Mix-ORAM: Using delegated shuffles.

Raphael R. Toledo
University College London
United Kingdom

George Danezis
University College London
United Kingdom

Isao Echizen
National Institute of Informatics
Japan

ABSTRACT

Oblivious RAM (ORAM) is a key technology for providing private storage and querying on untrusted machines but is commonly seen as impractical due to the high overhead of the re-randomization, called the eviction, the client incurs. We propose in this work to securely delegate the eviction to semi-trusted third parties to enable any client to accede the ORAM technology and present four different designs inspired by mix-net technologies with reasonable periodic costs.

1 INTRODUCTION

Thanks to cloud technologies, people have been able to seamlessly store impressive amounts of data on remote servers. Besides accessibility, availability and integrity, the storage providers also have to ensure to their clients the data's and the meta data's confidentiality and secure them from not only external adversaries but also from the cloud itself. They thus employ cryptographic mechanisms to protect the communication channels, such as user authentication, data encryption and integrity checking. These, however, do not prevent the leakage of all meta data: the servers can monitor user activities and watch which records are accessed.

Oblivious RAM (ORAM) [15], or Oblivious Storage [7], precisely prevents an adversary from observing the record access. In these schemes, the records are first locally encrypted and permuted in a new order before being uploaded to the untrusted cloud storage. When the user seeks a given record, the local client computes the corresponding remote index, fetches the encrypted data block and decrypts it. After a number of accesses, the database is randomized locally by the client to bring to naught any leaked information from the accesses: this is the "eviction process".

This eviction is the main bottleneck of ORAM. Indeed, the eviction consists in randomizing the whole database by permuting the records and refreshing their encryption so that an adversary loses any insight on the correspondence between the remote and local, or virtual and real, record indices. However, as we assume the number of records stored remotely to be orders of magnitude higher than what the client can store, the client has to download and randomize the database during the eviction in chunks, and do so several times so that all record ordering is equally likely. Thus as the database size grows, the eviction cost rises super linearly.

This is why we propose in this work to delegate the eviction process to dedicated semi-trusted parties. Doing so, light-weight clients could accede the ORAM technology and thanks to the use of mix networks [8] inspired designs, ORAM would become more portable.

In this work, we present several privacy friendly distributed systems inspired by mix-nets to safely delegate ORAM's randomization process to semi-trusted third parties. Their advantages include the reduction of the client computation, the possibility to delay the eviction to quieter times, the guaranteed database availability during the eviction process regardless of the ORAM design and the independence from centralized parties. However careful design is required to make them scalable. Our contributions are as follows:

- We introduce and motivate the use of mix-net to construct delegated ORAM eviction schemes, letting very thin clients use most ORAM designs.
- We present a number of eviction schemes relying on mix-net, improve them with load balancing via parallel mixing.
- We finally evaluate the performance of our delegated eviction designs, compare them between each other and with regular eviction schemes.

We first present the related work in Section 2. We then introduce the ORAM model and how our model differ, its associated threat model and explain the different costs in Section 3. We then present two simple designs over a cascade mix-net and optimize them using random transposition shuffles over a stratified mix-net in Section 4. We then hand out our security arguments in Section 5, evaluate and discuss our designs in Section 6 and Section 7 before concluding.

2 RELATED WORK

ORAM. ORAM was first presented by Goldreich and Ostrovsky in 1990 [29] to prevent reverse engineering and protect software running on tamper resistant CPUs. The model was also formally extended in 2011 [7] to data protection on untrusted remote clouds and in 2015 some designs were evaluated on Amazon Simple Storage Service (S3) [**?**]. Since its introduction, ORAM enhancements have been proposed including *data structures* diversification [16, 32–34], the use of more and more sophisticated *security definitions* with statistical security [1, 10] and differential privacy [35], and the revision of *item lookups* with cuckoo hashing [31] and bloom filters [37]. Most ORAM constructions are based on a single client-server model, but multi-user designs were gradually introduced [4, 14, 20].

Shuffling and Sorting. Shuffle and sorting algorithms are a thoroughly researched subject central to ORAM for the randomization process. However most of the existing methods are not useful for ORAM as they are not oblivious in that the permutations done depends on the data itself. Examples of oblivious sorting algorithms include sorting networks such as Batcher's [5] and the ones based

This work is supported by H2020 PANORAMIX Grant (ref. 653497) and EPSRC Grant EP/M013286/1; and Toledo by Microsoft Research.
Conference'17, July 2017, Washington, DC, USA
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

on AKS [2], which unfortunately were proven to be impractical because of the high number of I/Os, but also more recent and efficient ones [30]. Newer designs include the randomized Shellsort [17], an elegant simple data-oblivious version of the Shellsort algorithm, the Zig Zag sort [18] presented in 2014, the Melbourne shuffle [28] and work of particular interest written by Goodrich in 2012 [19] assess the information leakage due the use of a partially compromised parallel mix-net.

