Semi-Practical Single Database Private Information Retrieval By Keyword In the Data Enrichment Use Case

Dilan Hayes

August 2021

Abstract

This paper outlines a single database private information retrieval scheme based on the Goldwasser–Micali cryptosystem which enables querying a key value database by keyword. We implement the scheme and investigate the practicality with a focus on the real world use case of data enrichment. We find the scheme to be semi-practical with the scheme requiring a large precomputation stage and the communication size growing prohibitively large for key-value databases with a large number of elements. We also highlight potential weaknesses in the scheme and propose several mitigations as well as possible variations which may allow for larger sized databases.

1 Introduction

Single database private information retrieval or 1dPIR, also in some cases referred to as oblivious transfer (OT) enables two parties to interact such that one party can query a database held by the other and obtain a result but without the other party gaining any knowledge about the query (including both the body of the query and the result returned by the query). In the stronger formulation the querier also gains no additional information from the database other than the result of the specific query [1]. For simplicity we will refer to the querier or client as A, the party/server holding the database as B and the database as D.

Data enrichment is a huge industry with data mining and customer analytics becoming a vital part of almost every companies operations, however as privacy concerns grow and more countries introduce data protection legislation the area becomes increasingly difficult to navigate [2]. A data protection violation or a data breach can lead to large fines, negative media exposure and can have devastating effects on a company. Under the EU GDPR legislation transfer of any customer data to a 3rd party such as a data enrichment service by any company or organisation, including for example simply making an API call, requires a prior data processing agreement to be signed and for the entity to take on liability for potential non-compliance or compliance failures of the provider they entered into the agreement with [3] [4].

Much of the previous works have focused on the theoretical scenario where D is an indexed bitstring and a query is a request for the value of the bit at a specific index [5]. There have been other works which describe variants such as [6] which describes a scheme to query blocks of bits rather than a single bit or [7] which describes a scheme which enables querying by a keyword but in the k-dPIR model where $k > 1$. This paper will consider a more realistic scenario for which this type of scheme might be utilised.

We consider the scenario where the database is a key-value map and the primary overheads we wish to minimise are on the side of the client. Specifically: minimal client side computation, minimal client side disk/memory use, minimal data transfer and minimal interaction (the protocol should ideally be a single query-response). We also desire the properties of the stronger formulations of 1dPIR whereby the client can query via a key rather than an index, the server gains no knowledge of the key or value queried and the client obtains knowledge of only the value corresponding to their queried key.

An example application of this may be a data enrichment provider with a database of emails and associated credit risk scores and a bank that wants to enrich it's own customer database with this 3rd party companies score. In such a scenario simply querying the 3rd party provider and therefore disclosing their customers email address would be a breach of data privacy laws. Traditionally the bank may enter into a contractual agreement with the 3rd party company, which may involve a lengthy evaluation of the 3rd party's own security and still expose the bank to additional risk of the 3rd party having a data breach [8]. Another possibility is that the bank purchases the companies entire database and installs it within their own systems. Neither of these solutions may be ideal as the first option exposes the bank to more risk and the 3rd party may

not want to share it's entire database or may want more real time control of the database contents.

This paper will attempt to construct a protocol that might function under the described scenario. We propose a generic scheme which requires a partially homomorphic encryption scheme and a hashing function which has all desired properties except for small query size and small client to server data transfer. We implement and demonstrate this scheme using the Goldwasser–Micali cryptosystem and the KangarooTwelve extendable output hashing algorithm. The various overheads were measured for varying parameters such as database size, and key length. Several potential variations to the underlying scheme are then outlined, including how it could be implemented using a fully homomorphic encryption scheme allowing the key expansion step to be performed on the server thereby achieving all desired properties of the protocol.

2 Outline of scheme

Denote the database D by the function f

$$f\colon K \to V$$

where

$$V \subseteq \mathbb{F}_2^l, \qquad l \in \mathbb{N}$$

and

$$K \quad \text{is some arbitrary set s.t.} \quad \exists \quad H\colon K \to \mathbb{F}_2^n$$

where

$$n = |K|$$

i.e. D is a set of n key-value pairs where the values are bit-strings of length l and there is some hashing function H which takes the keys and maps them to bit-strings of length n. Formally a bit-string of length n is an element of $\mathbb{F}_2^n$, the vector space of dimension n over the field $\mathbb{F}_2$ ($\mathbb{F}_2$ is also sometimes written as $GF(2)$, the Galois field of order 2).

Let D' be the key-value database obtained by applying H to every key in D denoted by the function g

$$g\colon K' \to V \quad \text{where} \quad K' \subseteq \mathbb{F}_2^n$$

Take M to be the $n \times n$ square matrix over $\mathbb{F}_2$ who's rows consist of the elements of

$$K' = H(k) \ \forall \ k \in K$$

It can be shown

$$P(\det M \neq 0) = \prod_{i=1}^{n}(1 - 2^{-i})$$

and

$$\lim_{n\to\infty} P(\det M \neq 0) = \prod_{i=1}^{\infty}(1 - 2^{-i}) \approx 0.288788$$

i.e. the probability that M is invertible. Choosing successive random hashing functions $H\colon K \to \mathbb{F}_2^n$, the number of attempts required to find a suitable hashing function is distributed according to the Geometric distribution with $mean = \frac{1}{p}$. So:

$$\begin{aligned}\text{Expected number of attempts} &= p^{-1} \\ &\approx 0.288788^{-1} \\ &\approx 3.46275\end{aligned}$$

In short, it will take an average of 3.46275 attempts to find a hashing function that will map the keys of the initial key-value database to an invertible binary matrix. Generating a new random H can be achieved by incrementing a 'nonce' value or through some other means. This result is somewhat surprising as while the H must be found by trial and error the probability of a random binary matrix being invertible is high enough that this step is quite feasible.

