# OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks

Daniel Demmler, Marco Holz$^{(\boxtimes)}$, and Thomas Schneider

Technische Universität Darmstadt, Darmstadt, Germany
{daniel.demmler,thmomas.schneider}@cysec.de, onionpir@marcoholz.de

**Abstract.** While great effort has been put into securing the *content* of messages transmitted over digital infrastructures, practical protection of *metadata* is still an open research problem. Scalable mechanisms for protecting users' anonymity and hiding their social graph are needed. One technique that we focus on in this work is private information retrieval (PIR), an active field of research that enables private querying of data from a public database without revealing which data has been requested and a fundamental building block for private communication. We introduce two significant improvements for the multi-server scheme RAID-PIR (ACM CCSW'14): precomputing queries using the *Method of four Russians* and optimizing the database layout for parallel queries. We then propose *OnionPIR*, an anonymous messaging service as example application for PIR combined with onion routing that prevents the leakage of communication meta-data. By providing and evaluating a prototype, we show that OnionPIR is usable in practice. Based on our results, we conclude that it is possible to build and deploy such a service today, while its operating expenses are within the order of magnitude of those of traditional messaging services that leak metadata.

**Keywords:** Private information retrieval · Tor · Privacy · Meta-data protection

## 1 Introduction

Communication has never been more important for our society than it is today. Large parts of our 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 availability of mobile devices, communication shifted from the offline world to online communication platforms. End-to-end encryption is nowadays often deployed to protect the content of exchanged messages such that it cannot be read by an adversary. However, these digital platforms produce large amounts of metadata, i.e., who communicates with whom at what time. 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

D. Gollmann et al. (Eds.): ACNS 2017, LNCS 10355, pp. 599–619, 2017.
DOI: 10.1007/978-3-319-61204-1_30

suppressive regimes, for whom trying to contact government-critical organizations can be a tremendous risk. The right to remain anonymous is a fundamental right in our modern society.

Developing systems that protect the users' privacy and at the same time scale to a large number of users is challenging and a very active research field. Private information retrieval (PIR) can be used as a fundamental building block for those systems and also has many other applications, e.g., private querying of entries from patent registers or medical databases, 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] or privacy-preserving location-based services [MCA06,HCE11,DSZ14].

**Outline and Our Contributions.** In Sect. 2, we propose and implement two optimizations for the multi-server PIR scheme RAID-PIR [DHS14] that lead to significant performance improvements. The optimization reduces the server's computation time by using the Method of four Russians for precomputation (Sect. 2.2), thus increasing the overall throughput of the system. The second optimization targets the process of privately retrieving multiple database entries within a single PIR query by adjusting the database layout to increase the number of entries that can be queried in parallel (Sect. 2.3). After introducing multiple generic implementation enhancements (Sect. 2.4), we perform benchmarks to evaluate our optimizations and measure speedups of up to factor 7x over RAID-PIR for many cases when querying through a WAN connection and a performance close to the theoretical optimum when using a DSL connection for the queries (Sect. 2.5).

Based on our improved implementation of PIR, we propose *OnionPIR*, a novel anonymous communication system that combines PIR with onion routing to create a scalable way to privately exchange messages (Sect. 3). OnionPIR is designed to be ready for deployment and provides a way to establish a secure communication channel without the need of exchanging any type of cryptographic keys out-of-band. We built a proof-of-concept implementation and our evaluation demonstrates its practical performance. We will publish our implementations as open-source software upon acceptance of this paper.

Finally, we provide an overview and comparison with related work in Sect. 4.

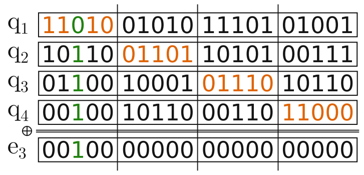
## 2 Private Information Retrieval and Improvements

Private information retrieval (PIR) is an active research topic and enables the querying of information from one or multiple servers without disclosing which information was requested. Since this technique is a fundamental building block for higher-level protocols and applications, its performance is of prime importance. In this section, we briefly summarize RAID-PIR [DHS14] and introduce and implement two significant performance improvements.

## 2.1 PIR Background

PIR protocols can be grouped into information-theoretic schemes and computational schemes. RAID-PIR [DHS14], the work that we base our system on, is a multi-server scheme based on the original information-theoretic design by Chor et al. [CKGS95]. Its security guarantees rely on a non-collusion assumption between multiple servers. By using fast XOR operations, it achieves great performance. The alternatives are single-server PIR schemes that rely on computational assumptions, as first introduced in [CGN98] and soon followed by another approach [KO97]. These schemes are typically based on computationally expensive, partially homomorphic encryption. A recent scheme using lattice-based cryptography is presented in [AMBFK16]. Both approaches were combined in [DG14] in a design called hybrid PIR.

**Notation.** Throughout the paper we use the following notation: The database that may be queried by clients is divided into $B$ blocks of size $b$ bits each 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 is then sent to a different server which will XOR all blocks whose corresponding bit $i$ is set in the query. Each server returns the resulting block of XORs of length $b$ bits. To recover the $n$-th block, the clients computes the XOR of all received blocks.



| | | | | |
|---|---|---|---|---|
| $q_1$ | 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 |

**Fig. 1.** RAID-PIR Query. 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 the plaintext query $e_3$ that retrieves the third database block. (Color figure online)

RAID-PIR improves this scheme by splitting each query into $k$ chunks, as shown in Fig. 1. The queries are constructed in a way that all chunks sent to the $l$-th server, except the $l$-th chunk are generated from a seed using a pseudo random generator (PRG). In addition, a redundancy parameter $r$ is introduced as trade-off between security and performance and chosen such that $2 \leq r \leq k$. It allows reducing 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 plaintext of a query from $k$ to $r$.

Another optimization provides the ability to request multiple blocks of the database within a single query. This is achieved by XORing the requested blocks only within a 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 the following sections, we introduce two additional improvements to RAID-PIR that further increase its efficiency.

The first one speeds up the generation of the PIR responses at the server-side by using precomputations (Sect. 2.2), while the second one achieves a significant speedup when querying multiple blocks in parallel by using a database layout where blocks are uniformly distributed (Sect. 2.3).