Mix-nets. Mix-nets were first presented for anonymous e-mailing by David Chaum in 1981 [8]. As they became popular many improvements were made over the years [11–13, 27]. Mix-nets' main goal is to give users anonymity by hiding the correspondence between the incoming users' packets and the mix-nets output. To do so, the users' messages go through several mixes which permute them and refresh their encryption. Either re-encryption [36] and onion encryption can be used, proofs of shuffle [6, 21, 22] and Randomized Partial Checking [23] can help verify the shuffle correctness.

This work is inspired by the mix-net technology for its encryption and permutation functionalities, however, only the packet unlinkability property is of interest for ORAM. From now on, we refer traditional ORAM solutions as ORAM and our designs as Mix-ORAM.

3 PRELIMINARIES

3.1 ORAM introduction

The Oblivious RAM system is a distributed system composed of two parties, the ORAM *server* and the *client*. The *server* handles two data arrays, a first one we call the *database* which comprises the user's encrypted records and a temporary one that we call the *cache* which is of lesser size and used to hide the number of times a record was accessed. Only read and write operations are allowed on these arrays. The *client* comprises a small memory in which the cache and some additional records can fit and provides two main methods, the ORAM records access and eviction.

As stated in Bindschaedler's work [**?**], most ORAM algorithms can be classified in four distinct families, the layered ORAM, the partition-based ORAM, the large-message ORAM and the tree-based ORAM, depending on the database structure and the eviction method employed. In this work we consider all algorithms that rely on periodic data-oblivious shuffle of the database. This excludes from the scope of our study solutions relying on the tree-based architecture or using higher client memory and the use of recursive algorithms such as [33**?**] or as Path ORAM [34] which uses $O(\log n)$ private memory and $O\left((\log n)^2\right)$ access overhead. We can now thus describe the client methods as follows.

The access method: In order to perform a read or a write operation, the client first downloads if needs be the cache and checks locally whether the desired record is present. If so, a dummy record is fetched from the database, else the client fetches the desired record. The cache is then updated with a newly encrypted version of the fetched element before finally being uploaded back to the remote ORAM server.

The eviction method: When the cache is full, the client starts the eviction process to prevent too important information leakage. The eviction consists of two parts where the client first *rebuilds* the database before starting the *oblivious shuffle*.

During the *rebuild phase*, the client obliviously uploads back the records from the cache back to the ORAM database. After doing so, the client can finally starts *the oblivious shuffle* during which chunks of the database are permuted and encrypted obliviously before being sent back to the database.

Current ORAM solutions have so far relied on the client locally encrypting and shuffling the records in an oblivious manner. Batcher's sorting network [5] for instance requires $O\left(n(\log n)^2\right)$ I/Os, AKS [2] or Zig-zag sorting networks [18] which use $O(n \log n)$ I/Os but with large constant factors or finally the Melbourne Shuffle [28] which is not not based on a data-oblivious sorting algorithm, using only $O\left(\sqrt{n}\right)$ I/Os but with a large and fixed message size of $O\left(\sqrt{n}\right)$. We propose in this paper a new oblivious shuffle performed by semi-trusted third parties, the difficulty of which being that the records must be shuffled in a scalable way without leaking information about the correspondence between indices to any party.

In this paper, we will reuse the previously defined system with the addition of the mix-net, a group of independent servers capable of encryption and permutation, and the following modification of the oblivious shuffle. When starting the delegated eviction, the client first selects a set of mixes which will randomize the database. It then generates and sends to them randomization instructions. The mixes use these to compute the encryption keys and permutation seeds and fetch their allocated records from the database. They then randomize the records by encrypting and shuffling them with the keys and seeds, and forward them to the next mix(es) in what we call a round. This randomization process is then done a number of times as specified in the instructions before the records are uploaded back to the database ready to be accessed.

3.2 Security definitions and Threat model

We presume here the existence of a motivated adversary trying to subvert a target user's privacy by learning the correspondence between the remote and the local record indices. We furthermore assume that the user protects its data with an ORAM system compliant with the Privacy Definition 1 introduced by Stefanov et al. [33] (see below) and additionally that all communications between the client, ORAM server and mixes are secured but may be intercepted as in the *global passive adversary* assumption. Finally, we suppose the adversary has corrupted the ORAM server and all but one mixes, and that the compromised machines behave in a *honest but curious* way in that all operation are correctly performed but passively recorded and all secrets shared with the adversary.

Privacy Definition 1. *Let's denote a sequence of k queries by $seq_k = \{(op_1, ad_1, data_1), \ldots, (op_k, ad_k, data_k)\}$, where op denotes a read or write operation, ad the address where to process the operation and data the block to write if needs be else $\perp$. We denote by $ORAM(seq_k)$ the resulting randomized data access from the ORAM*

process with input seq_k. The ORAM guarantees that $ORAM(seq_k)$ and $ORAM(seq'_{k'})$ are computationally indistinguishable if $k = k'$.