Once a suitable H and corresponding M are found, for each bit b_i, $0 \leq i \leq l$, of the value, we can obtain coefficients for a linear equation over $\mathbb{F}_2$ as follows:

Let $\mathbf{V}$ be the $n \times l$ binary matrix and $\mathbf{V}_i$ be the i_{th} row of $\mathbf{V}$ s.t. $\mathbf{V}_i = g(M_i)$ where M_i is the i_{th} row of M. Now let $X = \{x_0, x_1, \ldots, x_l\}$ be the set of column vectors of $\mathbf{V}$. For each bit in the l-bit value vector, the coefficients can be computed as

$$\begin{aligned}Mb_i &= x_i \\ M^{-1}Mb_i &= M^{-1}x_i \\ b_i &= M^{-1}x_i\end{aligned}$$

We are effectively computing each bit of a value independently, and, as we are working in $\mathbb{F}_2$, Mb_i is equivalent to selecting the columns of M for which $b_i = 1$ and XOR-ing all of the values in each row. For a single row in M this consists of simply selecting the entries of the row vector and XOR-ing them together to obtain the bit of the value mapped to by the corresponding key.

We have now outlined an algorithm for computing the value for a given key in a key-value map where the computation is equivalent for all keys and requires only the XOR operation. Given an encryption scheme E with the additive homomorphic property $(E(m_1) \cdot E(m_2)) = m_1 \oplus m_2$, the computation of each bit in the value bit-vector can be performed on encrypted bits rather than the plaintext bits of the key or hash of the key.

We now have all the necessary primitives to construct a 1dPIR protocol.

1. B finds some hashing function H which expands the keys of D and produces an invertible binary matrix.
2. B then computes the l coefficient vectors that compute the values for each key.
3. B then sends H (or the information required for B to recreate H such as the nonce value) to A.
4. For a given key k for which A wants to obtain the value, A computes $H(k)$.
5. A then generates the necessary parameters and obtains the public and private keys of E for the chosen cryptosystem.
6. For each bit with index i in $H(k)$ A then computes $e_i = E(H(k)_i)$.
7. A then sends $(e_0, e_1, \ldots, e_i)$ along with the public key to B.
8. B then computes $(E(v_0), E(v_1), \ldots, E(v_l))$ from the encrypted data using the private key sent by A and the homomorphic property of E and sends this to A.
9. A then decrypts each $E(v_*)$ using their private key to obtain the plaintext bits of the value corresponding to k.

A has obtained the value for the queried key and given a sufficiently secure cryptosystem B is unable to decipher the plaintext values of the encrypted bits of the key or computed value. B therefore gains no knowledge of A's query other than that A made a query.

3 Possible attacks

In this scheme we are assuming that A and B are semi-honest. The primary assumption which is not mitigated for is that the values B supplies are legitimate. For example, going back to the risk score example, we assume that the values in the database are in fact risk scores and not say, randomly generated values.

3.1 Key is not in the database

As A does not know the contents of the database, A has no way of knowing if a given key is actually in the database. Furthermore as the computation performed on the value does not depend on the value on which it is being performed, a key which is not present will produce a result just as if for a key that was present. Depending on the form of the values, A may be unable to recognise that the result is a 'phantom' value. It is also clearly not possible for A to simply ask B if a key is present in the database as this would reveal the key to B and defeat the point of the entire system.

If B is not concerned about the secrecy of the set of keys but only the values a simple way to mitigate this may be for B to simply publish the set of keys to A. Building on this, as the set of keys may be quite large B could instead publish a bloom filter or other set membership construction to allow A to check if a key exists in the database before querying.

If B is concerned about the secrecy of the set of keys one possible solution could be for B to append a hash or checksum to each value. As B has no knowledge of the key it should not be possible for B to spoof the result such that it contained a valid checksum. A key not present in the database would also with very high probability produce a result with an invalid checksum and A could therefore deduce that this key was not present in the database.

This second system has better consequences for privacy however would require the result of a query to be larger as it would need to include the bits of the checksum as well as the value. This increase in the length of the response would likely be negligible however, especially in comparison to the size of the request. Another drawback is that if this scheme is being used as a service where A pays B per request, A would likely have to pay equally for a query that returned no result as for a query that returned a valid result as revealing that the result was invalid would in of itself leak information about the query.

3.2 Invalid response

Depending on the exact implementation B could potentially cause some behaviour in A by intentionally returning an invalid value and through observing the behaviour of A deduce some information about the query.

More concretely, for example if B has published the set of keys but intentionally responds to a valid query with an invalid response, A may assume the corruption occurred on their end and attempt to perform the query again. Just the act of making a second attempt may leak information about the query for example via a maliciously constructed database and hashing function or by B simply publishing additional keys not actually present in the database.

Depending on the underlying cryptosystem it may be possible for B to exploit the properties of the cryptosystem by deceiving A into sending multiple encryptions of the same plaintext.

3.3 Spoofed response

As knowing the computation which computes the mapping f is equivalent to knowing the database, B cannot disclose this to A. (Or in the case where the concern is only for the privacy of A, this is the trivial 1dPIR protocol [9]). This means that B could return any valid response to A and A would be unable to tell if the value returned did in fact correspond to the queried key.

One possible solution to this is to also append the key to the value for every entry in the database. A could then simply check if the key contained in the response matches the key that they queried. Care should be taken however to implement such a mitigation in a way that B cannot create a seemingly valid spoofed response from the query itself.

3.4 Maliciously crafted queries

As the response is just a linear equation applied to the query, A could, through sending enough specially crafted queries, eventually determine the coefficients of each x_i and therefore effectively obtain the database. It would also be impossible for B to tell if a query was maliciously crafted or not.

This may be the hardest attack to mitigate against however for a large database this may be infeasible as the number of queries required would grow with the size of the database. It may also be possible to use more advanced cryptosystems to prevent such an attack such as zero knowledge proofs to prove that the query is of a certain non-malicious form without revealing any information about it.

4 Implementation

4.1 Choice of cryptosystem

One cryptosystem that possesses the desired homomorphic property $(E(m_1) \cdot E(m_2)) = m_1 \oplus m_2$ is the Goldwasser–Micali cryptosystem who's security is based on the quadratic residuosity problem [10].