## 2.2 Method of Four Russians

The *Method of four Russians* [ADKF70], is a technique for matrix multiplication with a limited number of possible values per entry. Since RAID-PIR makes heavy usage of such kinds of multiplications, this method can be used to reduce the computation time of the RAID-PIR servers for generating responses for PIR queries. This leads to lower latency and higher throughput and comes at the (low) cost of a preprocessing phase for the servers and increased memory requirements.

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 $\mathsf{X} \cdot \mathsf{Y} = \mathsf{Z}$ are done over the field with two elements ($\mathbb{F}_2$), i.e. only binary elements are being multiplied. Note, that this property is not required for the algorithm (i.e., it is also applicable to numbers in $\mathbb{Z}$) 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*.

Crucial for understanding the idea behind the four Russians algorithm, is the fact that the indices of all non-zero elements in row $r$ in the first matrix $\mathsf{X}$ indicate a subset of rows in the second matrix $\mathsf{Y}$ that have to be XORed in order to produce the $r$-th row of the output matrix $\mathsf{Z}$. As a naïve approach, it would now be possible to precompute all possible XOR combinations of all rows in $\mathsf{Y}$, hereby creating a lookup table (LUT) that returns the final results for the rows of $\mathsf{Z}$. In other words, a lookup could be performed using each row in $\mathsf{X}$ as key to the LUT, returning the corresponding row in $\mathsf{Z}$. This would reduce the computational complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$ assuming the lookup is done in constant time for matrices of size $n \times n$. Unfortunately, this naïve approach is impractical due to the exponential computational and memory complexity of the precomputation of $\mathcal{O}(n \cdot 2^n)$, which is worse than the regular approach for $n \geq 4$.

The reason why only $n$ XOR operations have to be performed for each of the $2^n$ possible combinations of rows of the matrix $\mathsf{Y}$ 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 a single bit that corresponds to only one row of $\mathsf{Y}$. For each of the $2^n$ combinations, it is now sufficient to XOR the last precomputed result with the row of $\mathsf{Y}$ that changed in the Gray code. The binary Gray code $g$ of a number $m$ can be calculated as $g = (m \oplus (m \gg 1))$. In Fig. 2 we depict the calculation of LUT entries for a given example matrix $\mathsf{Y}^*$.

$$\mathsf{Y}^* = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

| Gray code | decimal | result |
|---|---|---|
| 0000 | 0 | 0000 |
| 0001 | 1 | $1011 = 0000 \oplus \mathsf{Y}^*[4,:]$ |
| 0011 | 3 | $0010 = 1011 \oplus \mathsf{Y}^*[3,:]$ |
| 0010 | 2 | $1001 = 0010 \oplus \mathsf{Y}^*[4,:]$ |
| 0110 | 6 | $0101 = 1001 \oplus \mathsf{Y}^*[2,:]$ |
| ... | ... | ... |

**Fig. 2.** Example LUT precomputation for matrix $\mathsf{Y}^*$. $\mathsf{Y}^*[n,:]$ denotes the $n$-th row of matrix $\mathsf{Y}^*$.

Instead of precomputing a LUT for the full matrix $\mathsf{Y}$, it is divided into groups of $t \in \mathbb{N}$ rows each. The precomputation is then done within these $n/t$ groups using the Gray code as described above, resulting in a computational complexity of $\mathcal{O}(2^t \cdot n^2/t)$. If $t = \log_2(n)$ the complexity simplifies to $\mathcal{O}(n^3/\log_2(n))$, resulting in a speedup of $t$, compared to traditional matrix multiplication, for all queries that will be answered after the precomputation phase.

While the matrix $\mathsf{Y}$ is divided into groups horizontally, the matrix $\mathsf{X}$ has to be divided into vertical groups of $t$ columns. To multiply the two $n \times n$ matrices, $\mathsf{X}$ is now traversed column-wise, grouped by $t$ columns forming a group each. For each group, the corresponding LUT can now be created by $t$ rows of $\mathsf{Y}$ and a lookup can be performed for all $t$-bit parts of the $n$ rows of $\mathsf{X}$ in this group. This procedure is depicted in Fig. 3. We note that this optimization is generic and can potentially be used in any PIR scheme in which queries can be expressed as matrix-matrix or vector-matrix multiplication, e.g., [CKGS95, LG15, Hen16].

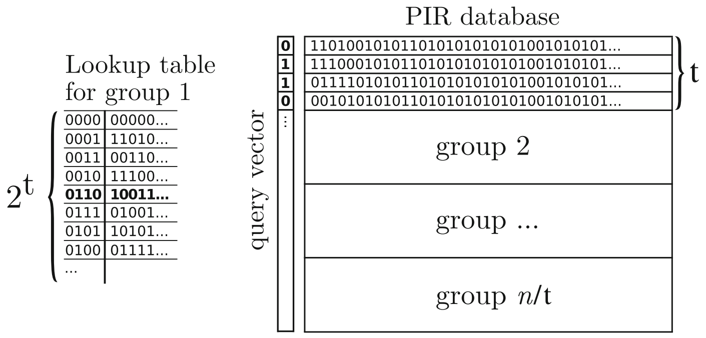


**Fig. 3.** Precomputation in the PIR database for $t = 4$. A query vector is one *row* in the matrix $\mathsf{X}$. The first 4 bits of the query vector represent the index in the LUT for the first group of $\mathsf{Y}$ (the PIR database).

*Implementation:* An efficient implementation of the Method of four Russians is provided in [ABH10]. However, since it only offers full matrix-matrix multiplications and uses different data structures than RAID-PIR, we implemented our own version and directly integrate it into the RAID-PIR library[1] to avoid costly memory movements and exploit memory locality. While the traditional Method of four Russians assumes the multiplication of two full matrices, this is not the case in RAID-PIR. Often, only a single PIR request has to be answered, making $\mathsf{X}$ effectively a vector. Several PIR queries could be batched to create matrix $\mathsf{X}$ at the cost of increased latency of the individual queries, similar to [LG15].

