TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor Thesis

Effective Protection of Sensitive Metadata in Online Communication Networks

Marco Holz

October 18, 2016

CRISP
Center for Research
in Security and Privacy

Technische Universität Darmstadt
Center for Research in Security and Privacy
Engineering Cryptographic Protocols

Supervisors: Dr. Thomas Schneider
M.Sc. Daniel Demmler

Declaration of Authorship

I certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other University.

Darmstadt, October 18, 2016

_____

Marco Holz

Abstract

While a lot of effort has already been put into securing the contents of messages transmitted over our digital infrastructure, protection of metadata is mostly ignored because of the lack of available technologies that can be used to secure this valuable part of our communication. Scalable mechanisms to protect the anonymity of the users and hide their social graph are in great demand.

This bachelor's thesis introduces and evaluates two significant improvements in private information retrieval – an active field of research that allows private querying of data from a database without revealing which data has been requested and a fundamental building block for private communication networks –, summarizes past efforts that have been put into building such networks and proposes OnionPIR, a novel private messaging service built upon efficient information-theoretic PIR and onion routing. A prototype has been built to substantiate the claim that OnionPIR is usable in practice.

On the basis of the results of this research, it can be concluded that it would be possible to build and deploy such a service today. OnionPIR describes a way to operate the system by combining highly efficient PIR mechanisms and nearly zero-cost onion routing. Its server operating expenses are within the order of magnitude of those of traditional messaging services.

Acknowledgments

First, I wish to express my sincere thanks to my supervisors Thomas Schneider and Daniel Demmler for the great support during the last six month. Thanks for the helpful guidance, for pointing out new directions but also for giving me the freedom to try out new ideas and the flexibility to organize meetings according to my self-determined work rhythm. Their continuous, valuable feedback and comments turned out to be a very conducive contribution to this thesis. I am deeply grateful for the privilege and opportunity to work with them in this exciting topic of private communication.

I would also like to thank the whole team of the Engineering Cryptographic Protocols group of the Center for Research in Security and Privacy at the Computer Science department of Technical University Darmstadt for organizing the seminar on Privacy-Preserving Technologies which sparkled my interest in this fascinating field of research and gave me the necessary inspiration to write this thesis.

I also place on record, my sense of gratitude to my friends, including the great people I met during my studies, for the stimulating discussions and the wonderful times we had together. Without this ongoing backing and support, this thesis would not have been possible.

Finally, I would like to express my very profound gratitude to my whole family for the unceasing encouragement, support and attention throughout my years of study and the time I spent writing this thesis.

Contents

1 Introduction

1.1 Motivation

Communication has never been more important for our society than it is today. Large parts of our logistic infrastructure depend on digital computer networks and would collapse when online communication suddenly were not possible anymore.

With the rise of instant messengers and the high availability of mobile devices, communication shifted more and more from the offline world to online communication platforms. These conversations can be and nowadays often are secured by end-to-end encryption. But even if the content of exchanged messages cannot be read by an adversary, these digital platforms produce large amounts of metadata. Electronic mass surveillance programs strengthen the need for systems providing communication channels without leaking metadata which has shown to be of tremendous value [Lan15; MMM16]. E.g. for people living in suppressive regimes just getting in contact with government-critical organizations can be a huge risk. The right to remain anonymous is a fundamental right in our modern democracy.

Privacy of correspondence is one of the most important principles of our analog postal system. By design, it is difficult if not practically impossible to hide the connection between a sender and a receiver in our physical world. In contrast, digital communication networks offer more advanced techniques to ensure anonymity. Developing such systems that scale to a large number of users is a very promising research field. Private information retrieval (PIR) can be used as a building block for those solutions and also offers many possibilities for further applications, e.g. private querying of articles from an online encyclopedia such as Wikipedia[1], enabling location privacy for services like OpenStreetMap[2] or improving the scalability of Tor, as proposed in PIR-Tor [MOT+11].

Moreover, building blocks of private and untraceable communication services could potentially also be reused in other privacy-critical applications that are currently under active research, such as electronic voting systems [BV14; Neu16] or privacy-preserving location-based services [MCA06; HCE11; DSZ14].

[1] `https://www.wikipedia.org/`
[2] `https://www.openstreetmap.org/`

1.2 Contributions

We implemented two extensive optimizations in the field of Private Information Retrieval PIR that lead to significant significant performance improvements. Our work is based on the existing RAID-PIR library [DHS14] by Demmler, Herzberg and Schneider. The first enhancement focuses on reducing the computation time of the servers required to build responses for PIR queries while the second one targets the process of querying multiple database entries within one PIR query by adjusting the database layout to increase the number of entries that can be queried in parallel.

Based on the efficient implementation of PIR, we propose OnionPIR – a novel anonymous communication network that combines PIR with onion routing to create a scalable way to privately distribute messages. OnionPIR was designed to be ready for widespread deployment and provides a way to establish a secure communication channel without the need of exchanging any type of cryptographic keys out-of-band. A proof-of-concept implementation has been created to demonstrate the practicability of the system.

1.3 Outline

In §2 the fundamental concepts of Private Information Retrieval are summarized. Server-side computation requirements are lowered in §2.1 and the throughput of database entries per queries is increased in §2.2. An experimental evaluation is given in §2.3. Next, an overview of existing private communication networks is given in §3.2. A highlevel overview of OnionPIR is given in §3.3 and explicated in §3.4. Security considerations are explained in §3.5. An analysis of the scalability properties of OnionPIR is given in §3.6.1. Correctness is defended in §3.6.2. Chapter §4 concludes this thesis and gives an outlook to future work.

2 Performance improvements in RAID-PIR

Private information retrieval (PIR) is a very active research topic in the field of Privacy Enhancing Technologies (PETs). PIR can be used to receive information from one or multiple servers without disclosing which information were requested. Since this technique is a fundamental building block for higher-level protocols and applications, its performance is of prime importance. In this chapter, the performance of the RAID-PIR library is further improved.

Introduction

RAID-PIR, introduced in [DHS14] by Demmler, Herzberg and Schneider, is an information-theoretic PIR (IT-PIR) scheme based on the original design by Chor et al. [CKGS95]. Information-theoretic PIR schemes are secure even if the adversary has unlimited computation power. They typically rely on a non-collusion assumption between multiple servers.

The first computational PIR scheme was introduced by Chor, Gilboa and Naor in [CGN97] and was soon followed by another approach by Kushilevitz and Ostrovsky [KO97]. Computational PIR (cPIR) schemes are an alternative approach to private querying of data and are usually based on partially homomorphic encryption to provide strong security guarantees in a single-server setup. Their security is based on the limited computational power of the adversary. A recent realization using lattice-based cryptography is presented and evaluated by Aguilar-Melchor, Barrier, Fousse and Killijian in [ABFK16].

Both approaches, IT-PIR and cPIR, are combined by Devet and Goldberg in [DG14]. This design was called *Hybrid PIR*. Single-server cPIR protocols can be applied recursively by selecting a chunk of the database in one step of the recursion, storing an encrypted version of the selected chunk at the PIR server and using this chunk as the new database in the next step. By making use of the partially homomorphic encryption, this can be done without revealing which chunks were selected. In Hybrid PIR, the first step of the recursion is done using the IT-PIR protocol and all subsequent steps are performed via cPIR. Though this design weakens the security of the protocol by relying on the security assumptions of both IT-PIR and cPIR, partial security is preserved in case one of the two assumptions is broken.

In the original approach by Chor et al. [CKGS95], a multi-server setup together with a non-collusion assumption of the servers offers great performance and strong security-guarantees

by making use of fast XOR operations instead of homomorphic encryption. The database containing the data that may be queried by clients is thereto divided into B blocks of b bits size and distributed to k servers. When a client wants to query the n-th block from the database, it generates $k-1$ random bitstrings, called queries, of length b and constructs the k-th bitstring in a way that the XOR of all k queries results in a bitstring where all bits except the n-th are zero. Each of the k queries, as depicted in Figure 2.1, is then sent to a different server which will XOR all blocks whose corresponding bit i is set in the query and return the resulting block of length b bit. When all received blocks are then XORed together, the n-th block is recovered because all blocks of the database except the n-th block were involved an even number of times in the server-side XOR operations.

q_1	11010	01010	11101	01001
q_2	10110	01101	10101	00111
q_3	01100	10001	01110	10110
q_4	00100	10110	00110	11000
$\oplus$				
e_3	00100	00000	00000	00000

Figure 2.1: Querying data in RAID-PIR. The four queries q_i that are sent to the $k = 4$ servers consist of one (orange) *flip chunk* and three (black) randomly generated chunks so that the XOR of all queries is equivalent to the plain text query e_3.

RAID-PIR improves this scheme by splitting each query into k chunks. The queries are then constructed in a way that all chunks sent to the l-th server except the l-th chunk are generated using a pseudo random generator PRG. In addition, a redundancy parameter r is introduced to reduce the downstream bandwidth and server-side computational costs. For that matter, each server will only handle r chunks of the database. This also reduces the number of servers that have to collude in order to retrieve the plain text query of the client. Instead of k servers, r colluding servers are now sufficient to break the non-collusion assumption. Another optimization provides the ability to request multiple blocks of the database within a single query. This is achieved by XORing the requested blocks per chunk at the server side and returning one block per chunk instead of one block per query. However, this optimization has the limitation that the requested blocks have to be in different chunks of the database.

In this chapter, two additional improvements to those presented in [DHS14] are introduced to speedup RAID-PIR even further. The first one improves the generation of the PIR responses at the server-side and the second one achieves a significant speedup by using an advanced database layout where data entries are uniformly distributed among the whole data storage.

2.1 Method of Four Russians

The Method of Four Russians [ADKF70], also known as the Kronrod's method, is a technique for efficient multiplication of matrices with a limited number of possible values per cell. Since RAID-PIR makes heavy usage of such kinds of multiplications, the Method of Four Russians can be used to further reduce the computation time of the RAID-PIR mirrors required while generating the response for a PIR query. This improvement results in lower latency and higher throughput of the system and comes at the (low) cost of a new preprocessing phase for the mirrors and slightly increased memory requirements as shown in §2.3. A related approach for information-theoretic PIR schemes based on secret sharing is presented in [Hen16].

The Method of Four Russians bases on the assumption that the number of possible values for the cells of a matrix is finite. Here, it is assumed that all matrices in the multiplication $\mathcal{A} * \mathcal{B} = \mathcal{C}$ are done over the field with two elements ($\mathbb{F}_2$), i.e. only binary matrices are being multiplied. Note, that this limitation is not needed for applicability of the algorithm (e.g. its correctness property would also be fulfilled for $\mathbb{F}_{10}$) but it reduces the complexity of the algorithm and is sufficient for our use case. In $\mathbb{F}_2$, the multiplicative operation is *AND* and the additive operation is *XOR*.