We chose to use this algorithm as it has the desired homomorphic property and the concepts it uses are well established and relatively easy to understand and implement compared to modern homomorphic constructs. The major drawback is the ciphertext it produces is much larger than the plaintext.

4.2 Choice of hash function

As mentioned in section 5 we initially implemented our own extendable output function (XOF) [11] by using SHA3-256 in CTR mode [12] and concatenating and truncating the outputs.

After benchmarking it was clear that this construction was extremely inefficient. It was replaced with the KangarooTwelve XOF, from the same family as SHA3-256 with both using the Keccak sponge construction [13].

4.3 The Goldwasser–Micali cryptosystem

For some $N \in \mathbb{N}$ we say a is a quadratic residue modulo N if $\exists$ some x s.t. $a^2 \equiv x \mod N$.

The quadratic residuosity problem states that given some $a \in \mathbb{N}$ and N where N is the product of two unknown primes, determining if a is a quadratic residue modulo N is hard.

The Goldwasser–Micali cryptosystem acts on single bits by mapping a plaintext $m \in \{0, 1\}$ to some quadratic residue (QR) if $m = 0$ or quadratic non-residue (QNR) if $m = 1$, modulo some $N = pq$. Decryption consists of determining if a ciphertext c is a QR or a QNR. This is achieved using knowledge of p and q.

The public key consists of the value N and some x with $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$ where $\left(\frac{a}{b}\right)$ is the Legendre symbol of a at b.

Encryption consists of first generating some y where $0 < y < N$ *and* $\gcd(y, N) = 1$. The ciphertext c is then computed as $c = y^2 x^m \mod N$.

If $m = 0$ the QNR x vanishes and c is a QR. If $m = 1$ the QR y^2 is multiplied by the QNR x and the resulting value c is a QNR.

Decryption is done by computing $c_p = \left(\frac{c}{p}\right)$ and $c_q = \left(\frac{c}{q}\right)$. If $c_p = c_q = 1$ then c is a QR and the plaintext is 0 otherwise c is a QNR and the plaintext is 1.

The homomorphic property

For two plaintexts $m_1, m_2 \in \{0, 1\}$, $E(m_i) = y_i^2 x^{m_i}$

$$\begin{aligned} E(m_1) \cdot E(m_2) &= y_1^2 x^{m_1} y_2^2 x^{m_2} \\ &= x^{m_1} x^{m_2} (y_1 y_2)^2 \\ &= x^{(m_1 + m_2)} (y_1 y_2)^2 \end{aligned}$$

If $m_1 = m_2 = 0$ then the x term vanishes and we get a QR.

If $m_1 = m_2 = 1$ then we get $x^2 (y_1 y_2)^2$, the product of two QRs and therefore a QR.

If $m_1 \neq m_2$ then we get $x(y_1 y_2)^2$, a QNR.

Mapping QRs to 0 and QNRs to 1 it is clear that this is equivalent to the exclusive or operation and $E(m_1) \cdot E(m_2) = m_1 \oplus m_2$ holds.

4.4 The full algorithm

Definitions

let D = The list of key-value pairs
n = Number of k-v pairs in D
$K12$ = The KangarooTwelve hash function
H_α = K12 initialised with $\alpha \in 0, 1, \ldots$

B (The Precomputation stage)

Initialise α to 0.

Generate M by applying H_α to all keys.

If $rank(M) \neq n$ increment α and try again.

For each column vector b_i in binary matrix of the database values compute $x_i = M^{-1}b_i$.

Send α to A.

A

Generate large primes p and q

Compute $N = p \cdot q$

Generate x with $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$

Send (N, x) to B

Query

A

let k be the key for which A wants to obtain the corresponding value.

Compute $H_\alpha(k)$.

Compute $(e_1, e_2, \ldots, e_n)$
where $e_i = E(H_\alpha(k)_i)$
and $H_\alpha(k)_i$ is the i_{th} bit of $H_\alpha(k)$.

Send all e_i values to B.

B

For each x_i compute $E(v_i)$ the encryption of the corresponding value-bit by multiplying the corresponding e_i values modulo N

Send all $E(v_i)$ values to B.

A

For each $E(v_i)$ compute $\left(\frac{E(v_i)}{p}\right)$ and $\left(\frac{E(v_i)}{q}\right)$ and determine if $E(v_i)$ is a QR or QNR and therefore recover the value of the bit v_i.

Assemble the recovered bits into the original data type (e.g. an unsigned 32 bit integer) and obtain the plaintext result.

5 Experiments and Analysis

We know from the description of the algorithm that the memory and computational overheads of the scheme depend primarily on the number of key-value pairs rather than the size of the keys or values themselves.

In these experiments we will measure the computational time and memory requirements for both parties at various steps in the algorithm as well as the data transfer size and how these vary with the size of the database. From analysis of the results we will attempt to determine if this scheme is practical in the context of real world data enrichment described in section 1 and at what database size does it begin to become impractical.

5.1 Initial Naive Implementation

An initial naive implementation of the algorithm described in subsection 4.4 was implemented in the Rust programming language using the OpenSSL cryptographic library [14] and the M4RI matrix library [15]. A profiling tool was also developed to automate executing command line applications with varying parameters and record the memory and CPU usage. The following results were recorded:

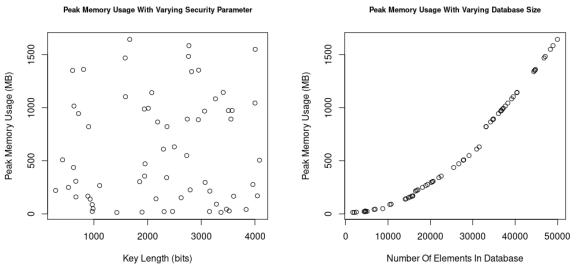

Figure 1: Effects of database size and key length

No significant monotonic correlation was found between the length of the encryption key and the peak memory usage of the program performing the algorithm using the Spearman's rank correlation coefficient with $n = 65, \quad P = 0.87$

The Spearman's rank correlation coefficient between the number of elements in the database and peak memory usage was found to be significant with $\rho = 0.99, \quad n = 65, \quad P < .001$

Following this preliminary experiment several parts of the implementation were identified to be particularly inefficient and were rewritten so as to produce results which better reflect the viability of the algorithm rather than a poor implementation.