In our implementation we do the LUT precomputation once while loading the database and store the LUTs in main memory. For every query we achieve a theoretical speedup of $t$. The downside of this approach is the increased memory requirement. Without loss of generality, we now assume that $\mathsf{Y}$ is a RAID-PIR database of size $B \times b$, i.e., $B$ blocks of $b$ bits each. While a larger $t$ decreases the computation time for the generation of response vectors, it also excessively increases the memory needed to store the LUTs. When $t$ blocks of the PIR database are put in a group, only $B/t$ XOR operations have to be performed per query, resulting in a speedup of factor $t$. On the other hand, each of the $B/t$ LUTs requires $2^t \cdot b$ bits of memory, where $b$ is the database block size. The choice of $t = 4$ gives a good trade-off between theoretical speedup and memory requirements for larger databases (see Table 1) and is used in our implementation.

**Table 1.** Comparison of speedup and memory for the Method of four Russians

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

The optimizations were implemented in C and integrated into the existing RAID-PIR library. The precomputation only affects the calculation of the XOR responses by the PIR servers and is completely transparent to PIR clients. This means that clients do not have to be aware of the changes and can send the same queries as before. Furthermore, different PIR server operators could decide independently whether they want to enable the precomputation or not.

In [LG15] a related approach to speedup server computation, based on the Strassen algorithm for matrix multiplication is introduced. This comes at the expense of higher latency for each individual query, as multiple queries are collected and processed together. Furthermore, this approach only introduces a speedup of $q/q^{0.80735} = q^{0.19265}$, where $q$ is the number of queries that are processed together, which is less efficient than our approach (constant speedup $t$) when $q$ is small (e.g., for $q < 1334$ and $t = 4$).

## 2.3 Uniform Distribution of Data Entries

In the following we present an optimization that is specific to so-called multi-block (MB) queries as proposed in RAID-PIR, that enable a client to privately request more than one block in a single query. The improvements of RAID-PIR,

[1] https://github.com/encryptogroup/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. PIR servers receiving an MB query will reply with one block per chunk. The XOR of the corresponding blocks is calculated as before, but only within the boundaries of these chunks. 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 showed 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 many smaller files that were distributed among the database. In RAID-PIR, files larger than the block size $b$ are simply placed in adjacent blocks to build the database. While this does not affect the performance of a query when retrieving the blocks consecutively, querying multiple blocks in parallel is often not possible, as they most likely happen to be located in the same chunk.

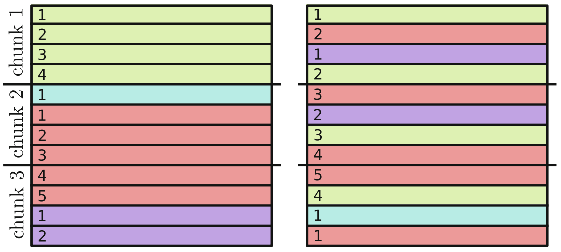


**Fig. 4.** Uniform distribution of the data entries in the PIR database. While the first file (green) filled up only the first chunk in the original RAID-PIR database layout (left), it is now uniformly distributed over the whole database (right). (Color figure online)

*Implementation:* To improve the performance of multi-block queries for files larger than the block size $b$, we change the layout of the database. Instead of placing files in consecutive blocks, they are now uniformly distributed over the whole database, as shown in Fig. 4. For each file, the number of blocks $n$ needed to store its contents is calculated. The file is then placed such that between two of its blocks $B/n - 1$ blocks are used for other files, where $B$ is the total number of blocks. If a chosen block is already assigned to another file, the next possible 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. It is guaranteed that all $B$ blocks of the database contain data and no block, except the last, one is only partially filled. Since both the PIR servers and the clients have to know the new locations of the database entries, both client-side and server-side code is extended to support the new layout.

## 2.4 Generic Implementation Improvements

We also introduced several implementation improvements in RAID-PIR, that are included in all measurements in Sect. 2.5. We pipelined the communication on the client side and decoupled the sending of queries and reception of answers. Furthermore, only a single seed is sent from the client to the servers during a session and a state is kept while the connection is alive. We also introduced SSE2 intrinsics on very wide data types. This feature is available in all x86 CPUs since several years and leads to a more efficient bit-wise XOR computation, which is one of the most crucial operations in RAID-PIR for both clients and PIR servers.

## 2.5 PIR Benchmark Results

In this section, we evaluate the performance of our modified implementation of RAID-PIR [DHS14]. All measurements include the generic optimizations from Sect. 2.4. We show the influence of the optimizations from Sects. 2.2 and 2.3.

**Benchmark Setting.** Three PIR servers were deployed as `r3.xlarge` instances on Amazon EC2 with 30.5 GiB RAM, 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 RAM, one core of an Intel Xeon E5-2676 v3 processor and a 250 Mbit/s ethernet connection (0.5 ms latency) in the WAN setup and a notebook with 8 GiB RAM, an Intel Core i3-3120M processor and a consumer grade DSL Internet connection (4.5 Mbit/s downstream, 400 kbit/s upstream, 30 ms latency) in the DSL setup. The used database consists of 964 files of Ubuntu security updates adding up to a total size of 3.8 GiB. All results are average runtimes of 5 iterations. Note that the y-axes in the figures are in logarithmic scale.

**PIR Server Startup Duration.** The time to load the database into the PIR servers' RAM is mostly independent of the block size and took ≈40 s for the database of 3.8 GiB. The four Russian precomputation took between 5 s and 7 s. The startup time scales linearly with the database size.

**File Query via WAN.** In the remainder of this section, the number of servers is set to $k = 3$ and the redundancy parameter is $r = 2$. The block size is varied from $b = 16$ kiB to $b = 4$ MiB.

