ and z, output b.

Now here is how we construct the scheme.

As before we assume the database is $w_{i,j}$, $1 \leq i \leq \mathtt{nb}$, $1 \leq j \leq \mathtt{wpb}$, each $w_{i,j}$ containing $\mathtt{bpw}$ bits. Note that we have $\mathtt{nb}\ \mathtt{wpb}\ \mathtt{bpw} = n$, the total bit size of the database. First we consider the convenient case where $\mathtt{bpw} = 1$ (every word is a single bit).

Suppose the client wants to recover block X consisting of $\{w_{X,j}\}_{1 \leq j \leq \mathtt{wpb}}$. The client picks a large random odd number p which will be its secret key. He selects $\mathtt{nb}$ random large numbers q_i and ϵ_i and computes $P_i = pq_i + 2\epsilon_i + \delta_{i,X}$.

Then the server computes for every j from 1 to $\mathtt{wpb}$, $R_j = \sum_{i=1}^{\mathtt{nb}} b_{i,j} P_i$. He then sends $\{R_j\}_{1 \leq j \leq \mathtt{wpb}}$ to the client.

For every R_j received, the client can compute $(R_j \mod p) \mod 2 = (\sum_{i=1}^{\mathtt{nb}} b_{i,j}(pq_i + 2\epsilon_i + \delta_{i,X}) \mod p) \mod 2 = \sum_{i=1}^{\mathtt{nb}} b_{i,j}(2\epsilon_i + \delta_{i,X}) \mod 2 = b_{X,j}$. As such, he retrieves the entire block X.

It is important to note that this construction is somewhat homomorphic and has been used to create fully homomorphic encryption [15], but here we

only care about the additive property of the scheme. This is very important because without any multiplications, the noise will progress very slowly and we realistically do not have to worry about it becoming too large. Also, our construction is unrelated to other PIR protocols based off generic somewhat or fully homomorphic encryption systems like Yi et al.'s [18] or Boneh et al.'s [2].

4.2 Complexity

First let us look at the communication complexity. The client sends $\mathtt{nb}$ values P_i and the server returns $\mathtt{wpb}$ values R_j. We call λ_p the bit size of the secret p and λ_q the bit size of the q_is. We also write $\lambda = \lambda_p + \lambda_q$. The actual values of these security parameters will be discussed in Sect. 4.4. Each P_i is thus λ bits long. Since R_j is essentially a sum of $\mathtt{nb}$ P_is, its bit size is $\lambda + \log_2 \mathtt{nb}$.

The overall communication of the protocol is $\mathtt{nb}\lambda + \mathtt{wpb}(\lambda + \log_2 \mathtt{nb})$ to retrieve a block of $\mathtt{wpb}$ bits. When $\mathtt{nb} = \mathtt{wpb} = \sqrt{n}$, and for fixed security parameters, the communication is $O(\sqrt{n} \log n)$ or $O(\log n)$ per bit recovered.

Now let us detail the computational complexity. For clarity's sake, we call operation any addition of two $\lambda + \log_2 \mathtt{nb}$ or less bits integers, or a multiplication of a λ_p bit-long integer by a λ_q bit-long one. The client performs $O(\mathtt{nb})$ operations to send the query, the server in turn performs $O(\mathtt{nb}\ \mathtt{wpb})$ operations to execute it. Finally the client performs $O(\mathtt{wpb})$ operations to recover the block values from the server's reply.

The overall computational complexity of the protocol is $O(\mathtt{nb}\ \mathtt{wpb}(\lambda + \log \mathtt{nb}))$. When $\mathtt{nb} = \mathtt{wpb} = \sqrt{n}$, and for fixed security parameters, this becomes $O(n \log n)$.

4.3 Precomputations

As the scheme uses the 2-dimensional structure of the database, we can use precomputations in the exact same way we described in Sect. 3.2. We write $P_i = \sum_{k=0}^{\lambda-1} P_{i,k} 2^k$, where $P_{i,k} \in \{0, 1\}$. The server selects a precomputing parameter r and computes every possible sum of r consecutive words in the database. The client sends the $P_{i,k}$s by groups of r bits for a fixed k. Now the server can compute the R_js r times faster than previously by working with small integers and only performing λ large integer operations per R_j. See Sect. 3.2 for details on this technique.