5.2 Methodology

The code was refactored into separate modules so that each stage of the protocol could be executed as an independent command line application.

For the databases used in the experiments we generated key-value databases of varying number of elements and the various stages of the protocol were performed on these synthetic databases. The test databases were populated with synthetic data where the key was a randomly generated email address and the value was a random unsigned 32 bit integer.

While as mentioned, the format of the data has almost no influence on the results the use of synthetic data was beneficial to demonstrating the protocol working in the context of a real world application.

Note: that the term key is used here in two different contexts and has two separate meanings. In the context of a key-value database, key refers to some object (in our implementation an email address) which maps to some value (in our case a number). In the context of cryptography key refers to the values (usually simply some very large number or numbers) which are used to perform the encryption or decryption. As we are working with encryption of key-value databases and encrypting the database keys with the cryptographic keys there is some potential for confusion.

5.3 The Precomputation Stage

The precomputation stage performed by the server (B) has to be performed only once over a given key-value set. This does however mean that this step needs to be repeated each time the database is modified and the computational and memory costs of the precomputation stage grow with the number of elements in the database.

The stage consists of two parts: Finding a suitable initialisation value (Also referred to as the nonce value) which generates a suitable hash function; and solving the binary matrix generated by the hash function to obtain the coefficient vectors. Both steps primarily rely on the 'Method of Four Russians' algorithm [16] and implemented using the Rust M4RI wrapper for the C M4RI library [15].

The matrix rank computation was performed for randomly generated databases of varying number of key-value pairs and the memory usage and time taken to perform the operation was recorded. We use the matrix rank computation as a measure for both a single iteration of the nonce search and the calculation of one set of coefficients as although there are almost certainly better algorithms for each, in our naive implementation they are effectively the same computation.

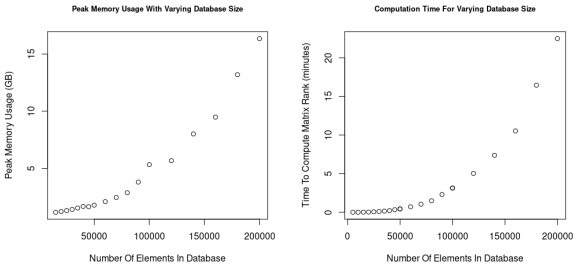

Figure 2: M4RI rank memory use and computation time

Figure 2 shows the memory usage and computation time both grow with the size of the database. While both look like they exhibit exponential growth rates, upon closer inspection we can see that for both the growth is subexponential. This can be seen in figure 3.

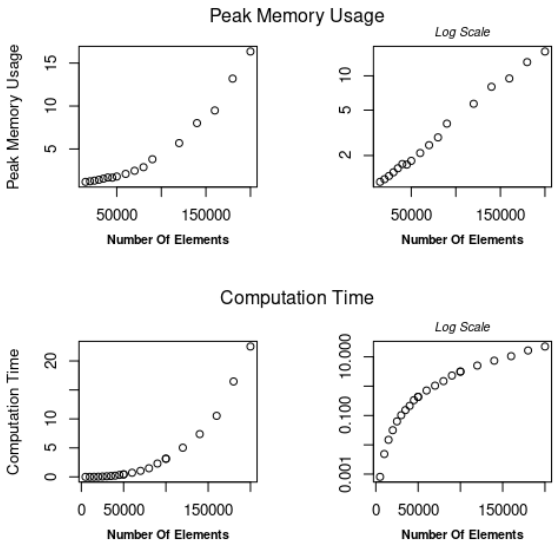

Figure 3: Computation and memory Use, linear and log scale side by side

This is in line with the theoretic complexity of $\mathcal{O}(\frac{n^3}{\log n})$ [17] and the fact that a database of n elements expands into an $n \times n$ binary matrix. The memory requirements were modeled using a quadratic equation and fit to the observed data. The graph of this curve is shown in figure 4 for larger values of n.

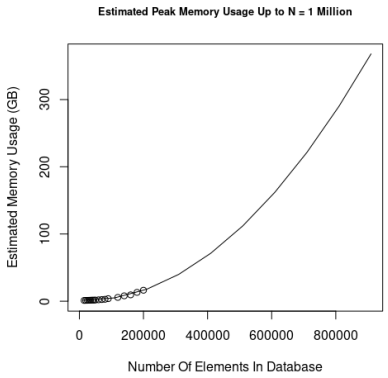

Figure 4: Extrapolated memory requirements

The experiments were initially run locally however it became apparent that more resources would be required to obtain adequate data. Subsequently most experiments were run on an AWS EC2 r4.4xlarge cloud instance. These instances have 122 GiB of RAM and CPU cores with 2.3 GHz clock speed [18]. We go into detail about parallelising the algorithm in subsection 6.2 however the code used to run the experiments and gather the data was almost exclusively single threaded.

After experiments beginning to fail at around $n > 300$ the choice of instance was re-evaluated. We found that the current instance type with the highest CPU clock speed was actually the M5zn family listed as 'General Purpose' instance rather than a 'Compute Optimized'.

Further experiments were conducted to explore if there would be a significant performance gain by switching to a machine with high compute performance as well as high memory. An m5zn.6xlarge instance with 96GiB of RAM but CPU clock speed of 4.5 GHz was created and the matrix rank computation was timed for several different large values of n and compared to the results from the r4.4xlarge instance. The results are shown in Table 1 below.

n	inst	CPU	time	ΔT
200K	m5zn	4.5GHz	16.7 min	6.9min
200K	r4	2.3GHz	23.6 min	
250K	m5zn	4.5GHz	26.1 min	10.6min
250K	r4	2.3GHz	36.7 min	
300K	m5zn	4.5GHz	46.2 min	19.3min
300K	r4	2.3GHz	65.5 min	
350K	m5zn	4.5GHz	75.3 min	24.6min
350K	r4	2.3GHz	99.9 min	

Table 1: Computation time for different systems

While unfortunately it was not feasible to produce a proper sample size and make a more rigorous conclusion, it seems quite apparent from the recorded observations that the increased processing power resulted in a noticeable reduction in computation time.

