Ethereum PIR - Spiral

December 24, 2023

Contents

1 Overview

An important dependancy of the Ethereum PIR architecture relies in the ability to privately retreive and process queries from a PIR database. Our implementation makes use of Spiral, which defines a family of single-server PIR protocols that relies on the composition of two lattice-based homomorphic encryption schemes: the Regev encryption scheme and the GentrySahai-Waters encryption scheme [1]. Spiral proposes a range of ciphertext translation techniques to convert between these two schemes and in doing so, is able to achieve at least a 4.5x reduction in query size, 1.5x reduction in response size, and a 2x increase in server throughput compared to previous systems[1]. The *"vanilla"* variant of Spiral is used throughout our implementation.

1.1 Database

The database of $N = 2^{v_1+v_2}$ (where $v_1, v_2 \in [2, 11]$) records is arranged as a hypercube with dimensions $2^{v_1} \times 2 \times 2 \times \cdots \times 2$. Processing the initial (large) dimension requires scalar multiplication (given the database is known) and is implemented using matrix Regev encodings. After processing the first dimension, the server has a $(2 \times 2 \times \cdots \times 2)$-hypercube containing 2^{v_2} matrix-Regev encodings. The client's index for each of the subsequent dimensions is encoded using GSW, so using v_2 rounds of the Regev-GSW homomorphic multiplication, the server can "fold" the remaining elements into a single matrix Regev encoding.

Database structure. Each database record d_i is an element of $R_p^{n \times n}$, where $\|d_i\|_\infty \leq p/2$. Spiral represents a database $\mathcal{D} = \{d_1, \ldots, d_N\}$ of $N = 2^{v_1+v_2}$ records as a $(v_2 + 1)$-dimensional hypercube with dimensions $2^{v_1} \times 2 \times 2 \times \cdots \times 2$. In the following description, Spiral index elements of $\mathcal{D}$ using either the tuple $(i, j_1, \ldots, j_{v_2})$ where $i \in [0, 2^{v_1} - 1]$ and $j_1, \ldots, j_{v_2} \in \{0, 1\}$, or the tuple (i, j) where $i \in [0, 2^{v_1} - 1]$ and $j \in [0, 2^{v_2} - 1]$.

1.2 The Spiral Protocol

The main steps of the Spiral protocol is as follows:

[1]Previous systems are defiend as SealPIR, FastPIR, MulPIR and OnionPIR. A complete and thorough analysis between these systems may be found in the original paper [1]

- Query generation: The client's query consists of a single scalar Regev ciphertext that encodes the record index the client wants to retrieve. Spiral structures the database of $N = 2^{v_1 \times v_2}$ records as a $2^{v_1} \times 2 \times \cdots \times 2$ hypercube. A record index can then be described by a tuple $(i, j_1, \ldots, j_{v_2})$ where $i \in \{0, \ldots, 2^{v_1} - 1\}$ and $j_1, \ldots, j_{v_2} \in \{0, 1\}$. The query consists of an encoding of the vector $(i, j_1, \ldots, j_{v_2})$, which Spiral can then pack into a single scalar Regev ciphertext.
- Query expansion: Upon receiving the client's query, the server expands the query ciphertext as follows:
  - Initial expansion: The server starts by expanding the query into a collection of (scalar) Regev ciphertexts that encode the queried index $(i, j_1, \ldots, j_{v_2})$. This will yield two collections of Regev ciphertexts, which is denoted by C_{Reg} and C_{GSW}.
  - First dimension expansion: Next, the server uses C_{Reg} to expand the ciphertexts into a collection of 2^{v_1} matrix Regev ciphertexts that "indicate" index i : namely, the i^{th} ciphertext is an encryption of 1 while the remaining ciphertexts are encryptions of 0. Spiral then views this collection of ciphertexts as an encryption of the i^{th} basis vector. This step relies on a scalar-to-matrix algorithm ScalToMat that takes a Regev ciphertext encrypting a bit $\mu \in \{0, 1\}$ and outputs a matrix Regev ciphertext that encrypts the matrix $\mu \mathbf{I}_n$, where $\mathbf{I}_n$ is the n-by- n identity matrix.
  - GSW ciphertext expansion: The server then uses C_{GSW} to construct GSW encryptions of the indices $j_1, \ldots, j_{v_2} \in \{0, 1\}$. This step relies on a Regev-to-GSW translation algorithm RegevToGSW in [1].
- Query processing: After expanding the query into matrix Regev encryptions of the first dimension and GSW encryptions of the subsequent dimension, the server homomorphically computes the response as follows:
  - First dimension processing: First, it uses the matrix Regev encryptions of the i^{th} basis vector to project the database onto the sub-database of records whose first index is i. This step only requires linear homomorphisms since the database records are available in the clear while the query is encrypted. At the end of this step, the server has matrix Regev encryptions of the projected database.
  - Folding in subsequent dimensions: Next, the server uses the Regev-GSW external product to homomorphically multiply in the GSW ciphertexts encrypting the subsequent queries. Each GSW ciphertext selects for one of two possible dimensions.