This work focuses on the ORAM eviction process and more precisely on the oblivious shuffle problem where sequences of data-blocks are shuffled and encrypted in order to hide the correspondance between records indices after a number of accesses has been performed. This problem refers to the eviction of the shelter in the database in the Square Root solution [29] and to the eviction of upper partitions in a lower ones in the Hierarchical case [16]. We consider the threat of a probabilistic polynomial-time (PPT) adversary and evaluate the security of our designs by looking at the information leakage of the oblivious shuffle and at the correctness of the cryptography methods used by the designs.

3.3 Cryptographic Primitives

PRG & Seeds. ORAM systems use pseudo random generators (PRG) and seeds to link remote and real indices. A distribution $\mathcal{D}$ over strings of length l is said pseudo random if $\mathcal{D}$ is indistinguishable from the uniform distribution over strings of length l [24]. That means it is infeasible for any probabilistic polynomial-time adversary to tell whether the string was sampled accordingly to D or was chosen uniformly at random. A PRG is a deterministic algorithm that receives as an input a short random key and stretches it into a long pseudo random stream.

Encryption. ORAM designs heavily rely on encryption to obfuscate the records during the eviction and the access. In this work we will use symmetric encryption for its rapidity and also public encryption for the key and seeds derivation. The Advanced Encryption Standard (AES) [9] has high speed and low RAM requirements: it has throughput over 700 MB/s per thread on recent CPUs such as the Intel Core i3 [26] which makes it the ideal choice for ORAM. We also make use of elements of a elliptic curve group of prime order satisfying the decisional Diffie-Hellman assumption to compress the instructions sent to the mixes.

3.4 Model

System. We consider in this work an ORAM remote server consisting of a database with memory of n b-bit long data blocks and a cache with memory of s, $s \ll n$, b-bit long data blocks. We furthermore consider a mix-net composed of m mixes, and a client with memory of s data blocks. The ORAM server, the mixes and the client additionally have a small memory of capacity $O(m)$ to store extra information about the permutation and encryption. We consider facing the threat of a PPT adversary and call κ our security parameter representing the length of our encryption keys and permutation seeds that we denote by k and σ respectively.

Costs. We are interested on one side in the costs incurred by the client for recovering a record index, for decrypting a record and the extra space needed, on the other side in the total costs incurred by the mixes encrypting the records, permuting them and the transferring them. Some operations can be preprocessed by the mixes while the records are being transferred, as the key and seeds generation and the record allocation, and as thus will not be the main focus.

4 MIX-ORAM

This work aims at obviously sorting the database from an old state $\Pi_\sigma(DB)$ to a new one $\Pi_{\sigma'}(DB)$ with the aid of a mix-net. As the seed space is not structured, it is a NP-hard problem to find for any σ_1 another seed σ_2 such that $\Pi_{\sigma_2} \circ \Pi_{\sigma_1} = I$, the overall mix-net is limited to perform a permutation $\Pi_{\sigma''}$ such that $\Pi_{\sigma'}(DB) = \Pi_{\sigma''} \circ \Pi_\sigma(DB)$. We present two ways to do the oblivious shuffle. We call the first way the *Layered method* which consists in having the mixes permute the records with independent random seeds, i.e. the permutation layers are simply stacked, and the client storing the indices. We call the second way the *Rebuild method* where the mixes obliviously undo the permutations done at the previous done, Π_σ, before shuffling the records with new random seeds.

In this section, we first introduce the two methods to randomize the records during the eviction over a simple cascade mix-net. We then optimize the Mix-ORAM schemes by considering a stratified mix-net together with distributed shuffle algorithms.

4.1 A simple Mix-ORAM

We introduce here the two randomization methods over a semi-trusted cascade mix-net, a topology in which the mixes receive and process a batch of packets sequentially has shown in Figure 1. For each method, we show how the mix-net encrypts and permutes the records and how the client recovers a record plain text.

4.1.1 Cascade Layered scheme. In this scheme, we use the the layered encryption method over the cascade mix-net for the eviction of the database. The underlying principle of the layered method is to have the whole database go through the mix-net once, with each mix independently encrypting and permuting the records. This method corresponds to the sole Wrapping phase (3) of Figure 1.

We assume in the Layered method that the records were preprocessed before being uploaded to the ORAM server as follows. Each record is first appended with its current index used as label and an initialization (IV) token as shown in Figure 1. The resulting data structure is then encrypted in two stages with AES in CBC mode. The label and record are first encrypted together using the IV token as initialization vector then the IV token is encrypted with the first bits of the label-record cipher. All the data structures are then permuted and finally uploaded to the ORAM server, while their indices are locally stored on the client.

IV token	label || record
$8 \cdot \lceil \log(n)/8 \rceil$ bit	$8 \cdot \lceil \log(n)/8 \rceil + b$ bit

Table 1: Layered method data structure.