Communications are unchanged (the client sends the same amount of bits, simply changing their order) at $O(\log n)$ bits transmitted for every bit recovered.

Computations are unchanged on the client side. On the server side, each R_j requires $\lambda\ \mathtt{nb}/r$ small integer operations and λ large integer operations. Overall computational complexity is $O(\lambda n/r + \lambda^2)$, which is asymptotically a r times improvement over the standard version.

4.4 Security

The security of a PIR scheme is defined by the following problem. Given two client queries for blocks X_1 and X_2 respectively, it should be computationally hard to distinguish them. In this scheme, this means that the server must distinguish two queries d_{X_1} and d_{X_2} where $d_X = \{P_i | P_i \xleftarrow{\$} \mathcal{D}(\lambda, p, 0)$ when $i \neq X, P_i \xleftarrow{\$} \mathcal{D}(\lambda, p, 1)$ otherwise$\}$. Let us call D_X the distribution of all possible queries d_X.

Now we can show that if an attacker can indeed distinguish these two queries when given access to as many samples from $\mathcal{D}(\lambda, p, 0)$ as it needs, it can break the Somewhat Homomorphic Encryption (Definition 2) and thus solve the Approximate GCD Problem (Definition 1).

Theorem 1. *Given access to values of $\mathcal{D}(\lambda, p, 0)$, if it is possible to distinguish two queries for distinct bits of a database in polynomial-time, then it is also possible to recover the value p in polynomial-time.*

Proof. Let us assume there exists a distinguishing algorithm $\mathcal{A}(d)$ that returns 1 with probability σ if $d \in D_{X_1}$ and returns 1 with probability $\sigma + \alpha$ if $d \in D_{X_2}$. Now given an encryption $e = pq + 2\epsilon + b$ of a secret bit b, we build an algorithm $\mathcal{B}(e)$ that recovers b with probability at least $1/2 + \alpha/4$.

First, with probability $\alpha/2$, we return 0 and stop immediately. Otherwise, we draw $r \xleftarrow{\$} \mathcal{D}(\lambda, p, 0)$ and $\mathtt{nb} - 2$ random elements from $\mathcal{D}(\lambda, p, 0)$. We generate a query d with the random elements in every position but X_1 and X_2. We randomly pick X_1 or X_2 and place e in this position, r in the other position. We then return $\mathcal{A}(d)$ if the picked position was X_1, $1 - \mathcal{A}(d)$ otherwise.

If b was equal to 0, then d contains only random encryptions of 0 and does not belong to some D_X. In this case, let us say $\mathcal{A}(d)$ returns 1 with probability σ'. Then $\mathcal{B}(e)$ returns 1 with probability $(1-\alpha/2)(1/2 \cdot \sigma' + 1/2 \cdot (1-\sigma')) = 1/2 - \alpha/4$. Now if b was equal to 1, then $\mathcal{B}(e)$ returns 1 with probability $(1-\alpha/2)(1/2 \cdot (\sigma + \alpha) + 1/2(1-\sigma)) = 1/2 + \alpha/2(3/4 - \alpha/4) \geq 1/2 + \alpha/4$ since $0 < \alpha \leq 1$.

Then we use the result from Van Dijk et al. [15] to show the reduction from recovering b to recovering the secret p and breaking the AGCD assumption. □

Note that here we had to assume that the attacker has access to random encryptions of 0. In real life scenarios, this is a fair assumption as it is common for a server to know some metadata about the encrypted information which includes always-0 bits.

4.5 Multiple Bits Recovery

The process can easily be modified to recover words of more than 1 bit at no additional cost. Essentially, instead of picking the P_is as $pq_i + 2\epsilon + \delta_{i,X}$, we can pick $P_i = pq_i + 2^{\mathtt{bpw}}\epsilon + \delta_{i,X}$ where $\mathtt{bpw}$ is the number of bits per word. Since λ_p has to be large for security reasons and the noise only progresses linearly when processing the database, we can afford to start with a fairly large noise.

For instance, if λ_p is 1024 bits and we process databases with less than 2^{80} blocks, we could pick an ϵ with 512 bits and that would allow $\mathtt{bpw}$ to be as high as 432.