Since each multiplication involves a ”fresh” GSW ciphertext derived from the original query, Spiral can then take advantage of the asymmetric noise growth property of Regev-GSW multiplication. The result is a single matrix Regev ciphertext encrypting the desired record.

- Response decoding: At the conclusion of the above protocol, the server replies with a single matrix Regev ciphertext encrypting the desired record.

A high-level illustration of this protocol is provided in Fig. 1.

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the database size N, the setup algorithm proceeds as follows:
  1) **Key-generation:** Sample two secret keys $\mathbf{S} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n)$ and $\mathbf{s} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^1)$ that are used for response encoding and query encoding, respectively.
  2) **Regev-to-GSW conversion keys:** Compute $\mathsf{ck} \leftarrow \mathsf{RegevToGSWSetup}(\mathbf{s}, \mathbf{S}, z_{\mathsf{conv}})$.
  3) **Automorphism keys:** Let $\rho = 1 + \max(\nu_1, \lceil \log t_{\mathsf{GSW}} \nu_2 \rceil)$. For each $i \in [0, \rho - 1]$, compute $\mathbf{W}_i \leftarrow \mathsf{AutomorphSetup}(\mathbf{s}, \tau_{2^{\rho-i}+1}, z_{\mathsf{coeff}})$.

  Output the public parameters $\mathsf{pp} = (\mathsf{ck}, \mathbf{W}_0, \ldots, \mathbf{W}_{\rho-1})$ and the querying key $\mathsf{qk} = (\mathbf{s}, \mathbf{S})$.
- $\mathsf{Query}(\mathsf{qk}, \mathsf{idx})$: On input the querying key $\mathsf{qk} = (\mathbf{s}, \mathbf{S})$ and an index $\mathsf{idx} = (i^*, j_1^*, \ldots, j_{\nu_2}^*)$ where $i^* \in [0, 2^{\nu_1} - 1]$ and $j_1^*, \ldots, j_{\nu_2}^* \in \{0, 1\}$, the query algorithm does the following:
  1) **Encoding the first dimension:** Define the polynomial $\mu_{i^*}(x) = \lfloor q/p \rfloor \cdot x^{i^*} \in R_q$.
  2) **Encoding subsequent dimensions:** Define the polynomial $\mu_{j^*} = \sum_{\ell \in [\nu_2]} \mu_{j_\ell^*}$ where for each $\ell \in [\nu_2]$,

$$\mu_{j_\ell^*}(x) = j_\ell^* \sum_{k \in [t_{\mathsf{GSW}}]} (z_{\mathsf{GSW}})^{k-1} x^{(\ell-1)t_{\mathsf{GSW}}+k}.$$

  3) **Query packing:** Define the “packed” polynomial

$$\mu(x) := 2^{-r_1} \mu_{i^*}(x^2) + 2^{-r_2} x \mu_{j^*}(x^2) \in R_q, \tag{IV.1}$$

  where $r_1 = 1 + \nu_1$ and $r_2 = 1 + \lceil \log(t_{\mathsf{GSW}} \nu_2) \rceil$.
  4) **Query encryption:** Compute the encrypted query $\mathbf{c} \leftarrow \mathsf{Regev.Encode}(\mathbf{s}, \mu) \in R_q^2$. Output the query $\mathsf{q} = \mathbf{c}$ and an empty query state $\mathsf{st} = \perp$.
- $\mathsf{Answer}(\mathsf{pp}, \mathcal{D}, \mathsf{q})$: On input the database $\mathcal{D}$, the public parameters $\mathsf{pp} = (\mathsf{ck}, \mathbf{W}_1, \ldots, \mathbf{W}_\rho)$, and a query $\mathsf{q} = \mathbf{c}$, the server response algorithm parses $\mathsf{ck} = (\mathbf{V}, \mathbf{W}, \mathbf{\Pi})$ and proceeds as follows:
  1) **Query expansion:** The server expands the query ciphertext $\mathbf{c}$ into 2^{ν_1} matrix Regev encodings (for the first dimension) and ν_2 GSW encodings (for the subsequent dimensions) as follows:
     a) **Initial expansion:** Homomorphically evaluate a single iteration of the coefficient expansion algorithm (Algorithm 1) on $\mathbf{c}$. Let $\mathbf{c}_{\mathsf{Reg}}, \mathbf{c}_{\mathsf{GSW}} \in R_q^2$ be the output encodings.
     b) **First dimension expansion:** Continue homomorphic evaluation of Algorithm 1 for ν_1 additional iterations on $\mathbf{c}_{\mathsf{Reg}}$ to obtain encodings $\mathbf{c}_1^{(\mathsf{Reg})}, \ldots, \mathbf{c}_{2^{\nu_1}}^{(\mathsf{Reg})} \in R_q^2$. For each $i \in [0, 2^{\nu_1} - 1]$, let $\mathbf{C}_i^{(\mathsf{Reg})} \leftarrow \mathsf{ScalToMat}(\mathbf{W}, \mathbf{c}_i^{(\mathsf{Reg})})$.
     c) **GSW ciphertext expansion:** Continue homomorphic evaluation of Algorithm 1 for $\lceil \log(t_{\mathsf{GSW}} \nu_2) \rceil$ additional iterations on $\mathbf{c}_{\mathsf{GSW}}$ to obtain encodings $\mathbf{c}_1^{(\mathsf{GSW})}, \ldots, \mathbf{c}_{t_{\mathsf{GSW}}\nu_2}^{(\mathsf{GSW})} \in R_q^2$. Discard any additional encodings output by Algorithm 1 whenever $t_{\mathsf{GSW}} \nu_2$ is not a power of two. For each $j \in [\nu_2]$, compute $\mathbf{C}_j^{(\mathsf{GSW})} \leftarrow \mathsf{RegevToGSW}\big(\mathsf{ck}, \mathbf{c}_{(j-1)t_{\mathsf{GSW}}+1}^{(\mathsf{GSW})}, \ldots, \mathbf{c}_{jt_{\mathsf{GSW}}}^{(\mathsf{GSW})}\big)$.