In the following, we present the *Mix instructions* sent to the mix-net, used for retrieving the records from the ORAM database and compute the permutation seeds and encryption keys and the *Mix*

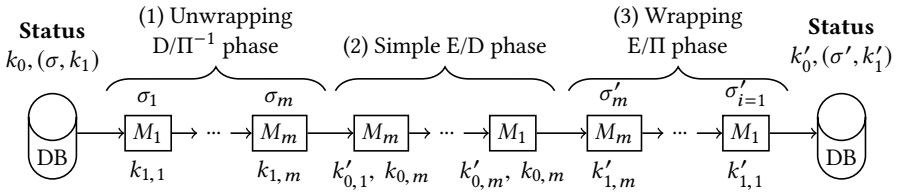

Figure 1: Cascade Mix-ORAM.
Rebuild method (all phases) and Layered method (only the Wrapping phase)

operations. The client decryption and access methods are then detailed in the *Client operations*.

Mix instructions. To start the eviction, the client sends to each mix M_i the ordered list of mixes $list = (ports,\ ips)$ involved in the oblivious shuffle, the database access information db, the security parameter κ, and α_i, an element of a cyclic group of prime order satisfying the decisional Diffie-Hellman Assumption.

$$C \rightarrow M_i\ :\ db,\ \alpha_i,\ \kappa,\ list$$

Let g be a generator of the prime-order cyclic group $\mathcal{G}$ satisfying the Diffie-Hellman Assumption and q the prime order of $\mathcal{G}$. We assume that each mix M_i has a public key $y_i = g^{x_i} \in \mathcal{G}^*$ with $x_i \in_{\mathbb{R}} \mathbb{Z}_q$ being its private key. We also assume that the list of (mix_i, y_i) is distributed in a authenticated way thanks to a Public Key Infrastructure (PKI). To generate the αs, the client pick at random in $\mathbb{Z}_q$ for each mix M_i the element z_i. The group elements and mixes' private keys are used to generate the shared secrets ss from which the encryption keys and permutation seeds are derived with the aid of the HKDF derivation function [25] as follows:

$$\alpha_i = g^{z_i}\ ;\ ss_i = y_i^{z_i} = \alpha_i^{x_i}\ ;\ k_i,\ \sigma_i = \text{hkdf}(ss_i,\ \kappa)$$

Mix operations. The mix M_i first decrypts the Mix instructions, generates the encryption key k_i and the permutation seed σ_i. The mix then receives the records from the mix M_{i-1} or fetch them if it is the first mix in the list. It then encrypts all the data structures with the new encryption key k'_i as said previously and permutes them with the new seed σ'_i before sending it to M_{i+1} or the database if M_i is the last mix in the list.

Client Operations. The client can find the record index locally as it was stored previously. To decrypt a record, the client uses a trial and error recursive algorithm : the client first decrypts the data-block successively with all the shared secrets set by the latest Mix instructions and decrypts it another time with its private key. If the label is the record index, the process stops and the record is returned. If not, the data-block is re-encrypted with the client's private key and the algorithm restarts with the newly encrypted record and the shared secrets used in older Mix instructions.

We moreover modify the *Access method* to prevent timing attacks as follows. When accessing a record, the client now directly encrypts the data-block with its own private key and updates it in the local cache. The client then uploads the cache to the remote server. After doing so, the client can perform the read/write operation: it either overwrites the record with its new version ; either decrypts the record with its private key and then starts the trial and error algorithm. Once the plain text is retrieved, the client finally encrypts the record a last time with its private key and stores it locally and at the next eviction overwrites the cached version with it.

Costs. As the whole database is sent through the mix-net, the mix communication cost is $(m+1) \cdot n \cdot b$, the mix permutation cost is $mnC_\Pi(n)$ with $C_\Pi(n)$ the cost of permuting n elements and the encryption cost mnC_{cbc} with C_{cbc} the cost of encrypting one data block. The client Lookup cost is of the order $O(1)$ thanks to the n indices stored locally for a total of $n\log(n)$ bits. We will discuss of the decryption cost in the Evaluation Section 6 as it depends on the average number of encryption layers, however $2\kappa m$ bits are used to store locally the group elements given that we always blind the same m elements.

4.1.2 Cascade Rebuild scheme. The rebuild method aims at replacing all the mix encryption and permutation layers with new ones ; the key challenge here is that the intermediaries should never see the underlying client records. In order to achieve this, the records are encrypted and decrypted in two phases : a simple encryption-decryption ((2) E/D phase) and then an encryption-permutation ((1) the Unwrapping phase and (3) the Wrapping phase) as shown in Figure 1. We use in the Rebuild method the AES encryption method in Counter mode and take as counter the record current index.