The first interesting phenomenon, that is crucial to understand the idea behind the Four Russians algorithm, is that the indices of the non-zero elements of a row r in the first matrix $\mathcal{A}$ indicate a subset of rows in the second matrix $\mathcal{B}$ that have to be XORed in order to produce the r-th row of the output matrix $\mathcal{C}$. As a naïve approach, it would now be possible to precompute all possible XOR combinations of the rows of $\mathcal{B}$ and hereby create a lookup table that returns the final results for the rows of $\mathcal{C}$ that correspond to the rows of $\mathcal{A}$. In other words, a lookup could be performed for each row in $\mathcal{A}$ returning row r in $\mathcal{C}$ that would also be computed by building the XOR of the rows in $\mathcal{B}$ indicated by row r in $\mathcal{A}$. This would reduce the computational complexity from $O(n^3)$ to $O(n)$ assuming the lookup is done in constant time and all matrices are of size $n \times n$. Unfortunately, this naïve approach is inapplicable in practice due to the computational complexity for the precomputation ($O(2^n \cdot n)$). For each of the 2^n rows in the matrix $\mathcal{B}$, n one-bit XOR operations have to be performed. Thus, for $n > 4$ the traditional matrix multiplication ($O(n^3)$) would be faster than this approach.

The reason why only n one-bit XOR operations have to be performed for each of the 2^n possible combinations of rows of the matrix $\mathcal{B}$ is, that it is possible to use a Gray code to reorder the possible combinations in a way that two consecutive combinations only differ by one row of $\mathcal{B}$. For each of the 2^n combinations, it is now sufficient to XOR the result of the last precomputed result with the row of $\mathcal{B}$ that changed, according to the Gray code. The Gray code g of a binary number m can be calculated as $g = (m \oplus (m \gg 1))$. A sample lookup table illustrating the efficient calculation of the lookup table entries is given in Table 2.1.

$$\mathcal{B}^* = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Gray code	decimal	result
0000	0	$0000 = \mathcal{B}^*[0,:]$
0001	1	$1011 = 0000 \oplus \mathcal{B}^*[3,:]$
0011	3	$0010 = 1011 \oplus \mathcal{B}^*[2,:]$
0010	2	$1001 = 0010 \oplus \mathcal{B}^*[3,:]$
0110	6	$0101 = 1001 \oplus \mathcal{B}^*[1,:]$
0111	7	$1110 = 0101 \oplus \mathcal{B}^*[3,:]$
0101	5	$0111 = 1110 \oplus \mathcal{B}^*[2,:]$
0100	4	$1100 = 0111 \oplus \mathcal{B}^*[3,:]$
1100	12	$1011 = 1100 \oplus \mathcal{B}^*[0,:]$
...	...	...

Table 2.1: Example lookup table for the sample matrix $\mathcal{B}^*$

Besides these huge computational costs, there also are tremendous memory requirements to store the lookup table (2^B times the memory needed to store one row of $\mathcal{B}$, where B is the number of rows of $\mathcal{B}$). Obviously, this naïve approach is not practicable.

Instead, the matrix $\mathcal{B}$ is divided into groups of t rows each. The precomputation is then done within these B/t groups using the Gray code as described above resulting in a computational complexity of $O(2^t \cdot n^2/t)$. When t is chosen to be $t = log_2(n)$ the computational complexity simplifies to $O(n^3/log_2(n))$ resulting in a speedup of $log_2(n)$ compared to the traditional matrix multiplication.

While the matrix $\mathcal{B}$ is divided into groups horizontally, the matrix $\mathcal{A}$ has to be divided into vertical groups of t columns. To multiply the two $n \times n$ matrices, $\mathcal{A}$ is now traversed column-wise - grouped by t columns forming a group each. For each group, the corresponding lookup table can now be created by t rows of $\mathcal{B}$ and a lookup can be performed for all t-bit subparts of the n rows of $\mathcal{A}$ in this group. This procedure is depicted in Figure 2.2. Note, that the lookup tables are no longer of any use when the corresponding groups were processed making it possible to reuse the allocated memory for the next lookup table.

Implementation

An efficient multiplication of the Method of Four Russians is provided by Albrecht, Bard and Hart in [ABH10]. However, since the existing implementations only implement full matrix-matrix multiplications and do not suit the data structures provided by RAID-PIR, none

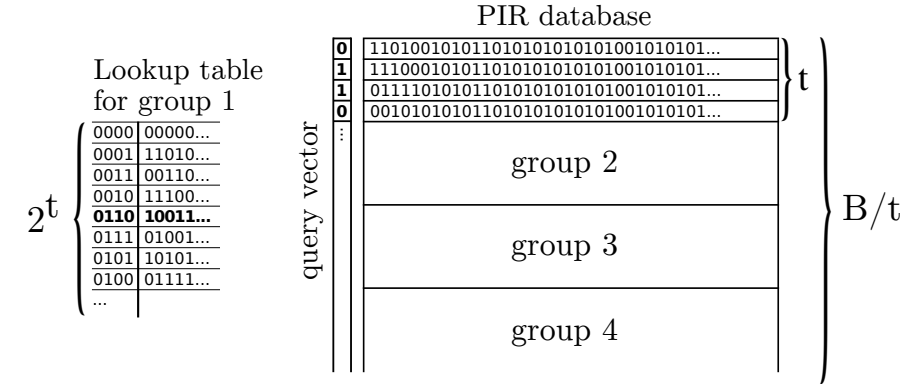

Figure 2.2: Precomputation in the PIR database. A query vector is one *row* in the matrix $\mathcal{A}$. The first four bits of the query vector represent the index in the lookup table for the first group of $\mathcal{B}$ (the PIR database).

of these were used. Instead, the idea behind the Four Russians multiplication was adopted to create a new implementation directly integrated into RAID-PIR to avoid costly memory movements and exploit memory locality.

While the traditional Method of Four Russians assumes two full matrices to be multiplied, this is not the case in RAID-PIR. Often, only a single PIR request has to be answered. For this vector-matrix-multiplication, the improvements introduced by doing the precomputation do not apply. Caching several PIR requests would solve this issue. However, the amount of cached requests would have to be in the order of magnitude of the number of entries of the matrix $\mathcal{B}$, i.e. our PIR database. Therefore, a slightly adapted version of the original Four Russians multiplication that fits well together with the requirements of the RAID-PIR library was implemented.

Instead of caching the PIR requests, the lookup tables for all groups are being cached. Even for a single PIR request, these lookup tables can now be used to gain a theoretical speedup of t while building the response vector. This exactly matches our scenario and is possible due to the fact that $\mathcal{B}$ can be preprocessed in the absence of any PIR request.

The downside of holding all lookup tables in memory are the increased memory requirements. While a larger t decreases the computation time for the generation of the response vectors, it also excessively increases the memory needed to store the lookup tables. When t of B entries of the PIR database are clustered in a group, only B/t XOR operations have to be performed per query resulting in a speedup of t. On the other side, each of the B/t lookup tables requires $2^t \times b$ bits of memory where b is the length of a database entry. A comparison of the theoretical speedup compared to the corresponding memory requirements is given in Table 2.2. The choice of $t = 4$ gives a good trade-off between the computational benefits and memory costs for larger databases.

t (speedup)	$\frac{1}{t} \cdot 2^t$ (memory requirements)
2	2.00
3	2.66
4	4.00
5	6.40
6	10.67
7	18.29
8	32.00

Table 2.2: Comparison of speedup and memory requirements for the adapted version of the Method of Four Russians

The optimizations were implemented in C and integrated into the existing RAID-PIR library. Since the PIR clients do not have to be aware of the changes to the calculation of the XOR responses, only the mirror servers of the implementation were affected by this optional precomputation. Note that two different server operators running a mirror server could decide independently whether they enable the precomputation or not.

In [LG15] another approach to speedup server computation time based on Strassen's algorithm for matrix multiplication was introduced by Lueks and Goldberg. However, this approach is only efficient if the number of PIR requests is a power of 2 and does not really fit into the RAID-PIR approach where the block size is typically relatively large and queries are mainly processed iteratively.

2.2 Uniform Distribution of the Data Entries

In the current design of RAID-PIR, the files to distribute via PIR are simply concatenated to build the PIR database. Thus, a larger file will be placed in consecutive database blocks. While this does not imply the performance of a query when retrieving the blocks consecutively, querying multiple blocks in parallel is often not possible.

RAID-PIR introduced so called "multi-block queries" (MB) enabling a client to request more than one PIR block in a single query to the mirror servers. The improvements of RAID-PIR compared to the original CKGS scheme [CKGS95] base on the fact that the PIR blocks are grouped into k chunks that divide the database into equally sized parts. Mirror servers receiving a multi-block query will reply with one block per query chunk. Within these chunks the XOR of the corresponding blocks is calculated as before. Since only one block per chunk can be queried in one multi-block query, the blocks to retrieve have to be located in different chunks. The performance analysis of RAID-PIR proved that no speedup could be achieved for a large consecutive file residing in a single chunk. However, a large speedup could be measured when querying 10 small files that were uniformly distributed among the database.

Implementation

As shown in [DHS14], the improvements introduced by the multi-block queries only apply for multiple files located in different chunks of the database. To improve the performance for multi-block queries for a single large file, the layout of the database has to be improved. Instead of placing files in consecutive blocks, they are now uniformly distributed within the whole database, as shown in Figure 2.3. For each file, the number of blocks n needed to store its contents is calculated and the file is then placed in a way that between two blocks that are assigned to the file $B/n-1$ blocks are used for other files, where B is the number of blocks in the database. If a chosen block is already assigned to another file, the next free block of the database is used instead. The first bytes of the next file will then fill the rest of the previous file's last block. Furthermore, it is guaranteed that all B blocks of the database contain data and no block except the last one is only partially filled.

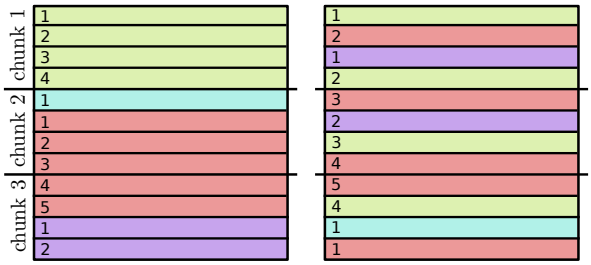

Figure 2.3: Uniform distribution of the data entries in the PIR database. While the first file (green) filled up only the first chunk in the original database layout (left), it is now uniformly distributed over the whole database (right).

These optimizations were integrated in the existing RAID-PIR library. Since both the mirror servers and the clients have to be aware of the new locations of the database entries, both client-side and server-side code was extended to support the new database layout.

2.3 Benchmark Results

In this section, the improvements introduced by the Method of Four Russians and the uniform distribution of the data entries in the database are presented and further elaborated.