It was also observed that the time taken to compute the coefficients tends to take very slightly longer than the computation of the rank however both are within the same order of magnitude. This could be a result of more read/write operations required to save the results compared to computing the rank which returns a single value.

A rank computation for $n = 400k$ was eventually performed and took 9.38 hours.

5.4 The Encrypted Query

The encryption operation consists of many modular multiplication operations with modulus in the range of several hundred to several thousand bits in length. The number of operations also grows with the size of the database n.

While the encryption operation itself is not completely trivial, even for extremely large values of n it should be quite feasible to perform this step. As $E(m_i) = y_i^2 x^{m^i}$ each encryption operation involves only one modular squaring and one modular multiplication operation.

Further, along with other possible optimisations, assuming that the values m_i are obtained from a sufficiently secure cryptographic hash function, we would expect that on average around half of them are 0 and therefore encrypting the query consists of n modular squaring operations and $\sim \frac{n}{2}$ modular multiplications.

Of greater concern is the expansion in the size of the encrypted query compared to the plaintext database key it is encrypting. Under the described construction each database key is expanded to a bit-string of length n and each bit is encrypted separately.

The cryptosystem we are utilising derives it's security from the decisional composite residuosity assumption. It can be shown that the quadratic residuosity problem reduces to the integer factorisation problem [19]. Therefore when picking selecting the length of the encryption key we should follow the general best practices for generating numbers secure against prime factorisation.

Under the current NIST guidelines for cryptosystems based on prime factorisation the current recommendation is to use a modulus of at least 2048 bits which provides the equivalent of 112 bits of symmetric security and to transition to at least 3072 bits by 2030 to achieve an effective 128 bits of security [20]. Using the minimum recommended length this means that the size of the encrypted

query is $n \cdot 2048$ bits in size.

A random email address was generated and the corresponding encrypted query was computed for values of n from 1000 to 1000000 at intervals of 1000 giving a sample size of 1000. Figure 5 shows the time taken to encrypt a query with varying database size. Some observations were removed at random before plotting in order to make the plots more readable.

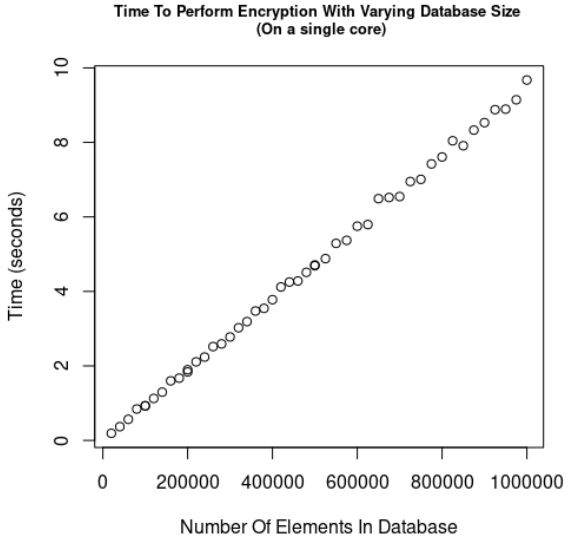

Figure 5: Time to encrypt query

We can see that the query encryption operation is very fast and the time is linear in n. These times are also for a single core sequential implementation. As the encryption is 'perfectly parallel' these times can effectively be divided by the number of CPU cores available to the client.

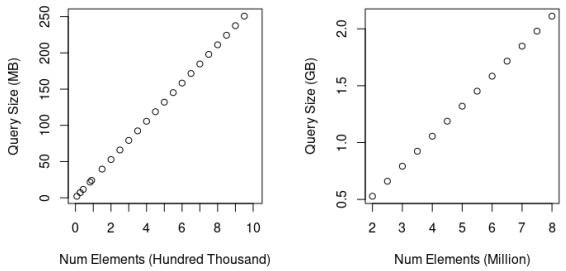

Figure 6: Size Of The Encrypted Query

Figure 6 shows how large the resulting encrypted query needs to be for different size databases with encryption key length fixed at 2048 bits. We already knew this relationship is linear however we include it to confirm the experimental results match the expected sizes and as a visual reference. The side by side plots display the same data but the left is for database sizes up to 1 million elements with the y-axis in Megabytes and the left is for database size above 1 million and y-axis in Gigabytes.

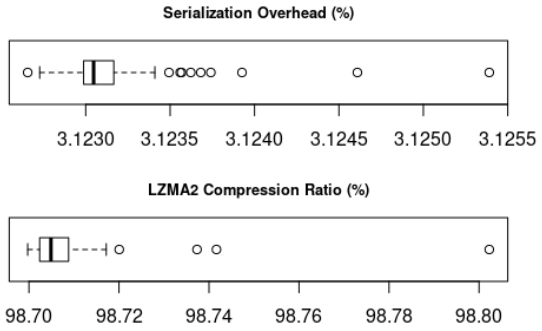

Figure 7: Effects of serialization and Compression

The serialization plot in Figure 7 shows the 'serialization overhead', or how large is the actual encrypted query file in proportion to the theoretical size of $n \cdot 2048$ bits.

In our implementation the query is serialized using the Bincode binary encoding scheme and from the results it looks like it is extremely efficient as the size of the serialized query i.e. the number raw bytes that would be transmitted from the client to the server is inflated by around 3% compared to the theoretic optimal with a very low variance. We can conclude here that we are not introducing any significant overhead during the actual data transfer stage.