*Small File Query via WAN:* The results for querying 10 small files adding up to 2.9 MiB in the WAN setup are depicted in the left part of Fig. 5. 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`. Here, the multi-block queries are significantly faster than single-block queries even when no optimizations are applied since the requested files are already distributed among the whole database. Therefore, uniform distribution of the data entries does not have a significant impact on the overall performance. Four Russians precomputation improves the runtimes by factor 2 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 $b = 16$ kiB, the runtime decreases from 10.1 s for the

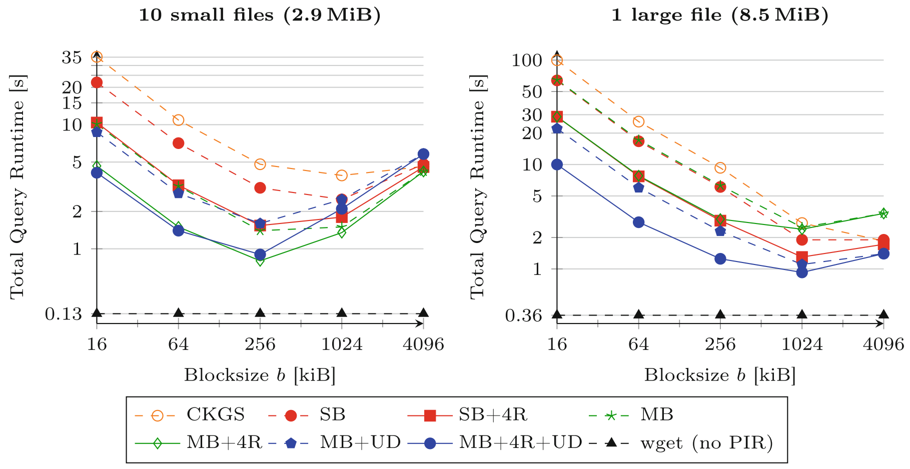


**Fig. 5.** WAN Benchmarks: Runtimes for varying block size $b$ with $k = 3$ servers, redundancy parameter $r = 2$ for 10 small files (2.9 MiB, left) and a large file (8.5 MiB, right). `CKGS`: [CKGS95], `SB`: Single-Block scheme, `MB`: Multi-Block scheme, `4R`: Four Russians precomputation, `UD`: Uniform distribution of data entries. DB size 3.8 GiB.

originally best performing multi-block queries to 4.1 s when all optimizations are applied. The best performance is obtained for a block size of $b = 256$ kiB where 7 queries are performed to retrieve 18 blocks in 0.8 s.

*Large File Query via WAN:* In the right part of Fig. 5, one large 8.5 MiB file is requested. As already observed in [DHS14], single-block and multi-block queries take similar time when querying one large file in the WAN setup, where bandwidth and latency typically are not the bottleneck. The four Russians precomputation leads to a significant speedup and effectively improves the runtime of most test cases by factor 2. This is a good result even though the theoretical speedup, that does not cover data locality and communication overhead, is 4. When the CPU cache size is too small to store the precomputed LUTs and all interim results for large block sizes, the Method of four Russians' speedup decreases and is not measurable any more for $b = 4$ MiB.

The speedup gained by uniformly distributing the data entries in the database is even greater and improves the overall runtime by approximately 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 s (or 100 s for the original CKGS scheme) to 10 s for block size $b = 16$ kiB and reaches its minimum for $b = 1$ MiB where the runtime decreases from 1.9 s for the originally best performing single-block queries to 0.9 s for multi-block queries using both optimizations.

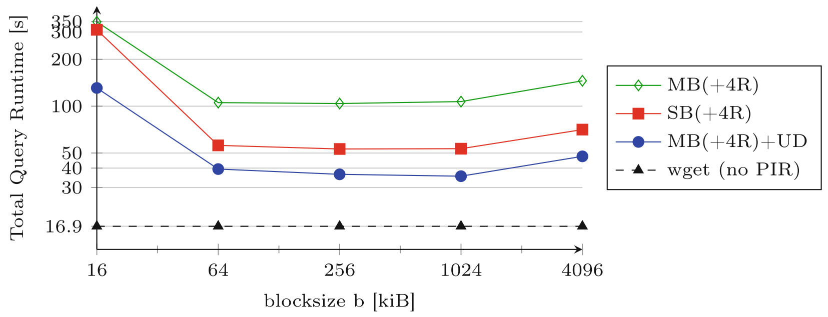


**Fig. 6.** Large File, DSL: Runtimes for varying block sizes $b$ with $k = 3$ servers, redundancy parameter $r = 2$ and one large file (8.5 MiB). `CKGS`: original PIR scheme [CKGS95], `SB`: Single-Block scheme, `MB`: Multi-Block scheme, `4R`: Four Russians precomputation, `UD`: Uniform distribution of data entries. DB size 3.8 GiB.

**Large File Query via DSL.** In Fig. 6, the results for querying one large file (8.5 MiB, 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 $b = 16$ kiB do not differ significantly. The reason for this is the low upstream bandwidth of the DSL connection. For small block sizes, the required upstream bandwidth, which depends on the number of blocks, has a larger impact than the downstream bandwidth, which depends on the block size. To query a database containing $B = 246360$ blocks of 16 kiB each, $B/3$ bits have to be transferred to each server (plus additional 16 Bytes for the PRG seeds) resulting in a total transmission time of $3 \cdot (B/3\,\text{bit} + 16\,\text{Byte})/(400\,\text{kbit/s}) \approx 602$ ms. This phenomenon can also be observed in the benchmarks run in [DHS14].

Due to this high latency and communication overhead of the PIR queries, the server-side four Russians precomputation does not significantly affect the overall runtime of a client's request. As in the first test case, single-block queries provide better performance than multi-block queries without the uniform distribution. However, when the data entries are distributed uniformly, a significant speedup to multi-block queries can be observed 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 in [DHS14] 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 $b = 1$ MiB using both new optimizations. Retrieving the file without PIR, using `wget` over an unencrypted HTTP connection takes 16.9 s and is only 2.1 times faster, indicating that the runtime of 35.5 s for the PIR approach almost reached the theoretically optimal results for the DSL setup, as we need twice the communication. When the network is the bottleneck, computation has only a minor influence on total runtime. Our results also show that the previously assumed optimal block size of $b = B = \sqrt{\text{DB size}}$ does not lead to good performance, especially in the DSL setting with asymmetric up- and downstream bandwidth. For a database size of 3.8 GiB that would be ≈22 kiB block size, which our results show to be far worse than larger block sizes.

**Query Time for Varying Database Size.** To demonstrate how the size of the database influences the query runtime we performed the same queries as in Sect. 2.5 using multi block queries with four Russians precomputation and uniformly distributed database blocks. We varied the size of the database and depict our results in Fig. 7. The time that the clients needs to generate the queries to the servers is almost constant and always around or below 50 ms. The communication varies due to queries being sent over a public Internet connection and also remains mostly constant between 200 ms and 300 ms. This is due to the fact, that even though the database and thus the number of blocks increases, the query size only grows from 250 Bytes to 4 kiB, which is negligible compared to the block size $b = 256$ kiB, that the client receives. Server computation time and thus total time increase linearly with the database size.