The benchmarks are based on the implementation of RAID-PIR [DHS14] that was modified to support the improvements described in §2.1 and §2.2. In the test scenario, three mirror servers were deployed as `r3.xlarge` instances on Amazon EC2 with 30.5 GiB main memory, an Intel Xeon E5-2670 v2 processor and a 1 Gbit/s ethernet connection. The PIR queries were performed by a `t2.micro` instance with 1 GiB main memory, one core of an Intel Xeon E5-2676 v3 processor and a 250 Mbit/s ethernet connection (0.5ms latency) in the WAN

setup and a notebook with 8 GiB main memory, an Intel Core i3-3120M processor and a consumer grade internet connection (4.5 Mbit/s downstream, 400 Kbit/s upstream, 30ms latency) in the DSL setup. The database used in both scenarios consists of 964 files of Ubuntu security updates adding up to a total size of 3.8 GB. The average file size is 4 MB and the median file size is only 267 kB due to a large number of small patches. All experiments were run five times and the average runtimes are given in the figures below.

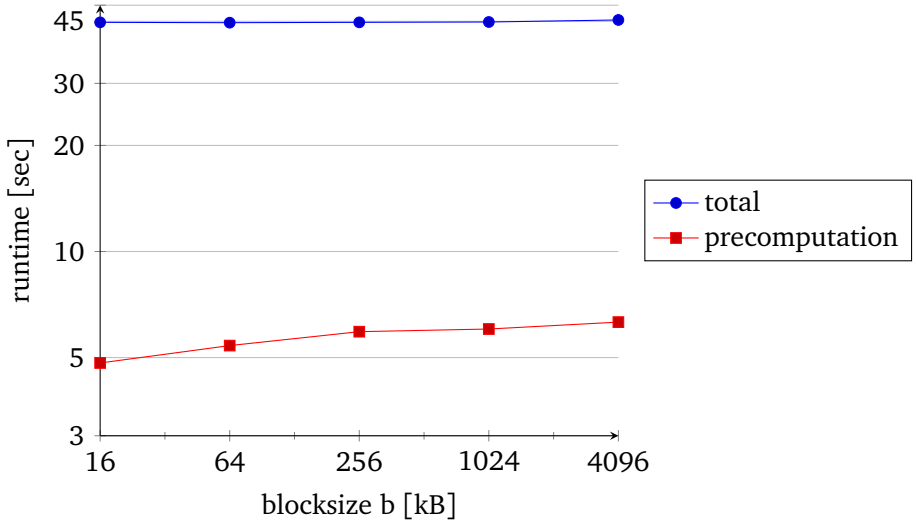

Figure 2.4: Startup duration for the PIR mirrors

Figure 2.4 shows the mirror startup duration of the PIR mirrors. This time includes the time needed to read the database from disk, store it into main memory and precompute the lookup tables introduced by the Method of Four Russians. The figure shows that the startup time does not heavily depend on the block size and the precomputation time is negligible even for larger databases. For a group size of $t = 4$, only $4 \cdot B$ XOR operations have to be performed during the precomputation phase where B is the total number of blocks in the database. Note, that in contrast to the generation of a PIR responses, data has to be written to main memory while the lookup tables are generated. This results in a much slower precomputation than the generation of 4 single-block responses – even if the same amount of XOR operations have to be performed.

Next, the number of servers is set to $k = 3$ and the redundancy parameter is set to $r = 2$. The block size is varied from $b = 16$ kB to $b = 4$ MB depicted on the x-axis and the total runtime is depicted on the y-axis. In Figure 2.5, one large 8.5 MB file is requested using various PIR schemes in the WAN setup. Single-block queries are abbreviated as `SB`, multi-block queries as `MB`, the Method of Four Russians as `4R` and the uniform distribution of the data entries as `UD`. The graphs for the single-block uniform distribution and the single-block uniform distribution with precomputation are identical to their respective versions not including the uniform distribution and were therefore omitted. This is not surprising since the uniform distribution was introduced to speedup the performance of multi-block queries and has no

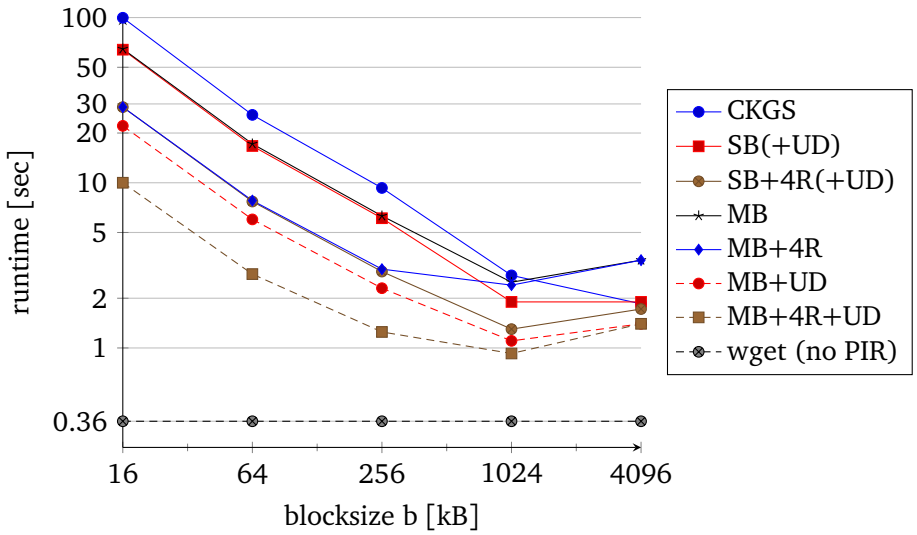

Figure 2.5: Runtimes for varying block sizes b with $k = 3$ servers and redundancy parameter $r = 2$ and one large file (8.5 MB, WAN setup). `CKGS`: original PIR scheme [CKGS95], `SB`: Single-Block scheme, `MB`: Multi-Block scheme, `4R`: Four Russians Precomputation, `UD`: Uniform distribution of the data entries.

impact in multiple consecutive queries where the location of the requested block does not matter.

As already observed in [DHS14], single-block and multi-block queries take roughly the same time when querying one large file in the WAN setup, where bandwidth and latency typically are not the bottleneck. For large block sizes in the non-uniformly distributed database, single-block queries profit from the lower amount of irrelevant data processed by the mirrors and transferred to the client compared to the multi-block queries. The Four Russians precomputation leads to a significant speedup and effectively halves the runtime of most test cases. This is a good result even if the theoretical speedup, that does not cover data locality issues, is 4. When the cache size of the CPU is not sufficient to store the precomputed lookup tables and all interim results for large block sizes, the Method of Four Russians' speedup decreases and is not measurable for $b = 4$ MB any more.

The speedup gained by uniformly distributing the data entries in the database is even greater and improves the overall runtime by a factor of 3 compared to the original multi-block queries for small block sizes. While all blocks have to be queried from the first chunk for a non-uniformly distributed database, it is now possible to retrieve 3 blocks from 3 different chunks in one multi-block query in parallel. When both optimizations are combined, the runtime decreases from 64 seconds (or 100 seconds for the original CKGS scheme) to 10 seconds for a block size of $b = 16$ kB and reaches its minimum for $b = 1$ MB where the runtime decreases from 1.9 seconds for the originally best performing single-block queries to 0.9 seconds for multi-block queries using both optimizations.

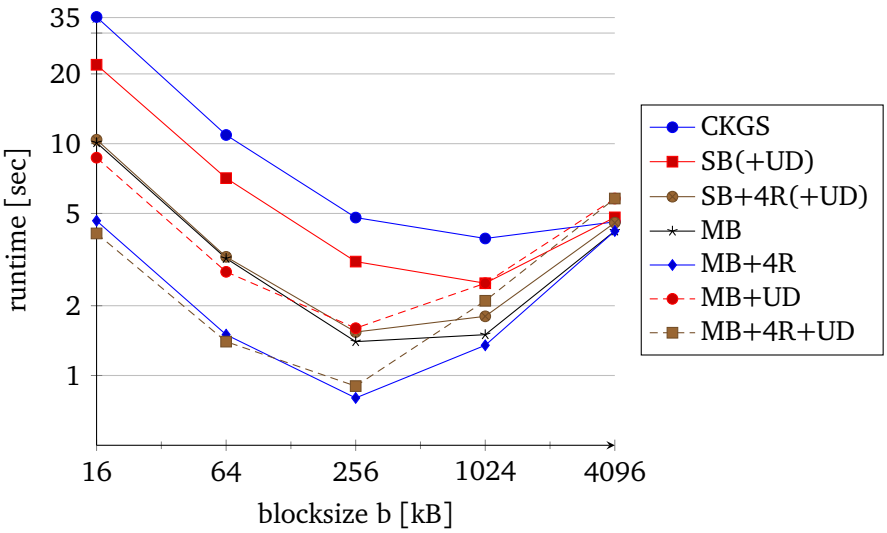

Figure 2.6: Runtimes for varying block sizes b with $k = 3$ servers and redundancy parameter $r = 2$ and 10 small files (2.9 MB, WAN setup). `CKGS`: original PIR scheme [CKGS95], `SB`: Single-Block scheme, `MB`: Multi-Block scheme, `4R`: Four Russians Precomputation, `UD`: Uniform distribution of the data entries.

The results for querying 10 small files adding up to 2.9 MB are depicted in Figure 2.6. Here, the multi-block queries are significantly faster than the single-block queries even when no optimizations are applied since the files being requested are already distributed among the whole database. Therefore, uniform distribution of the data entries does not have a significant impact on the overall performance. Again, the Four Russians precomputation halves the runtimes of the test cases and does not introduce improvements any more when the cache size of the processor is not sufficient for larger block sizes. For a block size of $b = 16$ kB, the runtime decreases from 10.1 seconds for the originally best performing multi-block queries to 4.1 seconds when all optimizations are applied. The best performance is obtained for a block size of $b = 256$ kB where 7 queries are performed to retrieve 18 blocks in 0.8 seconds.

In Figure 2.7, the results for querying the one large file (8.5 MB as in the first test case) over the consumer-grade DSL connection are depicted. The first interesting observation is that the results for single- and multi-block queries with a block size of $b = 16$ kB do not differ significantly. The reason for this is the low upstream bandwidth of the DSL connection. For small block sizes, the upstream bandwidth, which depends on the number of blocks, has a larger impact than the downstream bandwidth, which depends on the block size. To perform a query to a database containing 246 360 blocks of 16 kB each, 246 360/3 bit have to be transferred to each server (plus additional 16 Bytes for the PRG seeds) resulting in a total transmission time of $3 \cdot (246\,360/3 \text{ bit} + 16 \text{ Byte})/(400 \text{ kbit/s}) \approx 602$ ms. This phenomenon can also be observed in the benchmarks run in [DHS14].