The server's response in this case is unchanged, $R_j = \sum_{i=1}^{\mathtt{nb}} P_i w_{i,j}$ where each w_i is a word of $\mathtt{bpw}$ bits. The client then recovers block X consisting of all the $w_{X,j}$ by computing $w_{X,j} = (R_j \mod p) \mod 2^{\mathtt{bpw}}$.

The overall communication of the protocol is still $\mathtt{nb}\ \lambda + \mathtt{wpb}(\lambda + \log_2 \mathtt{nb})$ but we now retrieve a block of $\mathtt{wpb}\ \mathtt{bpw}$ bits. Besides, we have $\mathtt{bpw}\ \mathtt{wpb}\ \mathtt{nb} = n$, which means that when $\mathtt{nb} = \mathtt{wpb} = \sqrt{n/\mathtt{bpw}}$ the communication is $O(\sqrt{\frac{n}{\mathtt{bpw}}}(\lambda + \log \frac{n}{\mathtt{bpw}}))$ or $O(\frac{1}{\mathtt{bpw}}(\lambda + \log \frac{n}{\mathtt{bpw}}))$ per bit recovered (note that λ has to be larger than $\mathtt{bpw}$).

The overall computational complexity of the protocol is also unchanged at $O(\mathtt{nb}\ \mathtt{wpb}(\lambda + \log \mathtt{nb}))$. When $\mathtt{nb} = \mathtt{wpb} = \sqrt{n/\mathtt{bpw}}$ this becomes $O(\frac{n}{\mathtt{bpw}}(\lambda + \log \frac{n}{\mathtt{bpw}}))$. To sum up, both the communication and computational costs are improved by a factor of $\mathtt{bpw}$, which can be several hundreds high. Note that the same improvement was possible on Goldberg's original scheme, and it comes at the cost of recovering blocks of bits $\mathtt{bpw}$ times larger.

The question this naturally brings up is whether or not this affects the security of the scheme. First, the scheme can be reduced to a version of the Somewhat Homomorphic Encryption scheme (Definition 2) on $\mathtt{bpw}$ bits as detailed in Sect. 4.4. Furthermore, we can show that this version of the Somewhat Homomorphic Encryption scheme can be reduced to the AGCD problem for numbers $pq + 2^k\epsilon + b$, $b \in \{0, 1\}$. We do not detail the proof of this step as it follows exactly Van Dijk et al.'s [15].

This shows that this scheme is at least as secure as a version of the AGCD problem with $\mathtt{bpw}-1$ known noise bits. Alternatively, any generic AGCD problem instance can be turned into one of those instances by bruteforcing the value of said bits. As such, for small enough word sizes (say, up to 32 bits), the security of the scheme is mostly unaffected.

5 Implementation

Goldberg provided an implementation called `percy++` [8] that included the robust protocol from [7] along with that of other PIR protocols. Running tests showed that in multi-server settings, Chor et al.'s protocol [4] performed the fastest, followed by Goldberg's [7]. Note that Goldberg's, and by extension our version described in Sect. 3.2, contains much stronger security features. In single-server settings, Aguilar-Melchor et al.'s scheme [12] was deemed the best performing. The security of this scheme is based off the so called *Differential Hidden Lattice Problem* which the authors introduced and studied. In comparison, our single-server scheme defined in Sect. 4 relies on the computational Approximate GCD assumption on which the main candidate for fully homomorphic encryption is based and has thus received a lot of attention.

We made a straightforward implementation of our protocols without aiming for high levels of optimization. For multi-server protocols, we always picked $t = 2$ servers, the minimum to keep the protocol safe. In real world situations, a much higher number would be desirable, multiplying the overall time complexity by a non negligible constant. For our single server scheme, $\lambda = 1024$ was chosen with a noise ϵ always around $p/\mathtt{nb} > 2^{896}$. This is more than enough to stop all known lattice-based and other attacks against the assumption. In fact, much lower values may provide enough security if no other attacks are developped.

We only care about the computational complexity of the server since the communication and client computational complexities are asymptotically negligible. In situations where the preprocessing would generate files too large for our environment, we simulated reading from random files instead, which makes no difference from the server's point of view since the data received is indistinguishable from random data.