We discuss the idea of compressing the query in section 6.3 as a easy way to obtain some reduction in size. Each query was compressed using the LZMA2 compression algorithm. The second plot in Figure 7 shows the resulting compression ratios for each serialized query. We can see that the reduction is negligible and almost certainly not worth the computational and time cost of performing the compression in order to achieve at most a 2% reduction in size.

This was also to be expected and if the result had been otherwise it would likely imply sub-optimal serialization or an insecure encryption scheme.

5.5 Computing the response

This computation is an evaluation of a linear equation over $\mathbb{F}_2$ which under the GM encryption scheme just becomes multiplication modulo N. This is the same computation we measured in the previous step and was already observed to be extremely fast at over a million modular multiplication operations in just under 10 seconds using a single core on low end hardware.

While the computations in this step are not perfectly parallel as in the encryption step it is

still very well suited to basic divide and conquer techniques. or specific modular multiplication algorithms such as Montgomery multiplication [21].

5.6 Decryption

Decryption time was found to be trivial as the decryption operation only needs to be performed once for each bit in the plaintext value. In our implementation this is a 32 bit integer and therefore the client must perform 32 decryption operations. By the nature of public key cryptography this is an easy computation as the client possesses knowledge of p and q.

5.7 Experimental Results

The various experiments demonstrate that the scheme works and is semi-practical.

The context we are primarily considering is data-as-a-service model where we want to reduce the burden on the clients to provide a usable service while maintaining zero knowledge. Within this model we assume that the resources available to the provider relatively unlimited within the realm of practicality however the first experiment showed that the size of the database can quite quickly grow to the point where the resources required might be considered infeasible.

Having said this, with cloud providers now offering machines with RAM in the double-digit Terabyte range the what is considered infeasible may vary or be more a matter of cost rather than impossibility.

As well as this during the course of the research and implementation of the protocol the implementation of the matrix algorithms were somewhat neglected and it is likely that the M4RI algorithm could be replaced with something more efficient. In fact in the paper released by the authors of the M4RI C library they recommend using the Strassen-Winograd algorithm rather than M4RI for very large matrices [16].

The Rust wrapper library also provides extremely user friendly interfaces but this level of abstraction can also result in a tendency to treat the matrix functions as a black box. This also led to cases where the process was terminated as it was unclear if the algorithm had stalled or if it would have completed given enough time.

While computation time for the pre-compute step can also grow incredibly fast with the size of the database this is another case where it is up to the provider to decide the resources they are willing to spend. It is not uncommon for the equivalent of hundreds of years of compute-time to be used when training machine learning models [22] or searching for cryptographic parameters [23].

Due to the encryption key length directly multiplying the size of the encrypted query it was determined that the best option was to use the smallest length considered safe.

The main limiting factor to the scheme appears to ultimately be the query size as a practical service cannot realistically expect clients to send Gigabyte sized requests in order to obtain the results for a single value. The construction of the protocol means that setting a maximum query size determines the maximum database size. Some potential ways to mitigate this are discussed in section 6.

6 Extension To Described Protocol

In addition to the modifications already outlined in section section 3 which address privacy and integrity there are several further steps that could be implemented in relation to performance.

6.1 Partitioning

Using the graph in figure 6, one possible variation is to select an acceptable size for the encrypted query and then divide the database into partitions of length corresponding to the value of n shown in the graph. Or given by $n = \frac{s}{l}$ where s is our desired query size and l is the security parameter or number of bits of N. For a value of $s = 52\,\text{MB} = 416\,000\,000\,\text{bits}$ and taking l to be 2048 we get a partition size of 203125.

This has various trade offs as it would mean that a response would have to be computed and sent for each partition increasing the size of the response. The integrity checks described in section 3 or some other means of validation would need to be used in order for the client to know which of the multiple responses is in fact the legitimate value.

The trade off between the decrease in query size and increase in response size is substantial assuming the value is relatively short. Table 2 fixes the database at 1 million rows and shows for varying partition numbers the decreasing size of the required query and the comparatively insignificant increase in response size. Key-size is assumed to be 2048.

Partitions	1	2	3	4	5
Query	256 MB	128 MB	85.3 MB	64 MB	51.2 MB
Response	8.19 KB	16.38 KB	24.58 KB	32.77 KB	40.96 KB

Table 2: Query-Response trade off for a 1 million row database

Another trade off as a result of partitioning would be that in order to maintain a single query encryption computable on all partitions, the expected number of iterations to find a valid hash

function would increase as the probability of success for any hash now becomes:

$$\lim_{n\to\infty} P(H_{\text{good}}) = \prod_{i=1}^{\infty}(1-2^{-i})^d \approx 0.288788^d$$

Where d is the number of partitions.

Although this might not be a bad trade off as the increases resources for the client is quite minor in comparison to the reductions elsewhere, and performing a larger number of matrix operations over smaller matrices may be preferable than performing operations on a single matrix for huge databases, it is unfortunately quite limited as the probability of finding a successful hash function rapidly tends towards zero as the number of partitions increases.

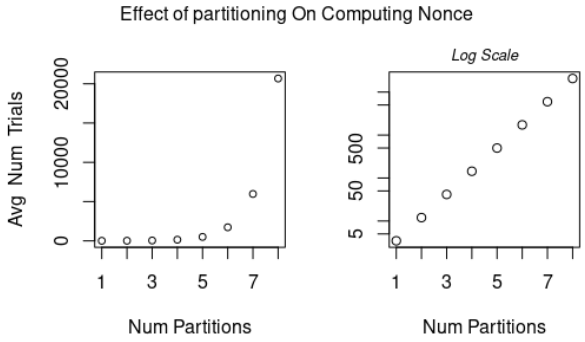

Figure 8: Number of Nonce values that would need to be tested

The above graph shows this effect, at just 8 partitions the expected number of values tested would be $\approx$ 20671, while for no partitions it is $\approx$ 3.46. The number where it remains just about within the realms of feasibility seems to be 5 or 6 partitions with expected number of trials of 497.9 and 1734 respectively.