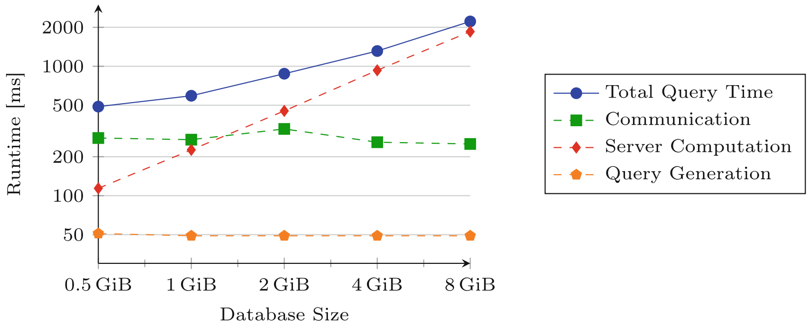


**Fig. 7.** Varying DB size: Runtimes for varying database size, $k = 3$ servers, redundancy parameter $r = 2$ for querying one large file (8.5 MiB) and block size $b = 256$ kiB. All queries are multi-block (MB) queries using four Russians precomputation and uniformly distributed database blocks. Note the logarithmic scale of the y-axis.

## 3 OnionPIR

As an application for our previously presented improvements, we introduce a private communication system called *OnionPIR* for anonymous communication. We use RAID-PIR with our optimizations as a building block for public key distribution, and onion routing to get a system efficient enough for practical use.

### 3.1 Motivation

When two parties register under pseudonymous identities to a classical messaging service and connect to the service by using Tor [DMS04], a malicious server can link those two pseudonyms together. Even if these pseudonyms do not reveal 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 to reveal information about users. Therefore, Tor alone is not enough to provide protection against metadata leakage. Additionally, it has turned out that the assumption of users willing to establish a shared secret is problematic. 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 System Model and Goals

Existing privacy-preserving communication systems do often not scale for large numbers of users or require the exchange of a shared secret out-of-band, which is error-prone and leads to usability issues most users are not willing to accept. The success of the popular messaging app Signal[2] 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 an in-depth knowledge of cryptography, it is necessary to build technologies that provide privacy and usability by design and give reasonable defaults for its users.

To combine the strong privacy guarantees of PIR protocols with the efficiency of onion routing protocols, such as Tor [DMS04], the interaction with the server is divided into two phases: an *initialization phase* where the communication channels are established and keys are exchanged via PIR and a *communication phase* where the actual message exchange takes place through Tor.

In the initialization phase, PIR is used to *privately* exchange information between two parties which want to communicate securely. The exchange happens in a way that no one except these two parties will find out that the communication between them happened. This procedure could theoretically be used to exchange all kinds of data. However, when it comes to practical applicability, querying large amounts of information via PIR does not scale well for a larger number of users. Hence, in the communication phase no PIR techniques will be used.

Instead, the information retrieved from PIR is then used to place messages, encrypted using Authenticated Encryption with Associated Data (AEAD), in an anonymous inbox, called "dead drop". This dead drop can only be read and written at the OnionPIR control server in the communication phase by clients that know its identifier. By using onion routing to hide the identity of the communication partners, the server is not able to determine who sent and received messages. For real-time communication, the users could also establish direct connections using TCP streams or web sockets [FM11] through the server.

The OnionPIR system, depicted in Fig. 8, serves clients who want to communicate with each other and two types of servers. All honest clients form the anonymity set among which a user is anonymous. That means that a potential

[2] downloaded by 1–5 Mio. users according to the Google Play Store, https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms.

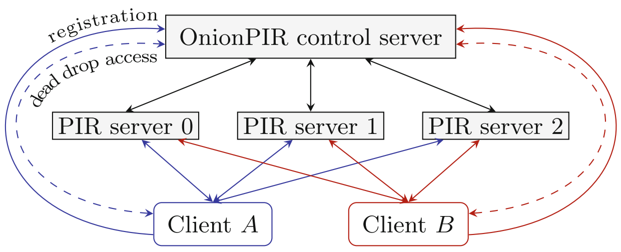


**Fig. 8.** OnionPIR system. The OnionPIR control server handles user registrations, distributes users' public keys to the PIR servers and holds the dead drops. Clients perform PIR queries to privately receive other users' public keys from the PIR servers. Clients connect to the OnionPIR control server directly or via Tor (dashed lines).

adversary cannot determine which users within this set communicate with each other. The central OnionPIR control server handles user registration and serves as a content provider for the PIR servers. It also acts as a database for the "dead drops" in the communication phase. The PIR servers are used only in the initialization phase to privately distribute the public keys of clients.

## 3.3 Protocol Description

In this section we describe the OnionPIR protocol. A (simplified) version is depicted in Fig. 9. When a client $A$ registers for the service, it first runs through an account verification process and sends its public key to the OnionPIR control server which will later distribute it to the PIR servers. Another client $B$, that has an address book entry for $A$ (e.g., a mobile number or e-mail address), will later *privately* query $A$'s public key via PIR. In addition, each user periodically queries for his or her own public key to make sure the OnionPIR control server does not distribute bad keys to the PIR servers (see Sect. 3.4 for details). Note, that a single PIR server is never able to replace keys unless it colludes with other PIR servers because the responses of multiple servers will be combined to

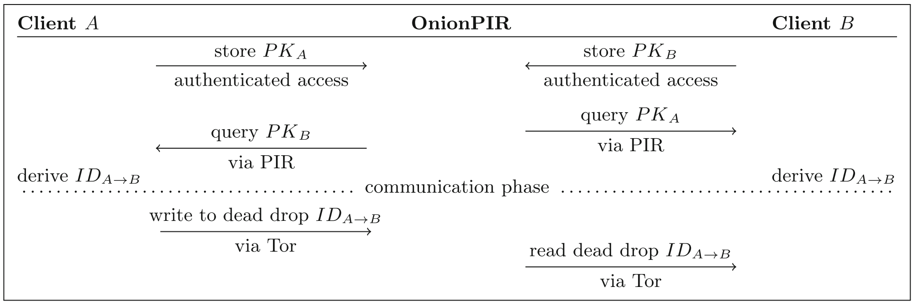


**Fig. 9.** Simplified OnionPIR protocol. The registration, dead drop and PIR servers were abstracted into one server. Key renewal and the answer of client $B$ are not shown.

retrieve the public key. $B$ 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 agreement. 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 $B$.