     Note that the above invocations of Algorithm 1 will use the automorphism keys $\mathbf{W}_0, \ldots, \mathbf{W}_{\rho-1}$.
  2) **Processing the first dimension:** For every $j \in [0, 2^{\nu_2} - 1]$, the server does the following:
     a) Initialize $\mathbf{C}_j^{(0)} \leftarrow \mathsf{ScalarMul}\big(\mathbf{C}_0^{(\mathsf{Reg})}, d_{0,j}\big)$.
     b) For each $i \in [2^{\nu_1} - 1]$, update $\mathbf{C}_j^{(0)} \leftarrow \mathsf{Add}\big(\mathbf{C}_j^{(0)}, \mathsf{ScalarMul}\big(\mathbf{C}_i^{(\mathsf{Reg})}, d_{i,j}\big)\big)$.
  3) **Folding in the subsequent dimensions:** For each $r \in [\nu_2]$, and each $j \in [0, 2^{\nu_2 - r} - 1]$, compute

$$\mathbf{C}_j^{(r)} = \mathsf{Add}\left(\mathsf{Multiply}\left(\mathsf{Complement}(\mathbf{C}_r^{(\mathsf{GSW})}), \mathbf{C}_j^{(r-1)}\right), \mathsf{Multiply}\left(\mathbf{C}_r^{(\mathsf{GSW})}, \mathbf{C}_{2^{\nu_2-r}+j}^{(r-1)}\right)\right). \tag{IV.2}$$

  4) **Modulus switching:** Output the rescaled response $\mathsf{r} \leftarrow \mathsf{ModulusSwitch}_{q_1,q_2}(\mathbf{C}_0^{(r)})$.
- $\mathsf{Extract}(\mathsf{qk}, \mathsf{st}, \mathsf{r})$: On input the query key $\mathsf{qk} = (\mathbf{s}, \mathbf{S})$, an (empty) query state st, and the server response r, the extraction algorithm first computes $\mathbf{Z} \leftarrow \mathsf{Recover}_{q_1,q_2}(\mathbf{S}, \mathsf{r}) \in R_{q_1}^{n \times n}$ and outputs $\mathbf{C} \leftarrow \mathsf{Decode}(\mathbf{Z}) \in R_p^{n \times n}$.

Figure 1: The SPIRAL PIR protocol.

2 Implementation

In our architecture, we have seperated the Spiral codebase into individual client and server processes. Specifically, the client is responsible for the Setup $(1^\lambda, 1^N)$, Query(qk, idx) and Extract(qk, st, r) stages of Fig. 1, while the server is responsible for the Answer$(\text{pp}, \mathcal{D}, \text{q})$ process.

In addition to the above, we have extended the existing Spiral implementation to load in a set of 32-byte hashes from a pre-defined database file. These set of hashes are defined by the colouring or PBC algorithms, and the hashes are loaded in sequence into the database.

Furthermore, Spiral attempts to greatly reduce the size of the database by encoding multiple hashes into a single database record. This results in faster query times while lowering the overall in-memory storage requirement for the server. However, a consequence to this approach will result in the server sending back the requested hash nested in other non-queried, but surrounding hashes in the database. Therefore, the client maintains the ability to *extract* the queried hash from the received database record.

Likewise, the server attempts to compress as many hashes into every sequnetial record it creates when building the database. A categorical explanation of these encoding methods will be mentioned in Encoding strategies.

In scenarios where the database size exceeds the number of hashes to load, the database is padded with dummy polynomials, where every coefficient is set to zero. This is to ensure database symmetry is maintained, and the server is able to perform matrix operations across the database correctly.

2.1 Encoding strategies

Recall that a database record d_i is an element of $R_p^{n \times n}$, where data is encoded over the coefficients of a ring polynomial. The encoding strategy that is used to compress multiple hashes into a single record will be determined by the plaintext modulus p. The p value is set to be a power of two with a maximum value of 2^{30} and is selected during automatic parameter selection[1].