Another option is to simply 'shard' the database, i.e. split it up into multiple completely separate databases and not search for a single nonce value that works across all shards. This would reduce the resources on the server side but would mean the client would have to send multiple queries effectively multiplying the query size by the number of shards.

6.2 Parallelization

Almost every step in the protocol is highly parallelizable with most being perfectly parallel. When performing the pre-computation each nonce value does not have to be checked sequentially.

Considering the case of a single unpartitioned database with $P(H_{\text{good}}) \approx 0.288788$ for any hashing function produced by a random nonce value, the chance of finding a valid nonce is within a single iteration is greater than 99% when 14 values are checked in parallel and greater than 99.9% at 21. In general the probability is given by the following formula:

$$\left(1-\left(1-\left(\prod_{i=1}^{\lceil \frac{n}{d} \rceil}\left(1-2^{-i}\right)^d\right)\right)^k\right)$$

Where

n = Number of elements in database

d = Number of partitions

k = Number of nonce values checked in parallel

Once the nonce value is found, each coefficient vector can also be computed independently. In our implementation the value is 32 bits long so this stage could be performed as 32 computations in parallel. As well as this there are techniques for performing the matrix operations themselves in parallel which could be applied [24].

On the client side each bit of the hashed key can be encrypted in parallel to produce the encrypted query.

Similarly on the server side each individual bit can be computed independently, furthermore, as the computation on the encrypted bits is just multiplication modulo N and is commutative, this step is well suited for optimization using basic divide and conquer techniques as mentioned in section 5.5.

Finally, taking a step back from the protocol itself, as with any read only service architecture multiple instances could be deployed to process multiple queries in parallel to accommodate demand.

6.3 Compression

One method that was tested to reduce the query size was to simply apply a standard compression algorithm. Unsurprisingly this did not produce any useful reduction.

7 Feasibility For Large Datasets

In order to answer the question of whether this protocol is practical as well as feasible we must also consider the use case and what size such a database might be, as well as both the value of the data to the client and the value of maintaining zero knowledge. One technique that is already being utilised by data enrichment providers is the collection and extraction of email addresses and estimated date of creation from data leaks and breaches [25]. This fits with our implementation as email addresses for the keys and unsigned integers for the value, as a date or score can both easily be represented as a number.

This data point could be incredibly valuable to an institution for example in making a decision weather to grant credit to a customer. In a report

published by the US Federal Reserve it was stated that in general it is the financial institutions who bear the majority of the cost in cases of identity theft or fraud committed using synthetic identities and that each instance resulted in an average incurred loss of $15,000 to the institution [26]. This figure gives some justification as to why a financial institution might value such data points so highly.

One dataset of this form has previously stated a figure of $772,904,991$ unique email addresses [27]. While it is now possible to rent out systems that only a few years ago would have cost several million Euro in up front investment in order to perform the resource intensive operations, and the idea of a 50 or 100 Megabyte query does not sound as outrageous as it might have when PIR was first conceptualised, using the equations outlined above we can calculate that for query sizes within practical ranges and using the minimum secure key length of 2048 bits and taking 5 to be the maximum number of feasible partitions, the maximum database size is in the single digit millions. This is several orders of magnitude less than the benchmark figure of 7.7 billion. This is shown in the graph below.

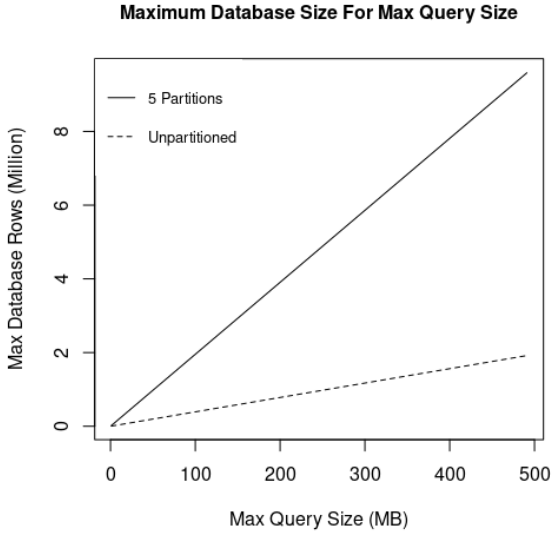

Figure 9: Maximum database size for a partitioned database

8 Future work

8.1 Implementation under a FHE system

The general scheme outlined in section 1 is agnostic to the cryptosystem used so long as it has the particular additive homomorphic property. It simply involves hashing and expanding the database keys so that the number of bits in the output is equal to the number of elements in the database and the bits are sufficiently random. The entire set of expanded keys are then treated as the rows of a square matrix over the field $\mathbb{F}_2$ and the rest of the algorithm is just matrix multiplication.

In order for this to work however the matrix must be invertible, and therefore have full rank. Such an expansion can only be achieved through a function which has sufficient non linearity. Hashing functions or block ciphers are perfect for this as they are designed to be fast and produce transformations which are non linear and appear random. This however is the reason why the query size under the GM cryptosystem implementations is so large. As only addition operations can be performed on the ciphertext it is impossible to apply any function to the ciphertext that will not result in a matrix whose columns are linearly dependent.

Under a fully homomorphic encryption scheme it would be possible to perform non linear transformations on the ciphertext and in theory offload the expansion step to the server. Significantly reducing the amount of data that must be transferred. This is an avenue we would like to explore and determine if the computations required are feasible with the existing FHE systems which have only been developed in the past few years.

8.2 An API proxy layer

One of the contexts in which this type of system might be employed would be in the context of business to business. Many companies have moved entirely onto the cloud and adopted a microservcie architecture. In such set ups the entire tech stack is composed of many individual programs which each perform a specific function and run as ephemeral instances on one of the major cloud providers.