**Symmetric Keys.** The shared secret $K_{AB}$ and both parties' public keys are used to derive identifiers of the dead drops and keys used to exchange messages. Client $A$ generates two different symmetric keys derived from the initial shared secret $K_{AB}$ and the respective public keys by a keyed-hash message authentication code (HMAC): $K_{A \to B} = \text{HMAC}_{K_{AB}}(pk_B)$ for sending messages to $B$ and $K_{B \to A} = \text{HMAC}_{K_{AB}}(pk_A)$ for receiving messages from $B$. These secrets constantly get replaced by new ones, transmitted alongside every message to provide forward secrecy.

**Dead Drop IDs.** The identifiers of the dead drop for sending from $A$ to $B$ is built by computing an HMAC with the key $K_{A \to B}$ of a nonce $N_{A \to B}$ that increases after a given time period. Hence, the identifier $ID_{A \to B} = \text{HMAC}_{K_{A \to B}}(N_{A \to B})$ changes in a fixed interval, even if no messages were exchanged at all. This prevents 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[3]. 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 key $K_{A \to B}$ is used to generate a point in time at which the nonce $N_{A \to B}$ will be increased. This leads to a different update time for all identifiers of dead drops.

**Sending Messages.** When a client $A$ wants to send a message $m$ to $B$, it encrypts the message using Authenticated Encryption with Associated Data (AEAD) using $K_{A \to B}$ so that only $B$ can read it. Note, that the ciphertext of the message $m$ does not reveal any information about the sender or the receiver as explained in [Ber09, Sect. 9]. $A$ then stores the encrypted message in the dead drop $ID_{A \to B}$, which $A$ accesses via Tor. $B$ is now able to fetch and decrypt $A$'s message from this dead drop. The shared secret will only be replaced by a new one transmitted alongside a message when $B$ acknowledges the retrieval of the

[3] The need for 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).

new secret in a dead drop derived from the *existing* shared secret. Until then, $A$ will not store any messages in a dead drop derived from the new secret and will retransmit messages until $B$ sends the acknowledgment. This procedure will guarantee that no messages will be lost. Note, that $ID_{A \to B}$ may still change over time because of the used nonce $N_{A \to B}$ which is known by both parties.

Querying for new public keys using PIR is 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 random point in time between the current query and the next one derived from the shared key $K_{A \to B}$. This ensures that the server cannot correlate the dead drop access to the PIR request because it could also have been initiated by any other clients that did a PIR request in the fixed interval. 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.

**Initial Contact.** If $B$ wants to contact $A$ without $A$ knowing about that (and thus not checking the associated dead drop at $ID_{B \to A}$), an anonymous signaling mechanism is required. We propose a per-user fixed and public dead drop that *all* other users write to, to establish contact. The fixed dead drop ID is $ID_A = \mathrm{HMAC}_0(pk_A)$. Messages into this dead drop are encrypted using hybrid encryption, similar to PGP, where $A$'s public key $pk_A$ is used to encrypt an ephemeral symmetric key, which encrypts the request message. This reveals how many contact requests $A$ receives, which we consider as non-critical. However, this number could be obscured by sending dummy requests.

### 3.4 Analysis

**Correctness.** Correctness of RAID-PIR is shown in [DHS14], correctness of Tor is explained in [DMS04] and has already been well-proven in practice. Messages are acknowledged and retransmitted if identifiers change 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 shown in [Ber09]. All operations involved in the private establishment of identifiers for the dead drops are deterministic and therefore result in the same identifiers for both parties.

**Security.** OnionPIR's security is based on the security guarantees of the underlying protocols and their assumptions. First, we assume that RAID-PIR is secure and does not leak any metadata about the queried information. A security argumentation for this is given in [DHS14]. In particular, it is important that the PIR servers are run by different non-colluding operators. Note, that the security guarantees are still fulfilled if less than $r$ servers collude, where $r$ is the redundancy parameter. 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, we assume 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 user traffic and colludes with the operator of the PIR servers or gains unauthorized access to them. Note, that it is necessary to at least de-anonymize two specific Tor connections in order to learn if two users are communicating with each other. Therefore, an attacker would have to be able to de-anonymize all users of the service to gain the full social graph of a given user. OnionPIR does not put any effort in hiding the fact that a user is using the service at all.

Anonymity is guaranteed among all honest users. A malicious user that announces its list of requests to the dead drops, would effectively remove himself from the anonymity set. Note, that no user is able to gain information about communication channels it is not participating in. In addition, no user is able to prove that a communication took place because the used AEAD guarantees repudiability.

OnionPIR relies on a Trust On First Use (TOFU) strategy to lessen the burden of manually exchanging keys. For the purpose of detecting the distribution of faulty keys, a client queries not only for the public keys of its contacts, but also for its own public key. This query can be performed at nearly no cost when querying together with other contacts using RAID-PIR's multi-block query. Since the PIR servers are not able to determine which PIR blocks are being requested, they are not able to manipulate the resulting response in a meaningful way, unless they collude. Of course, additional security can be achieved by adding out-of-band key verification, e.g., by announcing public keys on personal websites. Another interesting option are plausibility checks for the updates of the PIR database in the PIR servers and thereby extending the existing anytrust model.

Access to the dead drops is not protected against any type of manipulation by third parties since identifiers are only known to the involved parties. 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 and would require a federated or decentralized system.

Protection against malicious clients trying to flood the dead drops with large amounts of data could be achieved by making use of blind signatures [Cha83]. 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 cannot identify the client who generated them. Since a server only signs a fixed number of tokens in a given time interval per client, this approach rate-limits write requests to the database.