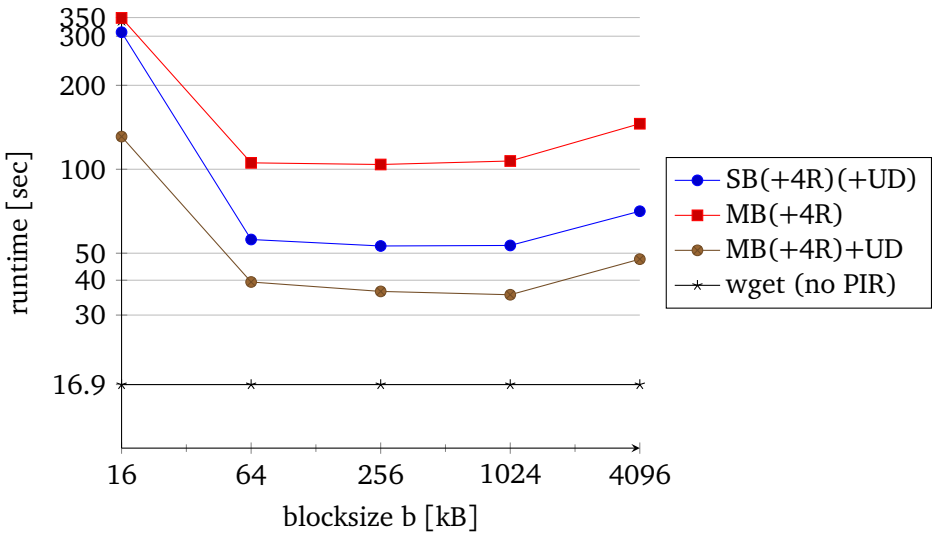

Figure 2.7: Runtimes for varying block sizes b with $k = 3$ servers and redundancy parameter $r = 2$ and one large file (8.5 MB, DSL setup). `CKGS`: original PIR scheme [CKGS95], `SB`: Single-Block scheme, `MB`: Multi-Block scheme, `4R`: Four Russians Precomputation, `UD`: Uniform distribution of the data entries.

Due to this high latency and communication overhead of the queries, the Four Russians precomputation has a significant impact in this scenario. As in the first test case, without the uniform distribution, single-block queries provide better performance than multi-block queries. However, when the data entries are distributed uniformly, a significant speedup to multi-block queries is introduced and – as already expected – the multi-block queries supersede the single-block approach because the number of requests reduces by about a third. It was also shown that larger block sizes have the disadvantage that large parts of some requested blocks contain no relevant data and therefore lead to a slower runtime.

For a redundancy parameter of $r = 2$, the data that needs to be sent to the client is about twice the size of the raw data. The best results are achieved for a block size of 1 MB in the uniformly distributed database using the multi-block queries. Retrieving the file without any PIR mechanisms[1] takes 16.9 seconds and is only faster by a factor of 2.10, indicating that the runtime of 35.5 seconds for the private information retrieval approach almost reached the theoretically optimal results for the DSL setup.

[1] using `wget` over an unencrypted HTTP connection

3 OnionPIR

The improvements in RAID-PIR automatically result in a better performance of all systems using this PIR library as a building block. In this chapter, a private communication system is introduced that makes use of private information retrieval to bootstrap anonymous communication.

3.1 Motivation

When two parties register under pseudonymous identities and connect to each other simply by using Tor [DMS04], a malicious server can link those two identities together. Even if these pseudonyms do not reveal any information about the users behind them, it is possible to build a social graph isomorphic to the one built with information from other sources. These two graphs can then be mapped together and thus reveal information about the users behind the pseudonyms. Therefore, more advanced techniques are needed to provide protection against leaking of metadata. Additionally, it has turned out that the illusion of users willing to exchange a shared secret is not realistic. In order to provide protection against mass surveillance, a system to establish private communication channels based on already existing contact information such as phone numbers or email addresses is needed.

3.2 Related Work

In our daily communication, end-to-end encryption is available and ready for widespread deployment. In the past, these technologies were not accessible to a large number of users because they often required knowledge of public key cryptography or were just not as easy-to-use as other messaging applications. End-to-end encryption was only used by computer specialists and a relatively small group of people who really saw a need to protect their communication. For example, users of OpenPGP[1] have to manually build a web-of-trust and should be familiar with the concept of public key cryptography and S/MIME[2] requires certificate management. When the Signal Protocol got integrated into popular messaging

[1] `https://tools.ietf.org/html/rfc4880`
[2] `https://tools.ietf.org/html/rfc2633`

services like WhatsApp[3] or Facebook Messenger[4] earlier this year, private messaging suddenly became available to a extremely large user base.

However, creating platforms that protect not only the content of the communication but also its metadata is still under active research. A couple of concurring techniques to build communication networks that avoid to leak metadata have been presented in the last years and will be compared in this section. Private Set Intersection [Mea86; PSZ14; PSSZ15], Private Function Evaluation [Yao82; KS08; KS16] and Oblivious RAM [WHC+14; LWN+15] are similar fields of research that could possibly also be conducive building blocks of anonymous communication networks in future research.

Not all protocols summarized in this section provide confidentiality. However, since this is an already solved problem, confidentiality could be added on top of all of the presented protocols requiring only some minor changes.

3.2.1 Redphone

One of the first applications that used privacy preserving technology to prevent leaking of metadata was Redphone by Open Whisper Systems[5]. In Redphone, bloom filters [Blo70] were used for private contact discovery. However, Redphone did not include any mechanisms to protect metadata when an actual call was initiated.

When the encrypted voice call feature Redphone provided got integrated into the more popular messaging app Textsecure (which was later renamed to *Signal*), the authors claimed that bloom filters would not have been able to handle the larger user base and had to be removed[6].

3.2.2 DP5

The Dagstuhl Privacy Preserving Presence Protocol[7] (DP5) described in [BDG14] by Borisov, Danezis and Goldberg provides a mechanism to exchange online presence information in a privacy preserving way by making use of PIR schemes. The goal of the protocol is to distribute these status information while the server is not able to read the associated data, learn who is online or link two participants of the protocol.

In DP5, the concept of epochs is used to describe successive time slots in which a client A can register their presence status at the server. In the subsequent epoch, another user is then able to query the presence information stored in the previous epoch privately using a PIR mechanism. The stored data is encrypted using Authenticated Encryption with Associated

[3] `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`
[4] `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`
[5] `https://whispersystems.org/`
[6] `https://whispersystems.org/blog/contact-discovery/`
[7] According to the authors, the extra 'P' is for extra privacy.

Data (AEAD) with a key K_j and stored under an identifier ID_j for the current epoch T_j. The key K_j and the identifier ID_j are both derived from a cryptographically strong symmetric key K_{AB} shared between the users A and B of the service. This guarantees that the server is not able to read the data or associate it with a specific user. The presence status can then be queried in the epoch T_{j+1} using private information retrieval by the user B who is able to derive the identifier from the pre-shared key K_{AB} and decrypt it using the shared key K_j also derived from K_{AB_j}.

However, this simple approach does not scale for many users because each user has to store its presence status n_f times, where n_f is the number of friends of the user. Therefore, instead of using only one epoch, a long-term epoch and a short-term epoch are used. The former one works as described above, but instead of storing the presence status directly, a single public key pk_{A_j} of the user A is stored and received by all n_f friends of A individually. This public key is then used in the short-term epoch t_j to derive an identifier for the record of A and query the presence information encrypted by a key also derived from pk_{A_j}. Since the public key pk_{A_j} of A is still distributed in an encrypted way in the long-term epoch, only the friends of A that received the public key are able to decrypt the presence status in the short-term epoch. In contrast to the simple approach, each user has to store its presence information only once (instead of n_f times) in each short-term epoch t_j. This way the general performance in terms of scalability of the protocol can be drastically improved.

As described in [BDG14], the updates in the long-term epoch are done in the order of a day while the updates in the short-term epoch are done in the order of magnitude of minutes. This makes DP5 a good choice for exchanging presence information, but also makes it unusable for messaging or real-time communication like voice or video calls. However, it is possible to use the information exchanged via DP5 to build more complex secure protocols on top of it. A big drawback of DP5 is that it assumes a cryptographically strong symmetric secrets to be shared between two communication partners as a prerequisite for the protocol.

3.2.3 Alpenhorn

Alpenhorn [LZ16] is a recent approach by Lazar and Zeldovich to bootstrap secure communication without leaking metadata. It was first introduced at the Privacy Enhancing Technologies Symposium 2016 and uses identity-based encryption (IBE) [BF01] and a mix network [CJK+16; Cha81] instead of private information retrieval to provide strong privacy guarantees.

The idea of identity-based encryption was first proposed by Adi Shamir in 1984 [Sha84] but remained an unsolved problem until 2001 when the pairing-based Boneh-Franklin scheme [BF01], which is used in Alpenhorn, and Cocks's encryption scheme [Coc01] based on quadratic residues were introduced. The idea of IBE is to distribute key pairs of a public-key encryption scheme in a way that it is possible to derive the public key of a mathematical function over some publicly known parameters and a unique identifier (such as an email

address) of the user. The secret key can then be obtained from a central authority which first requires the user to prove the ownership of the email address.

Obviously this concept relying on a central authority that is able to generate the secret keys for all users is undesirable. Alpenhorn solves this issue by using the exchanged keys only to exchange a shared secret between two participants of the protocol. Additionally, an anytrust thread model is used to ensure that only a single of multiple IBE servers has to be honest to protect the privacy of the metadata. Nevertheless, the design of the system does not provide a method to detect malicious actions by the servers – a server could give out the private key of any user to a third party without the users ever getting aware of this action.

Compared to a key exchange mechanism using a traditional public key infrastructure (PKI), this allows users to retrieve public keys in a secure way while no prior out-of-band communication is required at all.

When a user A wants to initiate a private communication channel with another user B of the Alpenhorn system, she first encrypts a secret with the identifier of B as a public key. This encrypted secret is then routed through a mix network to a mailbox identified by the username of B. The anytrust model used for the IBE servers is also used for the mixnet servers. When the user B retrieves the encrypted secret from the mailbox, he is able to decrypt it using the private key obtained by the central IBE authority. The secret then becomes a shared secret and is stored in the address books of both A and B.

To ensure forward secrecy, a new *keywheel* construct is introduced. This *keywheel* construct continuously evolves the pairwise shared secrets and also ensures that, at any given point in time, the same secrets are available on both end systems. This guarantees that an adversary who later compromises an end system still is not able to learn anything about communication channels initiated in the past.

Alpenhorn was integrated in the Vuvuzela private messaging system introduced in [HLZZ15] which aims to provide a scalable way to communicate privately without leaking any metadata. It was shown that Alpenhorn supports 10 million users using three Alpenhorn servers with an average dial latency of 150 seconds and a client bandwidth overhead of 3.7 KB/s.

3.2.4 Riposte

Riposte [CBM15] is an anonymous messaging system by Corrigan-Gibbs, Boneh and Mazières designed to handle a large number of users privately posting public messages to a public bulletin board. Its privacy guarantees include that it is not possible for an adversary to determine who posted which message making the system a perfect choice for latency-tolerant anonymous surveys or anonymous microblogging services where the number of readers is considerably smaller than the number of writers. By combining the presented techniques with public key encryption, Riposte could be used to build point-to-point private messaging channels.