The following table shows the relationship between a database size and the plaintext modulus p:

Database Size	Plaintext Modulus	Encoding Strategy
2^{10} up to 2^{17}	4	Bucketing
2^{18} up to 2^{21}	16	Linear
2^{22} up to 2^{30}	256	Packing

Table 1: Relationship between database size and plaintext modulus.

The following sections will briefly explain each strategy.

2.1.1 Bucketing

Bucketing is used when the maximum data representation provided by a single coefficent is 2 bits. Again, this is used when p is 4 and the value of a coefficient must wrap around the plaintext modulus.

Bucketing involves storing a single hex charecter of a hash across multiple coefficients. This will result in a *bucket* of coefficients which will combinatorily represent a single hex charecter as determined by a pre-defined lookup table. The resulting encoding will be the length of the input hash $\times$the number of coefficients in a bucket.

2.1.2 Linear

When p is 16, we can directly represent a hex charecter in a single coefficient. The resulting encoding will be the length of the input hash.

2.1.3 Packing

Packing involves bit packing two charecters into a single coefficient when p is 256. We can use a 256 decimal value to binary pack two 4-bit integers, where a hexadecimal charecter can be represent is 4-bits. This will result in an encoding that is half the size of the input hash.

3 Experiments & Evaluation

This section will benchmark the performance of Spiral against the coulouring and PBC datasets of our study.

3.1 Experimental Setup

Spiral Parameters We run our server and client executables over the same set of parameters. These parameters are selected through the automatic parameter selection[1], and is determined before the server and client are executed.

Hardware Evaluation of Spiral is performed over a `M5zn` AWS instance with 32 vCPUs and 128GiB of memory. `AVX512F` and `AVX2` support was enabled, and the instance was running on Ubuntu 20.04. We used `Clang 12` and compiled with `O3` optimisations.

Trials Trials are ran over a pre-defiend indices file generated by the colouring and PBC algorithms. These set of indices are a realistic depiction of how hashes will be queried in a realistic implementation.

Measurements Time measurements are taken using the `std::chrono::high_resolution_clock`. The average, minimum and maximum values are recorded for each metric. These averages are taken over the number of indices specified in a single indices file.

3.2 Metrics

We use the following metrics to evaluate the performance of Spiral:

- **Database-Dependant Compuation** The time taken by the server to process the query against the database and produce a result.
- **Database Generation** The time required to load and encoded a database of size N into memory on the server.
- **Query Generation** The time taken to generate a client-side query.
- **Response Extraction** The time taken to decode and extract the requested hash from the server response on the client.

- **Rate** The ratio of the response size to the size of the retrieved record. The rate stands to measure the overehead in the server-client communication.
- **Query Size** The size of the encoded query sent by the client to the server.
- **Response Size** The size of the received encoded response from the server.
- **Total Cost** The addtion of the query and response sizes.

3.3 Evaluation Results

3.3.1 Colouring

Table 2: **Average** Database Generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	353860.400	355294.636	357511.833	355324.615	360423.857	358491.333	371022.188	379311.706	406762.500	456448.211	904524.300	830715.667	1660272.727	3542360.435	7466600.417
16	NA	NA	354618.333	NA	NA	NA	384113.500	NA	NA	NA	844344.200	NA	NA	NA	7977308.333
128	NA	NA	NA	NA	370065.000	NA	NA	NA	NA	NA	NA	3513343.333	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	407814.500	NA	NA	NA	NA	NA	NA	NA	16148866.667

Table 3: **Min** Database Generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	352086.000	352238.000	352600.000	351499.000	358800.000	355466.000	367549.000	376227.000	404482.000	453702.000	901501.000	827848.000	1650040.000	3479510.000	7374400.000
16	NA	NA	352051.000	NA	NA	NA	383597.000	NA	NA	NA	839290.000	NA	NA	NA	7938300.000
128	NA	NA	NA	NA	369325.000	NA	NA	NA	NA	NA	NA	3492680.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	406270.000	NA	NA	NA	NA	NA	NA	NA	16038400.000

Table 4: **Max** Database Generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	355405.000	357555.000	359218.000	359891.000	362503.000	361552.000	372233.000	381380.000	410025.000	458209.000	908476.000	834783.000	1681920.000	3587380.000	7519070.000
16	NA	NA	358147.000	NA	NA	NA	384941.000	NA	NA	NA	849469.000	NA	NA	NA	7999380.000
128	NA	NA	NA	NA	370805.000	NA	NA	NA	NA	NA	NA	3540720.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	409359.000	NA	NA	NA	NA	NA	NA	NA	16276800.000

Table 5: **Average** Query generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	3773.880	3862.964	3865.833	3881.154	3866.286	3858.167	3866.300	3864.753	3819.756	3809.805	2308.110	6573.538	6414.500	5791.917	12645.454
16	NA	NA	3872.000	NA	NA	NA	3819.300	NA	NA	NA	6580.500	NA	NA	NA	12592.817
128	NA	NA	NA	NA	3869.300	NA	NA	NA	NA	NA	NA	5776.567	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	3807.500	NA	NA	NA	NA	NA	NA	NA	25337.767