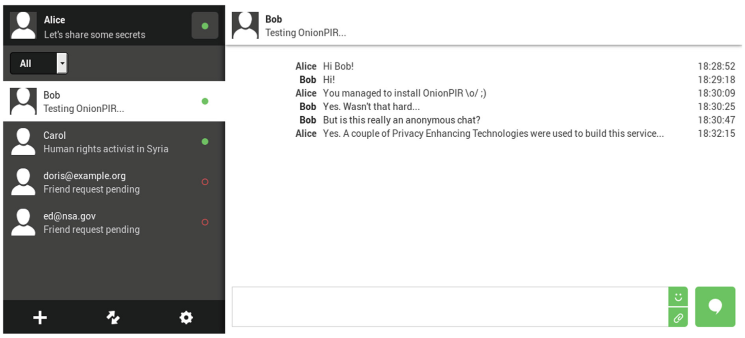
**Complexity and Efficiency.** OnionPIR aims at providing efficient anonymous communication. Many existing systems (see Sect. 4) require a high communication overhead or high computational costs. The dead drop database is therefore combined with scalable onion routing and can be implemented as simple key-value storage. The servers needed in the communication phase can be deployed with low operating costs, comparable to traditional communication services.

The initialization phase in which the PIR requests are performed is crucial for the scalability of the system. As shown in Sect. 2.5, PIR is a valid choice that offers reasonable performance and allows users to detect malicious actions of the servers. The 3.8 GiB PIR database used for the benchmarks in Sect. 2.5 is sufficient to store public keys of 127 Mio. users, using 256 bit elliptic curve public keys.

The database size and the server's computation will grow linearly in the number of users. The PIR servers' ingress traffic for PIR queries depends on the number of blocks $B$ in the database and therefore also scales linearly. Thanks to the PRG used in [DHS14], the size of a PIR query is $\lceil B/8 \rceil$ Bytes for all servers combined (excluding PRG seeds and overhead for lower level transport protocols). The egress bandwidth is constant since the size of the response only depends on the block size $b$ (and the number of chunks for multi-block queries).

## 3.5 Implementation

We implemented a desktop application for secure messaging with metadata protection based on OnionPIR. Our implementation is written in Python and C, using our version of RAID-PIR including the optimizations from Sect. 2 for PIR, the Networking and Cryptography library (NaCl) [BLS12] for cryptographic operations, and Stem[4] as controller library for Tor. The system is divided into a client, the OnionPIR control server and PIR servers. The client offers a GUI as depicted in Fig. 10. Our open-source implementation is publicly available.[5]



Alice Hi Bob! 18:28:52
Bob Hi! 18:29:18
Alice You managed to install OnionPIR \o/ ;) 18:30:09
Bob Yes. Wasn't that hard... 18:30:25
Bob But is this really an anonymous chat? 18:30:47
Alice Yes. A couple of Privacy Enhancing Technologies were used to build this service... 18:32:15

**Fig. 10.** Screenshot of the OnionPIR client GUI.

[4] https://stem.torproject.org/.
[5] https://github.com/encryptogroup/onionPIR.

## 4 Related Work

Nowadays, end-to-end encryption is available and deployed at large scale. In the past, these technologies were not accessible to a large user base because they required expert knowledge or were just not convenient, and thus only used by enthusiasts. For example, users of OpenPGP [CDF+07] have to manually build a web of trust and should be familiar with public key cryptography, while S/MIME [Ram99] requires certificate management. When the Signal Protocol was integrated into popular messaging services like WhatsApp[6] or Facebook Messenger[7] in 2016, private messaging became available to an extremely large user base. However, protecting not only communication content but also its metadata are still under active research. OnionPIR currently relies on RAID-PIR [DHS14] for public key distribution, but could also employ different PIR schemes. PIR is an active research area and there are other viable candidates [DG14, Hen16, AMBFK16]. Similarly, we rely on Tor to provide anonymity, which can also be achieved by using alternative techniques such as mixing networks [Cha81].

Next, we present system proposals that are related to OnionPIR. *Redphone* was one of the first applications that tackled metadata leakage using Bloom filters [Blo70] for private contact discovery. Redphone's encrypted call features were integrated into Textsecure/Signal, however, without anonymity due to scalability [Ope14]. With *DP5* [BDG15] users can anonymously exchange online presence information based on PIR. DP5 divides time in long-term and short-term epochs which are in the order of days and minutes respectively, which makes it impractical for real-time communication. Users must share symmetric keys before the protocol run, which can be hard to achieve in practice. Recently, *Alpenhorn* [LZ16] was proposed, which is based on identity-based encryption (IBE) for key distribution, a mix network [CJK+16, Cha81] for privacy and a so called keywheel construct for forward secrecy. Alpenhorn was integrated into Vuvuzela [vdHLZZ15], which supports 10 Mio. users using three Alpenhorn servers with an average dial latency of 150 s and a client bandwidth overhead of 3.7 KiB/s. *Riposte* [CGBM15] enables a large user base to privately post public messages to a message board. Its security is based on distributed point functions that are evaluated by a set of non-colluding servers. Time is divided into epochs, in which all users who post messages form an anonymity set. *Ricochet* (https://ricochet.im) is a decentralized private messaging service based on Tor hidden services, whose addresses must be exchanged out-of-band to establish connections. Recent research showed that HSDirs are used to track users [SN16], which might be problematic for Ricochet's privacy. *Riffle* [KLDF16] provides scalable low-latency and low-bandwidth communication through mix networks [Cha81] using verifiable shuffles [BG12] for sender anonymity and PIR for receiver anonymity. In Riffle, time is divided into epochs and each client sends and receives messages

[6] https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.

[7] https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf.

even if they do not communicate. *The Pynchon Gate* [SC05] is an anonymous mail system that guarantees only receiver anonymity by using PIR.

**Acknowledgments.** We thank the anonymous reviewers for their valuable feedback on our paper. This work has been co-funded by the German Federal Ministry of Education and Research (BMBF) and by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, and by the DFG as part of project S5 within the CRC 1119 CROSSING.