Before uploading the records to the untrusted storage for the first time, the client prepares the data as follows. The records are first encrypted with the client own private keys. The first encryption keys and permutation seeds are then generated and used to encrypt the records once with fixed counters and another time while permuting the records at the same time (with varying counters), i.e. locally doing the Simple Encryption phase (2) and the Wrapping phase (3) of Figure 1.

Mix instructions. The client sends to each mix the same information as in the Cascade Layered design but with two group elements : α_i being used to undo the old permutations and decrypt the old encryption layers (in the Unwrapping and E/D phase), and α'_i used for the new encryption and permutations (in the E/D phase and the Wrapping phase). The client thus send to each mix M_i:

$$C \rightarrow M_i\ :\ db,\ \alpha_i,\ \alpha'_i,\ \kappa,\ list$$

The mix M_i thus computes the permutation seeds and encryption keys as follows.

$$\alpha_i = g^{z_i},\ ss_i = y_i^{z_i},\ k_i, \sigma_i = hkdf(ss_i,\ \kappa)$$
$$\alpha'_i = g^{z_i},\ ss'_i = y_i^{z'_i},\ k'_i, \sigma'_i = hkdf(ss'_i,\ \kappa)$$

Mix operations. In this scheme, the mix M_i receives a list of encrypted records from the mix M_j or fetch the database if it is the first mix. During the Unwrapping phase, the mixes remove first the old encryption and then the permutations thanks to the old keys k_i and seeds σ_i and send the records to M_{i+1}, the last mix M_m instead sends them to itself. The mixes then in the simple E/D phase encrypt the records with both the new and old keys thanks to AES in Counter mode commutativity and send to the previous mix in the list, with the first mix M_0 sending it to the last mix M_m. Finally, in the Wrapping phase, the records are permuted with σ'_i, and then encrypted with k'_i and sent to M_{i-1} or the database for the first mix.

Client operations. To find a record index, the client uses the last m seeds to simulate the mix permutations during the previous eviction or the preprocess.

When retrieving a record, the client first computes the record's remote index using the permutation seeds. The client saves all intermediary and final indices and use them as counters to decrypt the record sequentially r times. The client then decrypts the record with all the shared secrets and its own encryption key together with the original index as counter to reveal the plain-text. The client then updates the encryption of the record to read or write in the local cache, and uploads the cache back to the ORAM server.

Costs. As the whole database is sent through the mix-net three times, the mix communication cost is $(3m) \cdot n \cdot b$, the mix permutation cost is $2mnC_\Pi(n)$ with $C_\Pi(n)$ the cost of permuting n elements and the encryption cost $4mnC_{ctr}$ with C_{ctr} the cost of encrypting one data block. The client Lookup cost is of the order $mC_\Pi(n)$. The client decryption cost is $2mC_{ctr}$, and the group elements stored on the client requires $2\kappa m$ bits of storage.

Both of the Cascade Layered and Cascade Rebuild designs are not efficient as they do no fully utilize the mixes' capacity: for a single user only one mix works at a time. However, the designs can be used in pipeline when dealing with several users.

To increase the mix-net efficiency, we next study in the following section parallelization to distribute the workload among mixes while keeping the shuffle oblivious. To do so, we change the mix-net configuration to a stratified one and introduce *random transposition shuffles.*

4.2 Parallelizing the Eviction process.

From here on, we replace the cascade configuration of the mix-net with a stratified one and have the mixes simulate random transposition shuffles (RTS) thanks to the use of private and public permutations. We also calculate the number of rounds needed to reach good security by presenting firstly the mixing time of k-RTS before introducing ORAM assumptions to reduce the expected time to achieve randomness.

4.2.1 k-Random Transposition Shuffle. Random Transposition Shuffles (RTS) are widely used models in the study of card shuffling. It consists in a player picking randomly a couple of cards from a same deck, permuting them according to a coin toss and putting them back at the same location. These steps, usually called a round, are then repeated until the deck of cards has been properly shuffled, i.e. until every card sequence is equally possible.

RTS are natural candidates for amortized ORAMs : the rounds are independent and can be run by different entities over time. Diaconis et al. in 1986 [3] have proved that the RTS mixing time of a deck of n cards is of the order $O(n \log n)$, we first look at oblivious k-RTS, an RTS where the client picks and transposes locally k distinct cards to make the scheme more efficient. We stress the difference between doing successively $k/2$ transpositions and what we call k-RTS: in the first case, an element can be transposed several times in a row of $k/2$ transpositions while in k-RTS it is transposed at most once. The result we present affirms that k-RTS converges to the uniform distribution more rapidly than repeating normal RTS.

Security Theorem 1. ***Mixing time of k-RTS.*** *A k-random permutation shuffle of a n card game reaches the uniform distribution in τ rounds, such that*

$$E(\tau) < \frac{2n}{k} \cdot \log(n)$$

Proof. See Appendix 10.1. □

Remark. This theorem gives an upper bound of the number of rounds for $k/2$ disjoint transpositions. However, we use in practice PRG keys which do not guarantee that $k/2$ transpositions are done. The permutation done with the PRG can be decomposed as a sequence of transpositions which may not be disjoint or of size $k/2$. We nevertheless consider that in practice an oblivious k-RTS implies computation and communication cost of the order of $O\left(\frac{n}{k} \cdot \log(n)\right)$.