Table 6: **Min** Query generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	3694.000	3701.000	3702.000	3707.000	3696.000	3692.000	3704.000	3683.000	3706.000	3699.000	2049.000	6355.000	6193.000	5195.000	12144.000
16	NA	NA	3720.000	NA	NA	NA	3706.000	NA	NA	NA	6375.000	NA	NA	NA	11964.000
128	NA	NA	NA	NA	3719.000	NA	NA	NA	NA	NA	NA	5235.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	3704.000	NA	NA	NA	NA	NA	NA	NA	24690.000

Table 7: **Max** Query generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	4390.000	4448.000	4420.000	4505.000	4454.000	4476.000	4474.000	4385.000	4416.000	4411.000	4370.000	7983.000	8357.000	7934.000	15008.000
16	NA	NA	4378.000	NA	NA	NA	4367.000	NA	NA	NA	7856.000	NA	NA	NA	14824.000
128	NA	NA	NA	NA	4400.000	NA	NA	NA	NA	NA	NA	7842.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	4454.000	NA	NA	NA	NA	NA	NA	NA	31403.000

Table 8: **Average** Database-dependant computation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	57316.980	57108.373	57318.825	57566.123	57448.607	57321.200	57274.488	57091.659	57261.717	57900.847	121588.920	78148.705	161403.841	336737.022	430979.271
16	NA	NA	57916.600	NA	NA	NA	57928.725	NA	NA	NA	77967.160	NA	NA	NA	547847.483
128	NA	NA	NA	NA	56640.500	NA	NA	NA	NA	NA	NA	335396.967	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	57075.950	NA	NA	NA	NA	NA	NA	NA	720810.700

Table 9: **Min** Database-dependant computation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	55999.000	55491.000	55881.000	56485.000	56178.000	55730.000	55845.000	55883.000	56381.000	55853.000	118969.000	75844.000	158261.000	330472.000	419124.000
16	NA	NA	56849.000	NA	NA	NA	56427.000	NA	NA	NA	75004.000	NA	NA	NA	540421.000
128	NA	NA	NA	NA	56066.000	NA	NA	NA	NA	NA	NA	329551.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	56711.000	NA	NA	NA	NA	NA	NA	NA	708649.000

Table 10: **Max** Database-dependant computation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	59070.000	58307.000	58450.000	58966.000	58687.000	58771.000	58498.000	58196.000	59054.000	60341.000	124799.000	80975.000	166049.000	346618.000	443440.000
16	NA	NA	58943.000	NA	NA	NA	58561.000	NA	NA	NA	80257.000	NA	NA	NA	553067.000
128	NA	NA	NA	NA	57699.000	NA	NA	NA	NA	NA	NA	341178.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	57770.000	NA	NA	NA	NA	NA	NA	NA	733579.000

Table 11: **Average** Response extraction (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	1394.273	1417.855	1424.392	1447.723	1468.793	1482.480	1537.500	1637.947	1739.461	2143.953	2801.725	4564.576	4561.195	9719.074	23280.092
16	NA	NA	1444.400	NA	NA	NA	1616.175	NA	NA	NA	4658.500	NA	NA	NA	47242.633
128	NA	NA	NA	NA	1511.950	NA	NA	NA	NA	NA	NA	5895.767	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	1628.350	NA	NA	NA	NA	NA	NA	NA	68564.933

Table 12: **Min** Response extraction (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	1301.000	1296.000	1300.000	1358.000	1369.000	1318.000	1338.000	1362.000	1324.000	1365.000	1332.000	1359.000	1319.000	1366.000	1306.000
16	NA	NA	1367.000	NA	NA	NA	1369.000	NA	NA	NA	1364.000	NA	NA	NA	1403.000
128	NA	NA	NA	NA	1372.000	NA	NA	NA	NA	NA	NA	1362.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	1371.000	NA	NA	NA	NA	NA	NA	NA	1430.000

Table 13: **Max** Response extraction (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	1691.000	1741.000	1724.000	1746.000	1760.000	1831.000	1743.000	1978.000	2402.000	3329.000	4971.000	8111.000	13699.000	23734.000	46440.000
16	NA	NA	1761.000	NA	NA	NA	2018.000	NA	NA	NA	8996.000	NA	NA	NA	95524.000
128	NA	NA	NA	NA	1814.000	NA	NA	NA	NA	NA	NA	18457.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	2283.000	NA	NA	NA	NA	NA	NA	NA	183618.000

Table 14: **Average** Rate (number of hashes per record)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.008	0.008	0.008	0.008
16	NA	NA	0.016	NA	NA	NA	0.016	NA	NA	NA	0.008	NA	NA	NA	0.004
128	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	0.008	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	NA	0.004

Table 15: **Min** Rate (number of hashes per record)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.008	0.008	0.008	0.008
16	NA	NA	0.016	NA	NA	NA	0.016	NA	NA	NA	0.008	NA	NA	NA	0.004
128	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	0.008	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	NA	0.004