In Riposte, time is divided into epochs and a database is used to store all messages posted by the users. This database is divided into a fixed number of rows and gets initialized with zeros at the beginning of each epoch by each of the n servers. When one user of a fixed set of users, called anonymity set, wants to privately write a message to the l-th row of the database, it first splits its write request into n shares by using a distributed point function (DPF) [GI14]. The shares are created in a way that the XOR of all shares results in a vector where the l-th entry contains the message to be stored in the database and all other entries are zero. However, due to the characteristics of the distributed point function it is not possible to gain any information from any proper subset of the n shares. The security of the protocol is based on a non-collusion assumption of the n servers. Each share is sent to a different server which will then add the share to its local database by using the XOR operation.

When all servers agree that the epoch has ended, they XOR their local databases with the ones stored at all other servers. The resulting database will then contain all messages that have been submitted in the corresponding epoch in cleartext.

The decision when an epoch has ended is important for the privacy guarantees of the system because the anonymity set of a user is only as large as the number of users who submitted their messages in the ongoing epoch. The end of an epoch can be determined by a fixed time in the order of hours, a fixed number of users who sent their shares to the servers or by any other more complex scheme that fulfills the privacy requirements of the system. When an epoch has ended and the cleartext database was published, all servers initiate a new epoch by clearing their local databases. The authors also describe a mechanism to exclude malformed client requests and handle two-way collisions.

By using distributed point functions, Riposte provides some kind of “reverse PIR” protocol that offers decent performance for anonymous broadcast messaging. However, the size of the users request is proportional to the square root of the size of the whole database.

3.2.5 Pond

Pond[8] is an attempt to create a federated email-like communication service that leaks no metadata. However, there are some differences and limitations compared to the traditional mail system. The most noticeable is the absence of public identifiers for contacts. When two users want to communicate, they first have to exchange a shared secret over an already established secure channel.[9]

When the initial key exchange was successful, the clients send and receive messages via Pond servers, implemented as Tor Hidden Services. Pond tries to hide metadata about the messages that have been sent by always sending data at random points in time. If there

[8] `https://github.com/agl/pond`

[9] Pond suggests some interesting ideas to make this manual key exchange independent of often insecure mobile devices. E.g. the author suggests a shuffled deck of cards, split into halves, which are given to the two participants who want to communicate, resulting in approximately 49 bits of entropy for one deck of cards or around 100 bits for two decks.

is no message to send, random data is transmitted instead. All messages are padded to a fixed size to leak no information of the traffic profile. To prevent anyone from sending messages to a user, Boneh-Boyen-Shacham group signatures [BBS04; CH91] are used. Each contact of a user will be part of the group that is able to sign a message to this user without revealing his or her own identity. Note that this design allows a server to learn how many messages a user receives even if it is impossible to determine *who* sent a message to a specific user.

The author of Pond discontinued the project and suggests to use other services instead. However, the project contributed some interesting ideas which may prove useful in future work.

3.2.6 Ricochet

Ricochet[10] is an instant messaging service that is fully decentralized and tries to eliminate the exposure of metadata. In Ricochet, Tor hidden services are used to create private communication channels and the address of a user's hidden service is used as an identifier that has to be exchanged out-of-band to establish a connection. Since each user creates one and connects to a number of hidden services, the plaintext identifiers are leaked to the Tor Hidden Services Directories (HSDirs). Recent research has shown that HSDirs are actively used to track users inside the Tor network [SN16]. Since Ricochet is based on the Tor Hidden Service architecture, its privacy assumptions strongly depend on the security thereof.

3.2.7 Riffle

Riffle [KLDF16] claims to provide scalable low-latency and low-bandwidth communication based on mix networks [Cha81] using verifiable shuffles [BG12; FS01; Nef01] to provide sender anonymity and private information retrieval to provide receiver anonymity. An anytrust model was chosen to provide the security guarantees using a multiple server setup.

In Riffle, time is divided into epochs and each client sends and receives messages even if they do not have anything to communicate. This is done to provide traffic analysis resistance. Each epoch is divided into two phases, the setup phase and a communication phase which is again split up into multiple rounds. The authors state that traditional verifiable shuffles would be unsuitable for high bandwidth communication and introduce a novel type of verifiable shuffles, called *hybrid verifiable shuffle*. Instead of sending the actual messages through the mix network, clients generate shared secrets which are then sent through the mix network to the last server S_m in the setup phase. The servers perform a verifiable shuffle and verifiable decryption of the onion-encrypted messages and send the result to the next server.

[10] `https://ricochet.im/`

Because the keys are onion-encrypted by the client, none of the other servers $S_{i,i\neq m}$ of the mix network are able to gain any information about the shared secrets. Each server S_i stores the permutations π_i used within the verifiable shuffle for later usage in the communication phase. This significantly reduces the computational cost for the verifiable shuffles and allows the servers to bootstrap verifiability from the initial shuffle of keys.

In each round of the communication phase, clients onion-encrypt their messages to be transferred and send them to the same server as in the setup phase. Splitting the communication phase into multiple rounds now reveals the advantage that the permutations π_i for each server S_i can be reused multiple times. When all messages have been collected by the last server S_m, they can now be decrypted by the keys exchanged via the verifiable shuffles in the setup phase. They are then distributed among all servers and, depending on the application, either broadcasted to all clients or selectively transmitted using private information retrieval. As described in the paper, information theoretic multi-server private information retrieval offers the opportunity to only download the messages a client is interested in and thus reduce bandwidth costs. The used PIR scheme based on the one initially introduced by Chor et al. in [CKGS95] could probably be easily replaced by RAID-PIR, introduced by Demmler, Herzberg and Schneider in [DHS14], to improve Riffles performance even further.

3.2.8 The Pynchon Gate

The Pynchon Gate is an anonymous mail system presented by Sassaman and Cohen in [SC05] that guarantees receiver but does not provide sender anonymity. The use of a mix-network is suggested to also guarantee the privacy of senders. To achieve receiver anonymity, private information retrieval is used. When a sender sends a message to the server, the message is placed in a tree structure of fixed-size buckets which can later be privately queried by its recipient. The used PIR scheme is the one introduced in [CKGS95] and could probably easily be replaced by RAID-PIR.

3.2.9 Conclusions & Comparison

In the protocols outlined in the previous sections, several building blocks have been used to create privacy-preserving communication channels. These building blocks will be summarized in this section.

One noticeable pattern is that time is often divided in epochs to assure some anonymity guarantees among a set of users or to separate different rounds of communication. This technique is often used in combination with PIR-based solutions or mix networks. The epochs used in Riposte [CBM15] have to be quite long[11] because a significant amount of clients have to send their messages in Ripostes “reverse PIR” protocol in order to assure anonymity

[11] in the order of several hours

among a significantly large group of users. A widespread approach to reduce the epoch duration is to send random traffic at the client-side. While this improves on the latency of the transmitted messages, it results in a significantly reduced available bandwidth of the whole system.

Mix networks [Cha81; CJK+16], which are another building block for privacy preserving communication protocols, typically require computationally expensive zero-knowledge proofs to be able to create verifiable shuffles and provide a protection mechanism against malicious servers. However, latest research [KLDF16] has shown that it is possible to reduce the computational cost of mix networks by introducing a hybrid verifiable shuffle and accomplish reasonable performance.

Onion-routing used in the Tor network [DMS04] is conceptually similar to the idea of mix networks but offers higher bandwidth, lower latency and therefore much better scalability. Because of the fact that onion routing does not depend on any kind of cryptographic proof by the servers and does not shuffle the messages in favor of lower latency, it is susceptible to traffic analysis attacks by exit node operators or global adversaries. Using onion-routing instead of mix networks is always a trade off between scalability and security.

Identity based encryption [Sha84; BF01; Coc01] used in Alpenhorn [LZ16] has shown to be an efficient way to privately distribute public keys that requires no interaction with a server at all. The biggest disadvantage of IBE is the concept of a central server generating and distributing private keys. By suggesting an anytrust model for the IBE servers, Alpenhorn manages to provide a solution for this disadvantage, making IBE a very interesting concept for future research.

An overview over the privacy enhancing technologies used in the presented protocols is given in Table 3.1.

Protocol	used privacy preserving technologies
Redphone	bloom filters
DP5	computational PIR
Alpenhorn	identity-based encryption, mix network, *keywheel* construct
Riposte	“reverse PIR”, broadcast
Pond	Tor Hidden Service
Ricochet	Tor Hidden Service
Riffle	mix network (using hybrid verifiable shuffles), broadcast or PIR
The Pynchon Gate	information-theoretic PIR, mix network
Here: OnionPIR	information-theoretic PIR, Tor

Table 3.1: Privacy Enhancing Technologies used in existing privacy preserving communication protocols

3.3 System Model and Goals

Existing private communication networks do not really scale for huge numbers of users or require the exchange of a shared secret out-of-band which leads to usability issues most users are not willing to accept. Even though there are some privacy-aware people who would take the burden of exchanging a shared secret in a secure way, most people will not. The success of the popular messaging app Signal[12] and the adaption of its protocol in WhatsApp and Facebook Messenger are significantly founded on the combination of strong security and great usability. Since most users do not have any deep knowledge on cryptographic primitives, it is necessary to build technologies that provide privacy and usability by design and give reasonable defaults for these users.

Another design goal was to create the possibility to build real-time communication channels like voice calls or large file transfers. These features require a tradeoff between information-theoretic security and practical applicability. The anonymity network Tor [DMS04] is used to provide security against most types of attackers. Only if the system scales well, it is possible to support a large number of users and therefore provide protection against mass surveillance. This comes with the restriction that the system may not be sufficient for people that are under direct attack by a global adversary that is able to break the security assumptions Tor is based on.

To combine the strong information-theoretic privacy guarantees of private information retrieval protocols with the efficiency of onion routing protocols, the interaction with the server is divided into two phases, an initialization phase where the communication channels are established and a communication phase where the actual communication takes place.

In the initialization phase, PIR techniques are used to privately exchange information between two parties which want to communicate securely. This information is exchanged in a way that no one except these two parties will ever get to know that the communication between them ever happened. Private information retrieval enables the two communication partners to retrieve contact details without revealing any information about the query to the server. This procedure could theoretically be used to exchange all kinds of data. However, when it comes to practical applicability, querying huge amounts of information via PIR has shown to not scale well for a larger number of users.

Therefore, the exchanged information are then used to place messages, encrypted using Authenticated Encryption with Associated Data (AEAD), in an anonymous inbox, called “dead drop”, at the OnionPIR server in the communication phase. By using onion routing to hide the identity of the communication partners, the server is not able to determine who sent and received the message.