To simulate the k-RTS over the stratified mix-net we will allocate to each mix a range of indices, for instance the mix M_i fetches from the database the records whose indices are comprised in $[\![i \cdot n/m : (i+1)n/m - 1]\!]$. Each mix then fetches its allocated records and permutes them locally. Finally, all mixes perform the same public permutation on all the indices to allocate the records in the next shuffling round and forward the records to the mixes accordingly. This last permutation is required to simulate the random card choice of the classic RTS shuffle.

ALGORITHM 4.1: Public Record Allocation for mix M_{idx} at round rnd

Input: Public seeds $\sigma_{pub,rnd}$;
Number of records n;
Number of mixes m;

```
records ← Π_{σ_{pub,rnd}}([[1 : n]]);
alloc ← [];
forall i ∈ [[1, m]] do
    alloc ← alloc ∪ records[i · n/m : (i + 1) · n/m];
    alloc[i] ← [alloc[i][j] for j ∈ [1 : n/m] if
      alloc[i][j] ∈ [idx · n/m : (idx + 1) · n/m]];
```

Output: $alloc$

When m mixes perform in parallel the k-RTS, we can improve in theory by another factor m the eviction computation time. However to guarantee that no information is leaked to the adversary, we need each honest mix to perform $r = 2m \log n$ rounds, hence we ask each mix to perform the k-RTS for r rounds.

4.2.2 Oblivious Merge. Before the eviction algorithm is run, the database can be divided in two sets of records depending on whether or not they were retrieved by the user. As such, the database can be represented as a simple binary array of n bits out of which s are 1s, the accessed ones, and $n-s$ are 0s, the others. We argue that in this representation, elements of the same sets are indistinguishable to the adversary thanks to prior encryptions and permutations and thus, fewer rounds are necessary to obliviously shuffle the database from this state since we only need to hide from which set the records are from. Indeed, this assumption significantly reduces the number of possible orders in the adversarial view, there are $\binom{n}{s}$ orders instead of $n!$ (using the Stars and Bars theorem [?]).

We now consider the RTS process in that scenario and assume the records (the bits) are re-encrypted before being permuted such that the merge of the two sets is oblivious to the adversary.

Security Theorem 2. *An oblivious merge (OM) of 2 indistinguishable sets of respective size n and s elements requires τ rounds of 2-RTS such that any arranging is possible, with*

$$\tau(\epsilon) \leq \frac{n}{2} \cdot \log\left(\frac{n}{s}\right)$$

Proof. See Appendix 10.2. □

The k-RTS decreased the mixing time by at least a factor k, and does so independently of the items to shuffle, we make the following conjecture.

Security Conjecture 1. *A k-oblivious Merge (k-OM) of 2 indistinguishable sets of n and s element requires τ rounds such that any order is equally possible, with*

$$\tau(\epsilon) \leq \frac{n}{2k} \cdot \log\left(\frac{n}{s}\right)$$

4.3 Parallel Mix-ORAM

We now consider the shuffling methods with the mix-net in a stratified configuration, where all the mixes perform the same operations in parallel and forward the output to each other as shown in Figure 2.

The mixes have each been allocated a chunk of the database (M_{idx} having $[idx \cdot n/m : (idx+1) \cdot n/m]$) and use the public permutation seeds σ_{pub} to compute which records to send to each mix.

4.3.1 Parallel Layered scheme. In this design, depicted as the Wrapping phase of Figure 2, the records are still appended with a label and an IV token, encrypted and permuted as in subsection 4.1.1, however now chunks of the database are assigned and processed by each mix. Before the eviction, the database is permuted with the old seeds σ_i and encrypted with the old keys k_i. Afterwards, the records are encrypted with both k_i and k'_i, permuted with both σ_i and σ'_i, and the new indices are saved on the client. As no permutation layer is ever removed, the record indistinguishability assumption holds, the eviction then consists of $r = m/2 \ \log(n/s)$ rounds.