Table 16: **Max** Rate (number of hashes per record)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.008	0.008	0.008	0.008
16	NA	NA	0.016	NA	NA	NA	0.016	NA	NA	NA	0.008	NA	NA	NA	0.004
128	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	0.008	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	NA	0.004

Table 17: **Average** Query size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000
16	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000
128	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	NA	28672.000

Table 18: **Min** Query size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000
16	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000
128	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	NA	28672.000

Table 19: **Max** Query size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000
16	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000
128	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	NA	28672.000

Table 20: **Average** Response size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	14336.000	14336.000	14336.000	14336.000
16	NA	NA	11264.000	NA	NA	NA	11264.000	NA	NA	NA	14336.000	NA	NA	NA	20992.000
128	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	14336.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	20992.000

Table 21: **Min** Response size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	14336.000	14336.000	14336.000	14336.000
16	NA	NA	11264.000	NA	NA	NA	11264.000	NA	NA	NA	14336.000	NA	NA	NA	20992.000
128	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	14336.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	20992.000

Table 22: **Max** Response size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	14336.000	14336.000	14336.000	14336.000
16	NA	NA	11264.000	NA	NA	NA	11264.000	NA	NA	NA	14336.000	NA	NA	NA	20992.000
128	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	14336.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	20992.000

Table 23: **Average** Total cost (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	43008.000	43008.000	43008.000	43008.000
16	NA	NA	39936.000	NA	NA	NA	39936.000	NA	NA	NA	43008.000	NA	NA	NA	49664.000
128	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	43008.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	NA	49664.000

Table 24: **Min** Total cost (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	43008.000	43008.000	43008.000	43008.000
16	NA	NA	39936.000	NA	NA	NA	39936.000	NA	NA	NA	43008.000	NA	NA	NA	49664.000
128	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	43008.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	NA	49664.000

Table 25: **Max** Total cost (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	43008.000	43008.000	43008.000	43008.000
16	NA	NA	39936.000	NA	NA	NA	39936.000	NA	NA	NA	43008.000	NA	NA	NA	49664.000
128	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	43008.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	NA	49664.000

3.3.2 PBC

Table 26: **Average** Database Generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	353404.800	356373.706	354553.167	356796.250	366146.619	368735.783	388051.667	414956.846	460634.593	913095.759	817589.767	1632080.938	3481143.030	7334201.714	7832016.389
16	NA	NA	347048.200	NA	NA	NA	414355.167	NA	NA	NA	1649953.750	NA	NA	NA	16367688.889
128	NA	NA	NA	NA	378010.667	NA	NA	NA	NA	NA	NA	7218398.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	822721.000	NA	NA	NA	NA	NA	NA	NA	34810240.000

Table 27: **Min** Database Generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	349699.000	349408.000	352148.000	353974.000	361188.000	365992.000	385791.000	412529.000	458809.000	908476.000	814252.000	1619220.000	3438160.000	7289880.000	7773870.000
16	NA	NA	345740.000	NA	NA	NA	413125.000	NA	NA	NA	1646460.000	NA	NA	NA	16291000.000
128	NA	NA	NA	NA	377270.000	NA	NA	NA	NA	NA	NA	7192980.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	818682.000	NA	NA	NA	NA	NA	NA	NA	34511200.000

Table 28: **Max** Database Generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	349699.000	361339.000	358688.000	359530.000	369829.000	370939.000	392160.000	417158.000	462321.000	917304.000	821251.000	1654050.000	3510440.000	7413630.000	7932170.000
16	NA	NA	347747.000	NA	NA	NA	415598.000	NA	NA	NA	1658090.000	NA	NA	NA	16442400.000
128	NA	NA	NA	NA	379264.000	NA	NA	NA	NA	NA	NA	7233520.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	824802.000	NA	NA	NA	NA	NA	NA	NA	35220600.000

Table 29: **Average** Query generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	3875.520	3859.194	3878.606	3870.835	3872.819	3877.491	3811.167	3820.308	3878.770	2293.007	6565.100	6417.413	5760.124	12614.294	12576.911
16	NA	NA	3858.520	NA	NA	NA	3798.317	NA	NA	NA	6393.288	NA	NA	NA	25166.778
128	NA	NA	NA	NA	3794.900	NA	NA	NA	NA	NA	NA	12587.400	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	2329.367	NA	NA	NA	NA	NA	NA	NA	22265.880

Table 30: **Min** Query generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	3696.000	3687.000	3705.000	3703.000	3689.000	3707.000	3698.000	3707.000	3701.000	2041.000	6353.000	6188.000	5201.000	12126.000	11950.000
16	NA	NA	3707.000	NA	NA	NA	3696.000	NA	NA	NA	6183.000	NA	NA	NA	24659.000
128	NA	NA	NA	NA	3691.000	NA	NA	NA	NA	NA	NA	12120.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	2069.000	NA	NA	NA	NA	NA	NA	NA	21468.000