[12] downloaded by 1.000.000-5.000.000 users according to Google Play, `https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms`

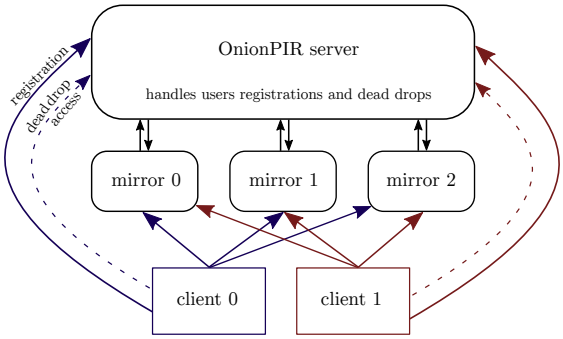

Figure 3.1: OnionPIR system model. Each client performs PIR queries via the mirror servers. The OnionPIR server handles user registrations, holds the database for the dead drops and distributes the PIR database to the mirrors. Client may connect to the OnionPIR server directly or via Tor.

3.4 Protocol Description

The OnionPIR system model, depicted in Figure 3.1, consists of the clients who want to communicate with each other and different types of servers. All honest clients together form the anonymity set among which a user is anonymous. That means that a potential adversary cannot determine which users within this set communicated with each other. The central OnionPIR server is a fundamental part of the system. It handles user registration and serves as a content provider for the PIR servers. It also acts as a database server for the anonymous "dead drops" that are used to communicate in the communication phase. The PIR servers are used in the initialization phase to privately perform the key exchange between two clients. A simplified version of the protocol is depicted in Figure 3.2.

When a client A registers for the service, it first runs through an account verification process and sends its public key to the OnionPIR server which will later distribute it to the PIR servers. Another client B, that has an address book entry for A, will later privately query for A's public key using private information retrieval. Using PIR to query the public key ensures that the servers are not able to selectively send specific public keys to a given user unless they collude. Each user periodically queries for his or her own public key to make sure the servers are not distributing bad keys (see §3.5 for details). B will then use his own private key and the received public key to generate a shared secret K_{AB} between A and B by performing an Elliptic Curve Diffie-Hellman (ECDH) key exchange. Since no communication with the server is required to derive the shared secret, this type of key agreement protocols is often also called *private key agreement*. In the same way, A is also able to derive the shared secret K_{AB} using her own private key and the public key of B.

The shared secret K_{AB} is then used to derive the identifiers for the anonymous dead drops used to exchange messages. Dead drops are always accessed via Tor to hide the user's identity. In order to hide which accesses are done by the same client, the connection to two different

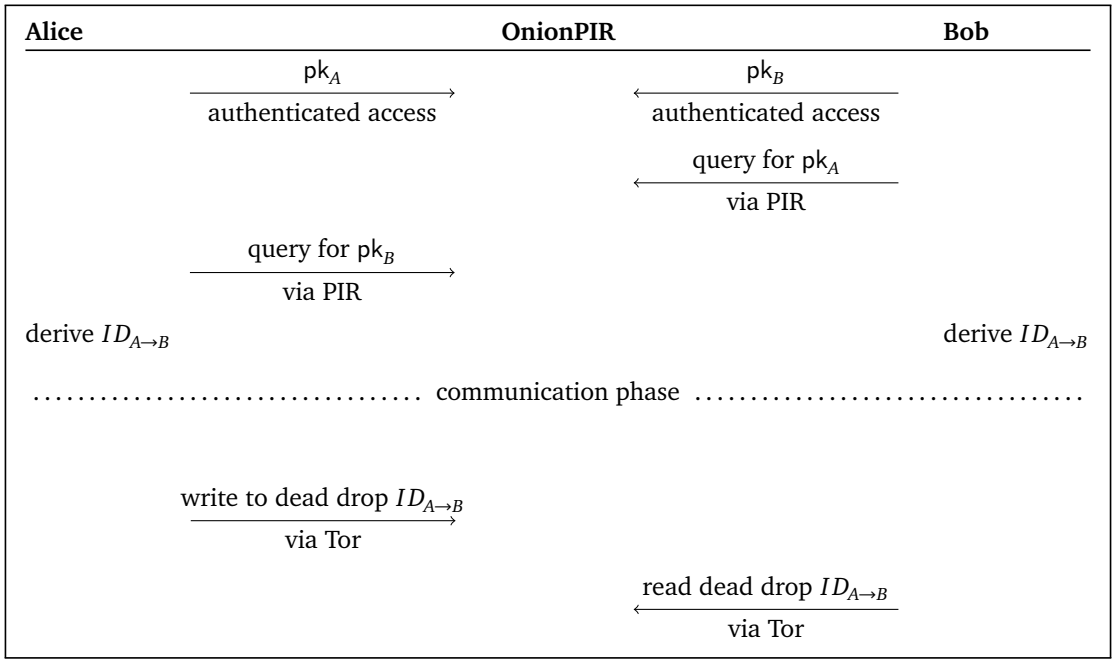

Figure 3.2: Simplified OnionPIR protocol. The registration, dead drop and PIR servers were abstracted into one server. Key renewal and the answer of Bob are not shown.

dead drops is thereby done using two different Tor circuits. The identifiers for the dead drops are derived by first concatenating the key K_{AB} with the public keys of both communication partners separately and hashing the resulting values using a cryptographic hash function. Thereafter, the client A holds two different shared secrets, $K_{A\to B} = hash(K_{AB}||pk_B)$ for sending messages to B and $K_{B\to A} = hash(K_{AB}||pk_A)$ for receiving messages from B. These secrets constantly get replaced by new ones transmitted alongside every message to provide forward secrecy. After this step, the shared secret K_{AB} is not needed anymore.

The identifiers of the dead drops could then be built by concatenating the keys $K_{A\to B}$ and $K_{B\to A}$ with a nonce N_t that increases after a given time period and hashing the resulting values using a cryptographic hash function. Hence, the identifier $ID_{A\to B} = hash(K_{A\to B}||N_t)$ would change in a fixed interval - even if no messages were exchanged at all. This would prevent the server from identifying clients that disconnect for several days and would otherwise reconnect using the same identifiers. However, a fixed point in time at which the nonce changes (i.e. a unix timestamp rounded to the current day) is not a good choice. Assuming synchronized clocks often is error-prone[13]. If all clients would update their identifiers at a given point in time, e.g. at midnight, the server could detect a correlation between all identifiers of a user whose clock is out of sync. Thus, the nonces will be handled per contact and the secret keys

[13]The need of secure time synchronization protocols lead to a number of secure time synchronization concepts such as ANTP [DSZ16], NTS (`https://tools.ietf.org/html/draft-ietf-ntp-network-time-security-14`) or Roughtime (`https://roughtime.googlesource.com/roughtime`).

$K_{A\to B}$ and $K_{B\to A}$ are used to generate two points in time during the next time period at which the nonces $N_{A\to B}$ and $N_{B\to A}$ will be increased. This results in a different update time for all identifiers of dead drops a user connects to.

When a client A wants to send a message m to client B, it encrypts the message using Authenticated Encryption with Associated Data (AEAD) provided by the Networking and Cryptography library[14] (NaCl) so that only B can read it and stores it in the dead drop $ID_{A\to B}$. Note, that the ciphertext of the encrypted message m does not reveal any information about the sender or the receiver as explained in [Ber09, Sect. 9].

Querying for contacts using PIR is also done in a fixed interval, e.g. once a day, to discover new users of the system. It is mandatory not to write to the dead drop specified by the shared secret immediately after receiving a new public key. This would allow an adversary to correlate a PIR request of a given user with (multiple) new requests to the dead drop database, even if the server does not know which public keys were queried. Instead, the first access to the dead drop database for new identifiers is delayed until a point in time between the current query and the next one derived from the shared key. Note, that this delay is only necessary when a new contact is discovered. New users joining the service will therefore also have to delay their first interaction with other users. This can be represented with the concept of "friend requests" in the user interface.

3.5 Security Assumptions

To ensure the security guarantees of OnionPIR, it is necessary to trust on the security guarantees of the underlying protocols. First, it is assumed that RAID-PIR is secure and does not leak any metadata about the information that have been queried by a client. A security argumentation why this is the case is given in [DHS14]. In particular, it is important that the PIR servers are run by different operators that fulfill the non-collusion assumption of RAID-PIR. Note, that the security guarantees of RAID-PIR are still fulfilled if $r-1$ out of r server operators collude with the other ones where r is the redundancy parameter[15] introduced in [DHS14]. A good choice for the operators would be a number of NGOs located in different legal territories. It is also mandatory that the different PIR servers are not in (physical) control of the same data center operator.

Next, it is assumed that Tor provides anonymity for the users that tunnel their connections through this anonymity network. This assumption implies that there is no global passive adversary which is able to monitor and analyze the traffic of the users and colludes with the operator of the OnionPIR servers or gains unauthorized access to them. Note, that it is necessary to at least deanonymize two specific Tor connections in order to learn if two users are communicating with each other or not. Therefore, an attacker would have to be able to deanonymize all users of the service to gain the full social graph of a given user.

[14] `https://nacl.cr.yp.to/`
[15] in the trivial case, r equals the number of PIR servers

An attacker that is only ably to attack a small number of preselected users would have to already hold some information about possible communication partners to verify or falsify these connections. OnionPIR does not put any effort in hiding that a user is using the service at all. It is also worth mentioning that the non-collusion assumption of RAID-PIR also applies to the used anonymity network. It would thus be possible to replace Tor by another anonymity system operated by the owners of the PIR servers or only use Tor nodes that are under control of the participating NGOs.

Anonymity is guaranteed among all honest users of the system. However, a malicious user could announce the list of requests to the dead drop database performed by himself which would remove himself from the anonymity set and effectively decreasing the sets size by 1. Since this would affect the user's own privacy and the anonymity set is typically large, this attack is negligible. Furthermore, it is also assumed that all end systems are secure and no user knowingly reveals any information about his contacts on purpose. Note, that no user is able to gain any information about communication channels it is not participating in. In addition, no user is able to prove that a communication took place. This is the case because the Authenticated Encryption with Associated Data (AEAD) used to encrypt the messages guarantees repudiability.

OnionPIR relies on a Trust On First Use (TOFU) strategy to lessen the burden of manually exchanging public keys through a trusted third channel. Nevertheless, some additional precautions were taken to minimize the risk of a server distributing bad public keys by detecting such attacks. For that purpose, a client queries not only for the public keys of its contacts, but also for its own public key. Since the PIR servers are not able to determine which PIR blocks are being requested during a query, they are not able to manipulate the resulting response in a meaningful way unless they collude. This query can be performed at nearly no cost when querying together with other contacts using RAID-PIR's the multi-block query. Of course, it is still possible, but not required, to provide additional security by adding additional out-of-band key verification techniques or publicly announce a user's public key on a personal website. Another interesting option would be to implement some plausibility checks for the updates of the PIR database in the PIR mirrors and thereby extend the existing anytrust model.