Mix Instructions. The client needs to send to each mix the session keys to access the database db, the private and public elements used to compute the encryption keys, the permutation seeds and the record allocation α_i and β_i, the security parameter κ, the number of records and rounds n and r, and the ordered $list = (ports, ips)$ of the mixes participating in the eviction. The client thus send :

$$C \rightarrow M_i \ : \ db, \ \alpha_i, \ \beta_i, \ \kappa, \ n, \ r, \ list$$

We generate the permutation seeds and encryption keys as before and furthermore refresh them at each round by blinding the group elements. Let $h_b : \mathcal{G}^* \rightarrow \mathbb{Z}_q^*$ be the hash function used for computing blinding factors, we can then compute recursively the α and β for the round $j+1$ as follows:

$$\alpha_{i,0} = g^{z_i}, \qquad ss_{i,0} = y_{i,0}^{z_i}, \qquad k_{i,0}, \ \sigma_{i,0} = hkdf(ss_{i,0}, \ \kappa)$$
$$\beta_{i,0} = g^{\Pi_{j \neq i} m_j}, \qquad sk_0 = \beta_{i,0}^{m_i}, \qquad \sigma_{pub,0} = hkdf(sk_0, \ \kappa)$$
$$b_{i,j+1} = h_b(\alpha_{i,j}, ss_{i,j}), \qquad \alpha_{i,j+1} = g^{z_i \Pi_{k \leq j} b_{i,k}}$$
$$b_{pub,j+1} = h_b(sk_j), \qquad \beta_{i,j+1} = g^{\Pi_{k \leq j} b_{pub,k} \Pi_{I \neq i} m_I}$$

Mix operations. In this scheme, the mix M_i receives a list of encrypted record from all mixes or from the database at the first round. It first merges the records and sorts them to the order given by the previous public record allocation. It then encrypts each record and permutes them with the private encryption key and private permutation seed. It finally blinds the public permutation seeds, computes the new record allocation and sends the records to the mixes accordingly, or to the database at the last round.

Client operations. The client can find the record index locally as it was stored before the eviction. To decrypt a record, the client uses a similar algorithm to the one used in the cascade configuration. The Parallel Trial and Error algorithm, as described in Algorithm 4.2, now needs to decrypt the records for each eviction and each rounds, and determines for each round which mix processed the desired record. The access method is the same as in the Cascade.

Costs. The mix communication cost is $(r+2) \cdot n \cdot b$, the mix permutation cost is $m \log(n/s) \cdot C_\Pi(n/m)$ with $C_\Pi(n/m)$ the cost of permuting n/m elements and the encryption cost $m/2 \log(n) \cdot C_{cbc}$ with C_{cbc} the cost of encrypting one data block. The client Lookup cost is of the order $O(1)$ as the indices, i.e. $n \log n$ bit, are stored locally. The client decryption cost will be talked in Section 6, and the group elements stored on the client represents $2\kappa m$ bits.

4.3.2 Parallel Rebuild method. This design, depicted in Figure 2, is composed of three phases as in the Cascade configuration. However, we now assign a chunk of the database to each mix which processes during a specified of rounds $r = 2n/k \log(n)$ during each permutation phase and encrypt some of the records in parallel. Before the eviction, the records are permuted and encrypted by the

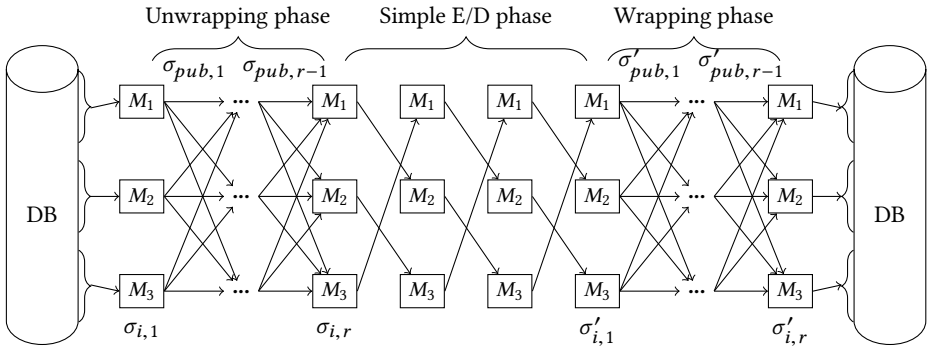

Figure 2: Parallel Mix-ORAM with 3 mixes.
Rebuild method (all phases) and Layered method (only the Wrapping phase).

ALGORITHM 4.2: Parallel Layered Trial Error Algorithm

Input: Record and index $rec,\ index$;
Shared and private encryption keys $k_{mix,eviction,round},\ prv$;
Permutation seeds σ;
Number of rounds r;
List of mixes $list = (ips,\ ports)$;

$j, e = 0$;
$r \leftarrow$ decrypt(prv, rec);
while $rec.data.label! = i$ **do**
  if $e! = 0$ **then**
    $rec \leftarrow$ encrypt$(prv,\ rec)$;
  forall $j \in [\![1 : r]\!]$ **do**
    $m \leftarrow$ retrieve_mix$(\sigma,\ e,\ j,\ index,\ list)$;
    $rec \leftarrow$ decrypt$(k_{m,\ e,\ j}, rec)$;
    $j \leftarrow j - 1$;
  $rec \leftarrow$ decrypt$(prv,\ rec)$;
  $e \leftarrow e - 1$;

Output: rec

mixes M_i with the permutation seeds σ_{pub} and σ_i and the encryption keys k_i. Afterwards, the database is encrypted with the keys k'_i and permuted with the seeds σ'_{pub} and σ'_i.