Table 31: **Max** Query generation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	4559.000	4419.000	4438.000	4476.000	4502.000	4494.000	4431.000	4428.000	4545.000	4406.000	7907.000	7932.000	7874.000	16106.000	14979.000
16	NA	NA	4445.000	NA	NA	NA	4389.000	NA	NA	NA	7838.000	NA	NA	NA	28788.000
128	NA	NA	NA	NA	4377.000	NA	NA	NA	NA	NA	NA	14779.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	4352.000	NA	NA	NA	NA	NA	NA	NA	28772.000

Table 32: **Average** Database-dependant computation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	57188.420	57206.394	57232.711	57074.590	57287.376	57430.100	57179.829	57347.488	57289.770	121332.517	78109.307	159994.909	336840.673	428128.763	548335.314
16	NA	NA	57027.980	NA	NA	NA	57297.450	NA	NA	NA	160486.888	NA	NA	NA	721724.778
128	NA	NA	NA	NA	56228.433	NA	NA	NA	NA	NA	NA	426167.920	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	120954.167	NA	NA	NA	NA	NA	NA	NA	1591304.000

Table 33: **Min** Database-dependant computation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	55560.000	56306.000	55978.000	55775.000	56074.000	55964.000	55582.000	55940.000	55847.000	118189.000	76053.000	156158.000	328160.000	415784.000	536162.000
16	NA	NA	55848.000	NA	NA	NA	56526.000	NA	NA	NA	155756.000	NA	NA	NA	699408.000
128	NA	NA	NA	NA	55375.000	NA	NA	NA	NA	NA	NA	416409.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	118145.000	NA	NA	NA	NA	NA	NA	NA	1552570.000

Table 34: **Max** Database-dependant computation (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	58523.000	58637.000	59006.000	60129.000	59137.000	59447.000	59477.000	58865.000	59735.000	125252.000	81596.000	164750.000	348963.000	442533.000	557970.000
16	NA	NA	58930.000	NA	NA	NA	58475.000	NA	NA	NA	165621.000	NA	NA	NA	732098.000
128	NA	NA	NA	NA	57729.000	NA	NA	NA	NA	NA	NA	434032.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	124426.000	NA	NA	NA	NA	NA	NA	NA	1612920.000

Table 35: **Average** Response extraction (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	1410.393	1410.547	1414.683	1416.395	1425.633	1436.809	1420.575	1447.831	1551.319	1565.397	1753.870	2063.278	2609.227	3357.877	5384.814
16	NA	NA	1420.060	NA	NA	NA	1523.433	NA	NA	NA	2424.338	NA	NA	NA	13684.000
128	NA	NA	NA	NA	1477.600	NA	NA	NA	NA	NA	NA	4627.080	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	1696.333	NA	NA	NA	NA	NA	NA	NA	28593.180

Table 36: **Min** Response extraction (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	1306.000	1297.000	1293.000	1295.000	1309.000	1297.000	1299.000	1303.000	1312.000	1294.000	1294.000	1296.000	1296.000	1311.000	1389.000
16	NA	NA	1298.000	NA	NA	NA	1365.000	NA	NA	NA	1295.000	NA	NA	NA	1449.000
128	NA	NA	NA	NA	1365.000	NA	NA	NA	NA	NA	NA	1322.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	1325.000	NA	NA	NA	NA	NA	NA	NA	1436.000

Table 37: **Max** Response extraction (µs)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	1702.000	1715.000	1737.000	1780.000	1799.000	1882.000	1934.000	2411.000	3285.000	4563.000	8371.000	13322.000	24807.000	46210.000	96498.000
16	NA	NA	1752.000	NA	NA	NA	2561.000	NA	NA	NA	14894.000	NA	NA	NA	168364.000
128	NA	NA	NA	NA	1968.000	NA	NA	NA	NA	NA	NA	36915.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	3372.000	NA	NA	NA	NA	NA	NA	NA	335465.000

Table 38: **Average** Rate (number of hashes per record)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.008	0.008	0.008	0.008	0.004
16	NA	NA	0.016	NA	NA	NA	0.016	NA	NA	NA	0.008	NA	NA	NA	0.004
128	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	0.008	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	NA	0.004

Table 39: **Min** Rate (number of hashes per record)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.008	0.008	0.008	0.008	0.004
16	NA	NA	0.016	NA	NA	NA	0.016	NA	NA	NA	0.008	NA	NA	NA	0.004
128	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	0.008	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	NA	0.004

Table 40: **Max** Rate (number of hashes per record)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.008	0.008	0.008	0.008	0.004
16	NA	NA	0.016	NA	NA	NA	0.016	NA	NA	NA	0.008	NA	NA	NA	0.004
128	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	0.008	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	0.016	NA	NA	NA	NA	NA	NA	NA	0.004

Table 41: **Average** Query size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000
16	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000
128	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	NA	28672.000

Table 42: **Min** Query size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000
16	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000
128	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	NA	28672.000