## References

[ABH10] Albrecht, M., Bard, G., Hart, W.: Efficient multiplication of dense matrices over GF(2). ACM Trans. Math. Softw. **37**, 9:1–9:14 (2010)

[ADKF70] Arlazarov, V., Dinic, E., Kronrod, M., Faradzev, I.: On economical construction of the transitive closure of a directed graph. USSR Acad. Sci. **134**, 1209–1210 (1970)

[AMBFK16] Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.-O.: XPIR: private information retrieval for everyone. In: Privacy Enhancing Technologies Symposium (PETS 2016), no. 2, pp. 155–174 (2016)

[BDG15] Borisov, N., Danezis, G., Goldberg, I.: DP5: a private presence service. In: Privacy Enhancing Technologies Symposium (PETS 2015), no. 2, pp. 4–24 (2015)

[Ber09] Bernstein, D.J.: Cryptography in NaCl (2009). https://cr.yp.to/highspeed/naclcrypto-20090310.pdf

[BG12] Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29011-4_17

[Blo70] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)

[BLS12] Bernstein, D.J., Lange, T., Schwabe, P.: The security impact of a new cryptographic library. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 159–176. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33481-8_9

[BV14] Budurushi, J., Volkamer, M.: Feasibility analysis of various electronic voting systems for complex elections. In: International Conference for E-Democracy and Open Government 2014 (2014)

[CDF+07] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP message format. RFC 4880, RFC Editor, November 2007. http://www.rfc-editor.org/rfc/rfc4880.txt

[CGBM15] Corrigan-Gibbs, H., Boneh, D., Maziàres, D.: Riposte: an anonymous messaging system handling millions of users. In: IEEE Symposium on Security and Privacy (S&P 2015), pp. 321–338 (2015)

[CGN98] Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords. IACR Cryptology ePrint Archive, Report 1998/003 (1998). http://eprint.iacr.org/1998/003

[Cha81] Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**, 84–90 (1981)

[Cha83] Chaum, D.: Blind signature systems. In: Advances in Cryptology - CRYPTO 1983, p. 153 (1983)

[CJK+16] Chaum, D., Javani, F., Kate, A., Krasnova, A., de Ruiter, J., Sherman, A.T., Das, D.: cMix: anonymization by high-performance scalable mixing. IACR Cryptology ePrint Archive, Report 2016/008 (2016). http://eprint.iacr.org/2016/008

[CKGS95] Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. In: Foundations of Computer Science (FOCS 1995), pp. 41–50 (1995)

[DG14] Devet, C., Goldberg, I.: The best of both worlds: combining information-theoretic and computational PIR for communication efficiency. In: Cristofaro, E., Murdoch, S.J. (eds.) PETS 2014. LNCS, vol. 8555, pp. 63–82. Springer, Cham (2014). doi:10.1007/978-3-319-08506-7_4

[DHS14] Demmler, D., Herzberg, A., Schneider, T.: RAID-PIR: practical multi-server PIR. In: ACM Cloud Computing Security Workshop (CCSW 2014), pp. 45–56 (2014)

[DMS04] Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: USENIX Security Symposium 2004, p. 21 (2004)

[DSZ14] Demmler, D., Schneider, T., Zohner, M.: Ad-hoc secure two-party computation on mobile devices using hardware tokens. In: USENIX Security Symposium 2014, pp. 893–908 (2014)

[DSZ16] Dowling, B., Stebila, D., Zaverucha, G.: Authenticated network time synchronization. In: USENIX Security Symposium 2016, pp. 823–840 (2016)

[FM11] Fette, I., Melnikov, A.: The websocket protocol. RFC 6455, RFC Editor, December 2011. http://www.rfc-editor.org/rfc/rfc6455.txt

[HCE11] Huang, Y., Chapman, P., Evans, D.: Privacy-preserving applications on smartphones. In: USENIX Workshop on Hot Topics in Security (HotSec 2011), p. 4 (2011)

[Hen16] Henry, R.: Polynomial batch codes for efficient IT-PIR. In: Privacy Enhancing Technologies Symposium (PETS 2016), pp. 202–218 (2016)

[KLDF16] Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: an efficient communication system with strong anonymity. In: Privacy Enhancing Technologies Symposium (PETS 2016), pp. 115–134 (2016)

[KO97] Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: Foundations of Computer Science (FOCS 1997), pp. 364–373 (1997)

[Lan15] Landau, S.: Mining the metadata: and its consequences. In: International Conference on Software Engineering (ICSE 2015), pp. 4–5 (2015)

[LG15] Lueks, W., Goldberg, I.: Sublinear scaling for multi-client private information retrieval. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 168–186. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47854-7_10

[LZ16] Lazar, D., Zeldovich, N.: Alpenhorn: bootstrapping secure communication without leaking metadata. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016), pp. 571–586 (2016)

[MCA06] Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The new Casper: query processing for location services without compromising privacy. In: International Conference on Very Large Data Bases (VLDB 2006), pp. 763–774 (2006)

[MMM16] Mayer, J., Mutchler, P., Mitchell, J.C.: Evaluating the privacy properties of telephone metadata. Natl. Acad. Sci. **113**(20), 5536–5541 (2016)

[MOT+11] Mittal, P., Olumofin, F., Troncoso, C., Borisov, N., Goldberg, I.: PIR-tor: scalable anonymous communication using private information retrieval. In: USENIX Security Symposium 2011, p. 31 (2011)

[Ope14] Open Whisper Systems. The difficulty of private contact discovery (2014). https://whispersystems.org/blog/contact-discovery/

[Ram99] Ramsdell, B.: S/MIME version 3 message specification. RFC 2633, RFC Editor, June 1999. http://www.rfc-editor.org/rfc/rfc2633.txt

[SC05] Sassaman, L., Cohen, B., Gate, T.P.: A secure method of pseudonymous mail retrieval. In: Workshop on Privacy in the Electronic Society (WPES 2005), pp. 1–9 (2005)

[SN16] Sanatinia, A., Noubir, G.: HOnions: towards detection and identification of misbehaving tor HSDirs. In: Hot Topics in Privacy Enhancing Technologies Symposium (HotPETS 2016) (2016)

[vdHLZZ15] van den Hooff, J., Lazar, D., Zaharia, M., Zeldovich, N.: Vuvuzela: scalable private messaging resistant to traffic analysis. In: Symposium on Operating Systems Principles (SOSP 2015), pp. 137–152 (2015)