Access to the dead drops used to store messages is not protected against any type of unauthorized manipulation by third parties at all. Since only the communication partners privately agreed on the identifier for the dead drop, this is not necessary. An adversary that is interested in deleting the messages for a specific client would have to brute-force the identifier of the dead drop which is impossible in practice. Protection against a server deleting messages or blocking access to the system is out of scope of this work. Federated or completely decentralized systems would be needed to solve this issue.

Protection against malicious clients requesting to store large amounts of useless data in the dead drops is not implemented but could easily be achieved by making use of blind signatures [Cha83; CFN90]. A client could encrypt a number of random tokens and authenticate against the server who will then blindly sign them. The tokens can then be decrypted by the client

and sent to the server with each write access to the dead drops. While the server is able to determine that these tokens have a valid signature, it is not able to identify the client who generated them. Since a server will only sign a fixed number of tokens in a given time interval per client, this approach can be used to rate-limit the write requests to the database.

3.6 Analysis

3.6.1 Complexity and Efficiency

OnionPIR was designed to provide efficient anonymous communication. Many of the existing systems summarized in §3.2 either require a high communication overhead or come with high computational costs. In order to create a scalable network, it is important to make communication as cheap as possible. The dead drop database used in OnionPIR is therefore combined with scalable onion routing and can be implemented as a simple key-value storage. These type of servers needed for the communication phase can be deployed with very low operating costs, comparable to traditional communication services.

The relevant part in terms of scalability of the system is the initialization phase in which the PIR requests are performed. As shown in section §2.3, PIR is a valid choice that offers reasonable performance and allows users to detect malicious actions of the servers. The size of the PIR database used to perform the benchmarks in §2.3 (about 3.8 GB) would be sufficient to store the public keys for 126 Mio. users.

As the number of users grows, the database size and the server-side computational overhead will grow linearly due to the higher amount of data involved during the XOR operations. The ingress traffic for a PIR query also depends on the number of blocks B in the database and therefore also scales linearly. Thanks to the pseudo random generators (PRG) introduced in [DHS14], the size of a PIR query is $B/8$ Byte for all servers together (excluding the seeds for the PRGs and the communication overhead for lower level transport protocols). However, as long as the block size b does not change, the egress bandwidth is constant since the size of the response does only depend on the block size (and the number of chunks for multi-block queries).

In §2.3 it was shown that an optimized multi-block PIR query for 190 blocks of 64 KB each in the database with uniformly distributed entries and $k = 3$ servers with redundancy parameter $r = 2$ took about 4 seconds (Figure 2.6). This scenario is relatively close to the OnionPIR use case. If each user has around 95 contacts on average, this time decreases to about 2 seconds and represents a pessimistic upper bound to the computational overhead (since this time also includes the communication overhead for the WAN setup). The calculation performed at each mirror is deterministic and only depends on the query from the client and the static database which allows replication to equally divide the computational overhead per instance. Since each of the servers used to answer the PIR request in the test scenario features eight

cores and only one was used in our test scenario, eight mirror instances could be deployed per server. As shown in the benchmarks, three servers with one instance each are able to handle $(24 \cdot 60 \cdot 60)s/2s = 43\,200$ users per day. When this setup is replicated three times, resulting in nine servers, and eight instance are executed per server the whole setup would be sufficient to handle $43\,200 \cdot 8 \cdot 3 = 1\,036\,800$ users for an epoch duration of 24 hours, as suggested in §3.4.

The multi-block queries for 190 blocks in the uniformly distributed database result in 69 individual requests returning 2.75 relevant blocks on average. Since there are 61 590 blocks in the database and the seed used for the pseudo random generator is 16 Bytes long, the data sent to each server sums up to $69 \cdot (\lceil 61\,590/8 \rceil \text{ Byte} + 16 \text{ Byte}) \approx 532$ KByte. Each server that runs eight instances and serves 43 200 clients will therefore receive around 171 GB per day and has an average ingress data rate of 16.25 Mbit/s.

For each query, $2 \cdot 65\,536$ Byte $= 131\,072$ Byte are sent per mirror in response to a client's request, resulting in a daily egress traffic of $43\,200 \cdot 69 \cdot 131\,072 \text{ Byte} \cdot 8 \approx 2\,911$ GByte or 276 Mbit/s per server.

3.6.2 Correctness

Correctness of RAID-PIR is defended in [DHS14], correctness of Tor is explained in [DMS04] and has already been well-proven in practice. Messages are acknowledged and retransmitted if the identifiers changed and the message has not been read yet. Therefore it is guaranteed that messages will reach their desired destination. Correctness of the ECDH key exchange is justified in [Ber09]. All operations involved in the private establishing of identifiers for the dead drops are deterministic and therefore result in the same identifiers for both communication partners.

3.7 Implementation

We implemented OnionPIR in Python using RAID-PIR as the private information retrieval library, the Networking and Cryptography library (NaCl) for elliptic curve encryption and Stem[16] as a controller library for Tor. The system was divided into a client including a web server to serve the user interface, the OnionPIR server handling user registrations and acting as a RAID-PIR vendor and the PIR mirrors that are used to answer the PIR requests. The client typically runs at the user's end system but could also be running on a private home server in control of the user. The OnionPIR server is typically operated by the service provider while the PIR mirrors should be operated by a number of third-parties.

In order to increase reusability of the code in future clients, major functionalities, such as sending or receiving messages, the PIR client and the registration process, were bundled in

[16] `https://stem.torproject.org/`

the *OnionPIR client library*. The user interface for the client was built using web technologies including AngularJS[17], jQuery[18] and Bootstrap[19]. CherryPy[20] was used as a webserver that communicates with the user interface.

We designed the client – as well as the OnionPIR protocol – with usability in mind. The user will therefore never see any cryptographic keys or other technical data. However, it is possible to enable a so called "demonstrator mode" where it is possible to see which operations are performed in the background. When this option is enabled, technical information, such as the identifier for the dead drop being used to send and receive a message, is shown.

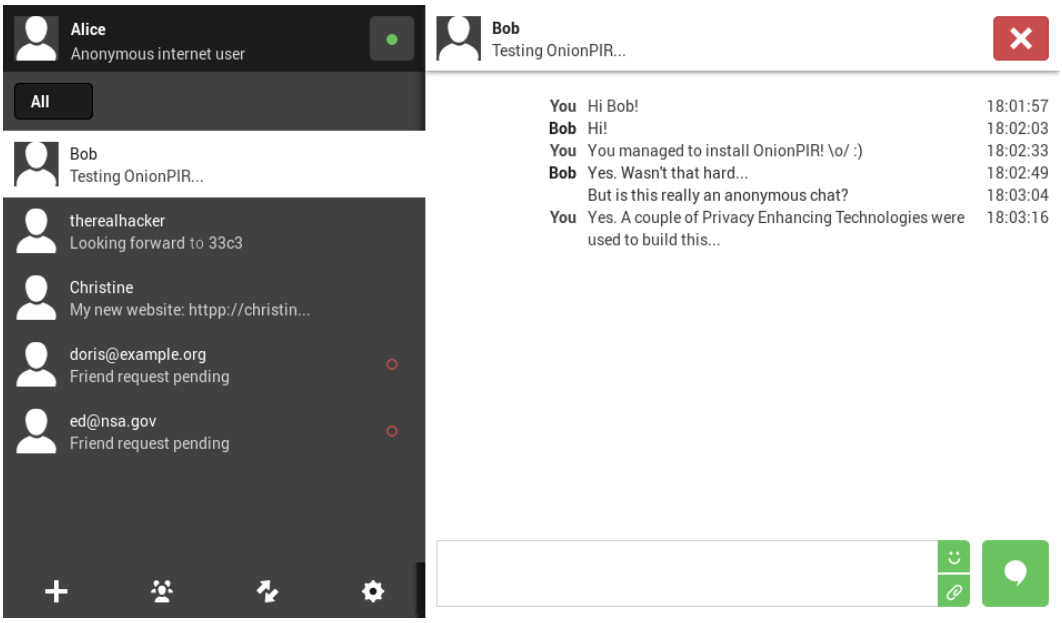

Figure 3.3: Screenshot of the OnionPIR client

Currently, only email addresses can be used as an identifier for contacts. Since identifiers are hashed before used to determine the location inside the PIR database at which the user's public key is stored, this is only an artificial restriction. Allowing any type of user names together with a first-come-first-serve policy or connecting the OnionPIR server to a SMS gateway to verify phone numbers would also be valid deployment scenarios.

The implementation is intended for testing and demonstration purposes only. For example, only a minimalistic end-to-end encrypted chat protocol was implemented in the client. The protocol does offer repudiability but, in contrast to more advanced end-to-end messaging protocols like the Signal Protocol, does not implement forward secrecy.

[17] `https://angularjs.org/`
[18] `https://jquery.com/`
[19] `https://getbootstrap.com/`
[20] `http://cherrypy.org/`

4 Conclusion

In this thesis, two optimizations to the RAID-PIR library [DHS14] were introduced, implemented and evaluated: Preprocessing the entries of the PIR database using the Method of Four Russians to speedup the generation of PIR responses and uniformly distributing those entries to maximize the number of entries that can be fetched with one multi-block query. We have shown that both improvements lead to a significant speedup for the designated use cases. By combining both optimizations, the practical performance measured in a realistic test scenario nearly reached its theoretic optimum and leaves almost no room for improvements in some test cases.

An interesting topic for further research would be to query a variable number of blocks in a single PIR request. This could be achieved by grouping the blocks of the PIR database into a client-defined number of *response groups* and generate one PIR response per group – similar to RAID-PIR's multi-block scheme where responses are generated per chunk. In combination with the presented uniform distribution of the data entries, most files could be requested in a single query which would drastically decrease the amount of data that has to be sent to the servers and also would eliminate the latency impact of the client's connection.

In the second part of the thesis, an overview over existing anonymous communication protocols was given and their building blocks were summarized. Later, RAID-PIR was used in combination with onion routing to build OnionPIR, a private communication system that was designed to achieve a tradeoff between the strong but computationally expensive privacy guarantees of PIR and the efficient but non information-theoretically secure onion routing. Using PIR to distribute public keys has shown to be an interesting approach to spot some kinds of attacks performed by the server, including the recognition of MITM attacks when both communication partners are actively using the service and are periodically querying for their public keys. Additionally, a prototype was implemented to show that the system is usable in practice.

List of Figures

List of Tables

Abbreviations

PET Privacy Enhancing Technology

PIR Private Information Retrieval

IT-PIR Information-Theoretic Private Information Retrieval

cPIR Computational Private Information Retrieval

DPF Distributed Point Function

PRG Pseudo Random Generator

HTTP Hypertext Transfer Protocol

NaCl Networking and Cryptography library

AEAD Authenticated Encryption with Associated Data

DP5 Dagstuhl Privacy Preserving Presence Protocol

NGO Non-Governmental Organization

TOFU Trust On First Use

ECDH Elliptic Curve Diffie-Hellman (key exchange)

MITM Man-In-The-Middle

Bibliography

[ABFK16] C. AGUILAR-MELCHOR, J. BARRIER, L. FOUSSE, M.-O. KILLIJIAN. "XPIR: Private Information Retrieval for Everyone". In: *Proceedings on Privacy Enhancing Technologies (PETS'16)* (2016), pp. 155–174 (cit. on p. 3).