There are a lot of parameters that can be modified when running actual tests. For Goldberg's scheme, the preprocessing parameter r described in Sect. 3.1, the number of bits per word and the size of the database. Our implementation shows that, as expected, in identical settings the version with preprocessing performs r times faster than the original one. This is true for databases small enough that the preprocessed version would still fit in RAM and for databases large enough that it does not fit in RAM even without preprocessing. For middle-sized databases, the added preprocessing forces the algorithm to read data directly from the harddrive in a semi-random manner, which can slow it down compared to the original version depending on reading speeds.

Goldberg already showed that his multi-server robust scheme was several orders of magnitude faster than Aguilar-Melchor et al.'s heavily optimized single server scheme. In our implementation of our single-server from Sect. 4.5, we obtain speeds comparable to Golbeger's protocol. Adding precomputations, our scheme can be several times faster.

6 Conclusion

A completely practical PIR scheme has yet to be found. As long as the entire database has to be read for every query, such schemes will always perform too slowly. Based on the observation that preprocessing is a necessity to reach that goal, we first showed how such precomputations can be added to some already existing protocols and designed a new protocol compatible with this technique.

We presented the first PIR protocol allowing the recovery of a block of bits, protection against strong server collusion and Byzantine servers while performing in sublinear time. Its performance is theoretically better than other such algorithms and the design allows efficient implementations by mostly avoiding the need for large integers libraries. Actual implementation corroborates the theoretical findings. Our scheme is both a generalization of Goldberg's protocol and Beimel et al.'s protocol. Compared to Goldberg's, ours can perform many times faster but requires a polynomial expansion of the database. Even a quadratic

expansion is sufficient to provide a much faster protocol. Compared to Beimel et al.'s, the scheme provides much stronger security against several servers cooperating, a major weakpoint of multi-server PIR protocols.

We also presented an efficient single-server scheme with a simple structure. It allows for precomputations and relies on a computational assumption which is fairly new but is receiving and will likely continue to receive a lot of attention from the research community. Furthermore, the way we use it allows for relatively small parameters compared to FHE schemes based on it.

Acknowledgment. This research was supported by CREST, JST.

References

1. Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers' computation in private information retrieval: PIR with preprocessing. J. Cryptol. **17**(2), 125–151 (2004)
2. Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2013/422 (2013). http://eprint.iacr.org/
3. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999)
4. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
5. Gasarch, W., Yerukhimovich, A.: Computational inexpensive cPIR (2006). http://www.cs.umd.edu/arkady/pir/pirComp.pdf
6. Gentry, C., Ramzan, Z.: Single-database private information retrieval with constant communication rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005)
7. Goldberg, I.: Improving the robustness of private information retrieval. In: Proceedings of the IEEE Symposium on Security and Privacy (2007)
8. Goldberg, I., Devet, C., Hendry, P., Henry, R.: Percy++ project on sourceforge (2014). http://percy.sourceforge.net. (version 1.0. Accessed January 2015)
9. Henry, R., Huang, Y., Goldberg, I.: One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In: 20th Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA, 24–27 February 2013 (2013)
10. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications (2004)
11. Lueks, W., Goldberg, I.: Sublinear scaling for multi-client private information retrieval (2015, to appear)
12. Melchor, C.A., Gaborit, P.: A lattice-based computationally-efficient private information retrieval protocol. Cryptology ePrint Archive, Report 2007/446 (2007). http://eprint.iacr.org/
13. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
14. Sion, R.: On the computational practicality of private information retrieval. In: Proceedings of the Network and Distributed Systems Security Symposium, Stony Brook Network Security and Applied Cryptography Lab Tech Report (2007)

15. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
16. Wang, S., Ding, X., Deng, R., Bao, F.: Private information retrieval using trusted hardware. Cryptology ePrint Archive, Report 2006/208 (2006). http://eprint.iacr.org/
17. Yang, Y., Ding, X., Deng, R.H., Bao, F.: An efficient PIR construction using trusted hardware. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 64–79. Springer, Heidelberg (2008)
18. Yi, X., Kaosar, M.G., Paulet, R., Bertino, E.: Single-database private information retrieval from fully homomorphic encryption. IEEE Trans. Knowl. Data Eng. **25**(5), 1125–1134 (2013)
19. Yu, X., Fletcher, C.W., Ren, L., Van Dijk, M., Devadas, S.: Efficient private information retrieval using secure hardware