These services usually run in a shared virtual private cloud and communicate primarily via standard REST HTTP requests or via event streaming systems. In this type of architecture an API proxy service could be implemented which would expose a regular REST API to the internal network and transparently encrypt the request and forward it to the external 1dPIR service. It would then receive the encrypted response, perform the decryption and respond to the original REST request. To the internal network it would appear operate in the same way every other service does while in the background transforming it into a PIR request and maintaining zero knowledge to the external provider. Such an abstraction layer could potentially allow for much smoother integration and adoption as well as enable higher bandwidth and transfer speeds and therefore enable larger database sizes.

9 Conclusion

Going into the project it was known that there will necessarily be overheads incurred by any private information retrieval construction compared to a standard database query.

The primary questions we set out to answer are how large are these overheads, what factors influence them and by how much. If the scheme is practical and if so under what conditions or what are the primary limiting factors to this scheme. We also focused on the protocol primarily in the context of the data enrichment use case where the scheme is implemented as a service and we are primarily concerned with minimising the overheads incurred by the client.

Several experiments were run which involved running the various steps of the protocol independently under varying conditions and measuring the time taken and the memory usage required to perform the step. The scheme was also examined analytically to determine the various relationships between the parameters.

From the experiments and analysis it was found that, at least for the implementation under the GM cryptosystem, while the scheme does work and some operations such as encryption and decryption were faster than expected, it is ultimately restricted by the relationship between the size of the encrypted data which must be sent to the server and the total number of elements in the database.

This bound on the number of database elements for a given query is inherent to the cryptosystem and severely limits the real world use cases. Using a well known dataset as a bench mark, even stretching all the constraints to the limits of where they could be considered feasible and applying additional theoretic mitigations, this implementation does not come close to being practical for large scale data enrichment.

It was also found that the time and memory requirements grow with the size of the database and quickly become prohibitively large, although as the most demanding steps only need to be performed once and were also identified to be highly parallizable this might be acceptable. In general however it was found that this scheme is only practical for database sizes up to around to around a few million records.

Despite this, some potential optimizations were identified that may warrant further investigation and may raise the bounds of practicality.

In conclusion this particular 1dPIR construction is semi-practical and may be useful in some very specific cases but is not practical for the use case of privacy preserving data enrichment.

References

[1] R. Ostrovsky and W. E. Skeith, "A Survey of Single-Database Private Information Retrieval: Techniques and Applications," in *Public Key Cryptography – PKC 2007*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2007, pp. 393–411.

[2] G. González Fuster and A. Scherrer, *Big Data and Smart Devices and Their Impact on Privacy*. Jan. 1, 2015.

[3] Council of European Union, *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*, 2016.

[4] Openprise, "The Complete Enrichment Survival Guide for Marketing and Sales," Openprise, 2021.

[5] W. Gasarch, "A Survey on Private Information Retrieval," *Bulletin of the EATCS*, vol. 82, pp. 72–107, 2004.

[6] C. Gentry and Z. Ramzan, "Single-Database Private Information Retrieval with Constant Communication Rate," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, vol. 3580, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 803–815.

[7] B. Chor, N. Gilboa, and M. Naor, *Private information retrieval by keywords*, 1998.

[8] J. Carbino, "State of Third-Party Risk Management," Venminder, 2021.

[9] A. Kiayias, N. Leonardos, H. Lipmaa, *et al.*, "Optimal Rate Private Information Retrieval from Homomorphic Encryption," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 222–243, Jun. 1, 2015.

[10] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, Apr. 1984.

[11] J. Kelsey, "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash," *NIST Special Publication*, p. 32,

[12] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques," National Institute of Standards and Technology, NIST Special Publication (SP) 800-38A, Dec. 1, 2001.

[13] G. Bertoni, J. Daemen, M. Peeters, *et al.*, "KangarooTwelve: Fast hashing based on Keccak-p," p. 23,

[14] The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS," Apr. 2003.

[15] M. Albrecht and G. Bard, *The M4RI Library*, The M4RI Team.

[16] M. Albrecht, G. Bard, and W. Hart, "Efficient Multiplication of Dense Matrices over GF(2)," *ACM Transactions on Mathematical Software*, vol. 37, no. 1, pp. 1–14, Jan. 2010. arXiv: `0811.1714`.

[17] G. V. Bard, "The Method of Four Russians," in *Algebraic Cryptanalysis*, Boston, MA: Springer US, 2009, pp. 133–158.

[18] (Aug. 2021). Amazon EC2 Instance Types, Amazon Web Services, [Online]. Available: `https://aws.amazon.com/ec2/instance-types/`.

[19] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, ser. CRC Press Series on Discrete Mathematics and Its Applications. Boca Raton: CRC Press, 1997, 780 pp.

[20] E. B. Barker and Q. H. Dang, "Recommendation for Key Management Part 3: Application-Specific Key Management Guidance," National Institute of Standards and Technology, NIST SP 800-57Pt3r1, Jan. 2015, NIST SP 800–57Pt3r1.

[21] P. L. Montgomery, "Modular Multiplication Without Trial Division," p. 3,

[22] T. B. Brown, B. Mann, N. Ryder, *et al.* (Jul. 22, 2020). Language Models are Few-Shot Learners. arXiv: `2005.14165 [cs]`, [Online]. Available: `http://arxiv.org/abs/2005.14165`.

[23] "Scalable Zero Knowledge via Cycles of Elliptic Curves (extended version)," p. 49,

[24] A. R. Benson and G. Ballard, "A Framework for Practical Parallel Fast Matrix Multiplication," *ACM SIGPLAN Notices*, vol. 50, no. 8, pp. 42–53, Dec. 18, 2015. arXiv: `1409.2908`.

[25] (Mar. 10, 2021). How to Use a Data Breach for User Verification, SEON, [Online]. Available: `https://seon.io/resources/data-breach-for-user-verification/`.

[26] "Synthetic Identity Fraud in the U.S. Payment System A Review of Causes and Contributing Factors," p. 19, 2019.

[27] (Jan. 16, 2019). The 773 Million Record "Collection #1" Data Breach, Troy Hunt, [Online]. Available: `https://www.troyhunt.com/the-773-million-record-collection-1-data-reach/`.