[ABH10] M. R. ALBRECHT, G. V. BARD, W. HART. "Efficient multiplication of dense matrices over GF(2)". In: *ACM Transactions on Mathematical Software, 37(1)* (2010) (cit. on p. 6).

[ADKF70] V. ARLAZAROV, E. DINIC, M. KRONROD, I. FARADZEV. "On economical construction of the transitive closure of a directed graph". In: *USSR Academy of Sciences 134* (1970) (cit. on p. 5).

[BBS04] D. BONEH, X. BOYEN, H. SHACHAM. "Short group signatures". In: *24th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2004), pp. 41–55 (cit. on p. 19).

[BDG14] N. BORISOV, G. DANEZIS, I. GOLDBERG. "DP5: A Private Presence Service". In: *Centre for Applied Cryptographic Research (CACR), University of Waterloo* (2014) (cit. on pp. 15 sq.).

[Ber09] D. J. BERNSTEIN. "Cryptography in NaCl". In: *Networking and Cryptography library* (2009) (cit. on pp. 25, 28).

[BF01] D. BONEH, M. K. FRANKLIN. "Identity-based encryption from the Weil pairing". In: *21st Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2001), pp. 213–229 (cit. on pp. 16, 21).

[BG12] S. BAYER, J. GROTH. "Efficient zero-knowledge argument for correctness of a shuffle". In: *31st Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2012), pp. 263–280 (cit. on p. 19).

[Blo70] B. H. BLOOM. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426 (cit. on p. 15).

[BV14] J. BUDURUSHI, M. VOLKAMER. "Feasibility analysis of various electronic voting systems for complex elections". In: *International Conference for E-Democracy and Open Government 2014, Edition Donau-Universität* (2014) (cit. on p. 1).

[CBM15] H. CORRIGAN-GIBBS, D. BONEH, D. MAZIÈRES. "Riposte: An anonymous messaging system handling millions of users". In: *36th IEEE Symposium on Security and Privacy* (2015) (cit. on pp. 17, 20).

[CFN90] D. CHAUM, A. FIAT, M. NAOR. "Untraceable electronic cash." In: *Advances in Cryptology (CRYPTO'90)* (1990), pp. 319–327 (cit. on p. 26).

[CGN97] B. CHOR, N. GILBOA, M. NAOR. *Private Information Retrieval by Keywords*. 1997 (cit. on p. 3).

[CH91] D. CHAUM, E. v. HEYST. "Group signatures". In: *Eurocrypt 1991* (1991), pp. 257–265 (cit. on p. 19).

[Cha81] D. L. CHAUM. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Communications of the ACM* (1981), pp. 84–90 (cit. on pp. 16, 19, 21).

[Cha83] D. CHAUM. "Blind Signature Systems". In: *Advances in Cryptology (CRYPTO'83)* (1983), p. 153 (cit. on p. 26).

[CJK+16] D. CHAUM, F. JAVANI, A. KATE, A. KRASNOVA, J. d. RUITER, A. T. SHERMAN, D. DAS. "cMix: Anonymization by High-Performance Scalable Mixing". In: *Cryptology ePrint Archive, Report 2016/008* (2016) (cit. on pp. 16, 21).

[CKGS95] B. CHOR, E. KUSHILEVITZ, O. GOLDREICH, M. SUDAN. "Private information retrieval". In: *Foundations of Computer Science (FOCS'95)* (1995), pp. 41–50 (cit. on pp. 3, 8, 11 sqq., 20).

[Coc01] C. COCKS. "An Identity Based Encryption Scheme Based on Quadratic Residues". In: *8th IMA International Conference on Cryptography and Coding* (2001) (cit. on pp. 16, 21).

[DG14] C. DEVET, I. GOLDBERG. "The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency". In: *Proceedings on Privacy Enhancing Technologies (PETS'14)* (2014), pp. 63–82 (cit. on p. 3).

[DHS14] D. DEMMLER, A. HERZBERG, T. SCHNEIDER. "RAID-PIR: Practical multi-server PIR". In: *6th ACM Cloud Computing Security Workshop (ACM CCSW'14)* (2014), pp. 45–56 (cit. on pp. 2 sqq., 9, 11 sq., 20, 25, 27 sq., 30).

[DMS04] R. DINGLEDINE, N. MATHEWSON, P. SYVERSON. "Tor: The Second-generation Onion Router". In: *13th USENIX Conference on Security Symposium* (2004), pp. 21–21 (cit. on pp. 14, 21 sq., 28).

[DSZ14] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. "Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens". In: *23rd USENIX Conference on Security Symposium* (2014), pp. 893–908 (cit. on p. 1).

[DSZ16] B. DOWLING, D. STEBILA, G. ZAVERUCHA. "Authenticated Network Time Synchronization". In: *25th USENIX Conference on Security Symposium* (2016), pp. 823–840 (cit. on p. 24).

[FS01] J. FURUKAWA, K. SAKO. "An efficient scheme for proving a shuffle". In: *21st Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2001), pp. 368–387 (cit. on p. 19).

[GI14] N. GILBOA, Y. ISHAI. "Distributed Point Functions and Their Applications". In: *33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2014), pp. 640–658 (cit. on p. 18).

[HCE11] Y. HUANG, P. CHAPMAN, D. EVANS. "Privacy-preserving applications on smartphones". In: *6th USENIX conference on Hot topics in security* (2011), pp. 4–4 (cit. on p. 1).

[Hen16] R. HENRY. "Polynomial Batch Codes for Efficient IT-PIR". In: *Cryptology ePrint Archive, Report 2016/598* (2016) (cit. on p. 5).

[HLZZ15] J. v. d. HOOFF, D. LAZAR, M. ZAHARIA, N. ZELDOVICH. "Vuvuzela: Scalable private messaging resistant to traffic analysis". In: *25th Symposium on Operating Systems Principles (SOSP'15)* (2015) (cit. on p. 17).

[KLDF16] A. KWON, D. LAZAR, S. DEVADAS, B. FORD. "Riffle: An Efficient Communication System With Strong Anonymity". In: *Privacy Enhancing Technologies (PETS'16)* (2016), pp. 115–134 (cit. on pp. 19, 21).

[KO97] E. KUSHILEVITZ, R. OSTROVSKY. "Replication is not needed: Single database, computationally-private information retrieval". In: *Foundations of Computer Science (FOCS'97)* (1997), pp. 364–373 (cit. on p. 3).

[KS08] V. KOLESNIKOV, T. SCHNEIDER. "A Practical Universal Circuit Construction and Secure Evaluation of Private Functions". In: *Financial Cryptography and Data Security (FC'08)* (2008), pp. 83–97 (cit. on p. 15).

[KS16] Á. KISS, T. SCHNEIDER. "Valiant's universal circuit is practical". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2016), pp. 699–728 (cit. on p. 15).

[Lan15] S. LANDAU. "Mining the Metadata: And Its Consequences". In: *37th International Conference on Software Engineering* (2015), pp. 4–5 (cit. on p. 1).

[LG15] W. LUEKS, I. GOLDBERG. "Sublinear scaling for multi-client private information retrieval". In: *International Conference on Financial Cryptography and Data Security* (2015), pp. 168–186 (cit. on p. 8).

[LWN+15] C. LIU, X. S. WANG, K. NAYAK, Y. HUANG, E. SHI. "ObliVM: A Programming Framework for Secure Computation". In: *36th IEEE Symposium on Security and Privacy* (2015), pp. 359–376 (cit. on p. 15).

[LZ16] D. LAZAR, N. ZELDOVICH. "Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata". In: *Hot Topics in Privacy Enhancing Technologies (HotPETS'16)* (2016) (cit. on pp. 16, 21).

[MCA06] M. F. MOKBEL, C.-Y. CHOW, W. G. AREF. "The New Casper: Query Processing for Location Services Without Compromising Privacy". In: *32nd International Conference on Very Large Data Bases* (2006), pp. 763–774 (cit. on p. 1).

[Mea86] C. MEADOWS. "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third part". In: *IEEE S&P'86* (1986) (cit. on p. 15).

[MMM16] J. MAYER, P. MUTCHLER, J. C. MITCHELL. "Evaluating the privacy properties of telephone metadata". In: *National Academy of Sciences* 113.20 (2016), pp. 5536–5541 (cit. on p. 1).

[MOT+11] P. MITTAL, F. OLUMOFIN, C. TRONCOSO, N. BORISOV, I. GOLDBERG. "PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval". In: *20th USENIX Conference on Security Symposium* (2011), pp. 31–31 (cit. on p. 1).

[Nef01] C. A. NEFF. "A verifiable secret shuffle and its application to e-voting". In: *8th ACM Conference on Computer and Communications Security (CCS'01)* (2001), pp. 116–125 (cit. on p. 19).

[Neu16] S. NEUMANN. "Evaluation and Improvement of Internet Voting Schemes Based on Legally-Founded Security Requirements". PhD thesis. Darmstadt, Germany: Technische Universität Darmstadt, 2016 (cit. on p. 1).

[PSSZ15] B. PINKAS, T. SCHNEIDER, G. SEGEV, M. ZOHNER. "Phasing: Private set intersection using permutation-based hashing". In: *24th USENIX Conference on Security Symposium* (2015), pp. 515–530 (cit. on p. 15).

[PSZ14] B. PINKAS, T. SCHNEIDER, M. ZOHNER. "Faster Private Set Intersection Based on OT Extension". In: *23th USENIX Conference on Security Symposium* (2014) (cit. on p. 15).

[SC05] L. SASSAMAN, B. COHEN. "The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval". In: *Workshop on Privacy in the Electronic Society (WPES'05* (2005), pp. 1–9 (cit. on p. 20).

[Sha84] A. SHAMIR. "Identity-Based Cryptosystems and Signature Schemes". In: *Advances in Cryptology (CRYPTO'84), Lecture Notes in Computer Science* (1984), pp. 47–53 (cit. on pp. 16, 21).

[SN16] A. SANATINIA, G. NOUBIR. "HOnions: Towards Detection and Identification of Misbehaving Tor HSDirs". In: *Hot Topics in Privacy Enhancing Technologies (HotPETS'16)* (2016) (cit. on p. 19).

[WHC+14] X. S. WANG, Y. HUANG, T.-H. H. CHAN, A. SHELAT, E. SHI. "SCORAM: Oblivious RAM for Secure Computation". In: *2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 191–202 (cit. on p. 15).

[Yao82] A. C. YAO. "Protocols for secure computations". In: *23rd IEEE Symposium on Foundations of Computer Science* (1982), pp. 160–164 (cit. on p. 15).