Table 43: **Max** Query size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000	28672.000
16	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000	NA	NA	NA	28672.000
128	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	28672.000	NA	NA	NA	NA	NA	NA	NA	28672.000

Table 44: **Average** Response size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	14336.000	14336.000	14336.000	14336.000	20992.000
16	NA	NA	11264.000	NA	NA	NA	11264.000	NA	NA	NA	14336.000	NA	NA	NA	20992.000
128	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	14336.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	20480.000

Table 45: **Min** Response size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	14336.000	14336.000	14336.000	14336.000	20992.000
16	NA	NA	11264.000	NA	NA	NA	11264.000	NA	NA	NA	14336.000	NA	NA	NA	20992.000
128	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	14336.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	20480.000

Table 46: **Max** Response size (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	11264.000	14336.000	14336.000	14336.000	14336.000	20992.000
16	NA	NA	11264.000	NA	NA	NA	11264.000	NA	NA	NA	14336.000	NA	NA	NA	20992.000
128	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	11264.000	NA	NA	NA	NA	NA	NA	NA	20480.000

Table 47: **Average** Total cost (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	43008.000	43008.000	43008.000	43008.000	49664.000
16	NA	NA	39936.000	NA	NA	NA	39936.000	NA	NA	NA	43008.000	NA	NA	NA	49664.000
128	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	43008.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	NA	49152.000

Table 48: **Min** Total cost (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	43008.000	43008.000	43008.000	43008.000	49664.000
16	NA	NA	39936.000	NA	NA	NA	39936.000	NA	NA	NA	43008.000	NA	NA	NA	49664.000
128	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	43008.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	NA	49152.000

Table 49: **Max** Total cost (KB)

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
2	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	39936.000	43008.000	43008.000	43008.000	43008.000	49664.000
16	NA	NA	39936.000	NA	NA	NA	39936.000	NA	NA	NA	43008.000	NA	NA	NA	49664.000
128	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	43008.000	NA	NA	NA
256	NA	NA	NA	NA	NA	NA	39936.000	NA	NA	NA	NA	NA	NA	NA	49152.000

3.4 Discussion

The most extreme database configuration evaluated in the original paper for Spiral was 2^{20} elements where every element size is 256 bytes[1]. PIR schemes, such as Spiral, are known to struggle with larger databases, and we found the required configuration to store anything above 2^{26} elements to be infeasible for most machines. As an example, a database of around 2^{28} elements may require upwards of $700GiB$ of memory.

Database performance across our PBC and colouring datasets are shown to have very similar performance. This is bacause, for the most part, the database sizes and the number of hashes to store is almost identical. The only exception to this is when n is above 20, where n is the number of leaves in the tree. During such cases, the PBC dataset contains more hashes than the colouring, which results in a larger database size being used. This will lead to an increase in database-dependent computations, as well as larger response sizes.

Metrics across $10 \leq n \leq 19$ for the colouring dataset and $10 \leq n \leq 18$ for the PBC dataset are shown to be very similar. This is because Spiral's automatic parameter selection defined in [1] determines the same database size of 2^{10} elements over these values of n. There is a noticeable increase over database-dependant computation across $n = 2^{20}$ (colouring) and $n = 2^{19}$ (PBC). Both of these values utilises a borderline configuration with a p value of 4 and a database size of 2^{11} with the next configurartion using $p = 16$. This "borderline" configuration is likely to be the cause of this increase in computation time, as the parameter selection may struggle to select optimal parameters in these edge cases. Note that we did use the unchanged and original automatic parameter selection method defined in [1].

The *rate* of our trials appears to diminish significantly as the size of the required database increases. This pattern is also noted in the original paper[1], where a decrease in rate corresponds to an increase in database size. A possible explanation for this observation is that encoding more elements into a single database entry reduces the overall size of the database. Consequently, a smaller database size leads to quicker database-dependent computations, especially in operations involving first dimension multiplication and repeated folding across subsequent dimensions.

Given that we have defined three methodologies to encode hashes across three different plaintext moduli p, we can observe the rate to be of three different corresponding values: 0.031, 0.008 and 0.004. The first of which is seen on smaller databases, as a single hash

takes up a larger portion of the database record. Likewise, as the value of p increases, we are able to pack more hashes into a coeffiecient of the ring polynomial, and thus the rate decreases as more hashes are represented in a record.

The response size provided by the server follows a similar pattern to the rate, with the sizes gradually increasing as the p value increases. Again, this could be explained by the need to encode more hashes into a single record in the database. This retrieved record is what gets sent back, and thus the response size increases as the rate decreases.

Unlike the response size, the query size stays constant throughout all trials. This is because the procedure to encode the hash index is consistent and irrespective of the database size used.

The time required to extract the requested hash from the server response is also seen to slightly increase along with the database size. This is likely due to the fact that the encoding methodolgies for larger database sizes are more complex, and may therefore require more time to decode the hash from the server response.

References

[1] Samir Jordan Menon and David J. Wu. Spiral: Fast, high-rate single-server pir via fhe composition. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 930–947, 2022.