Mix Instructions. The client sends the same information as in the Parallel Layered design but with twice the number of group elements:

$$C \rightarrow M_i\ :\ db,\ \alpha_i,\ \alpha'_i,\ \beta,\ \beta',\ \kappa,\ n,\ r,\ list$$

To derive the permutation seeds σ and encryption keys k, we make use of the random private elements z, i and m_i, the public and private keys y and x as in the Layered method. We furthermore derive the α r more times for the simple E/D phase.

$$b_{i,j+1} = h_b(\alpha_{i,j}, ss_{i,j}), \qquad \alpha_{i,j+1} = g^{z_i \Pi_{k \le j} b_{i,k}}$$
$$b_{pub,j+1} = h_b(y_c, sk_j), \qquad \beta_{i,j+1} = g^{\Pi_{k \le j} b_{pub,k} \Pi_{l \ne i} m_l}$$

Mix Operations. During the first r rounds, the records are first sorted according to the previous public record allocation, then unwrapped (permuted and decrypted with the old keys and seeds) and sent to the mix-net according to the public record allocation generated from the blinded public seeds. Then the groups of n/m records are encrypted and decrypted in m parallel cascades. Finally, the records are similarly sorted, wrapped (encrypted and permuted with the new keys and seeds) and sent to the mix-net during the last r rounds.

Client Operations. To find a record position, the client uses a similar algorithm as the one used in the Cascade Rebuild scheme. The Parallel Index Lookup Algorithm, as described in Algorithm 4.3, however needs to determine at which round where the record was sent and processed and compute the associated keys and seeds. These intermediary results, the list of indices, keys and seeds, can be stored to facilitate the decryption of the record ; the method being similar to the one used in the Cascade configuration.

ALGORITHM 4.3: Parallel Index Lookup

Input: Private and public seeds $\sigma_{i,round},\ \sigma_{round}$;
Number of records, mixes and rounds $n,\ m,\ r$;
Record index $index$;

$mixes \leftarrow \{,\}$;
$indices \leftarrow \{,\}$;
forall $i \in [\![1, r]\!]$ **do**
  $mix \leftarrow \lfloor index/m \rfloor$;
  $mixes \leftarrow mixes \cup \{mix\}$;
  $shuffle \leftarrow \Pi_{\sigma_{mix,i}}(i \cdot n/m,\ )i + 1) \cdot n/m)$;
  $index \leftarrow i \cdot n/m + shuffle.index(index)$;
  $indices \leftarrow \cup\{index\}$;
  $shuffle \leftarrow \Pi_{\sigma_i(1,n)}$;
  $index \leftarrow shuffle.index(index)$;

Output: $mixes,\ indices$

Costs. The mix communication cost is $(2m + r + 2) \cdot n \cdot b$, the mix permutation cost is $8m \log n \cdot C_\Pi(n/m)$ with $C_\Pi(n/m)$ the cost of permuting n/m elements and the encryption cost $n(4 \log n + 2) \cdot C_{ctr}$ with C_{ctr} the cost of encrypting one data block. The client Lookup cost is of the order $m(C_\Pi(n) + C_\Pi(n/m))$. The client decryption cost is $(r + m)C_{ctr}$, and the group elements stored on the client represents $2\kappa(m + 1)$ bits.

5 SECURITY ARGUMENT

We first remark that all of the eviction meta data is independent of data content, as it is entirely determined by the sole parameter n. The mix instructions are never shared between parties, the keys and seeds thus remain secret and are refreshed at every round.

Cascade mix-net. In this architecture, the whole database passes by every mix including the honest one where it is locally permuted and re-encrypted with the private shared keys. As a polynomial adversary cannot break the PRF, the database order is kept confidential.

For the *Rebuild method*, the simple Encryption/Decryption phase ensures that the records are always encrypted as the adversary is not able to break the double AES encryption.

Parallel mix-net. In this architecture, chunks of the database are exchanged between mixes during r rounds. The adversary can benefit of the fact that some records may never go to the honest mixes but this happens with negligible probability of $p = (e^{-r/m} << 1$ with our parameters.

For the *Rebuild method*, we derived the number of rounds needed to secure our design from the method used in Goodrich 2012 [19] to quantify the information leakage (see Appendix 10.3) and found that this number of rounds is sufficient to bound the expected sum of square error between the card assignment probabilities and the uniform distribution by at most $1/n^2$. The simple E/D phase similarly to the Cascade configuration prevents any mix or the adversary from the decrypting the records completely when they are not permuted and thus the leakage of which records were accessed.

For the *Layered method*, we proposed to use the previous randomness to reduce the number of rounds needed to be close to the uniform distribution. We can also reuse Goodrich's proof by changing the probabilities such that $w_i(t)$ being now the probability the i^{th} record at the t^{th} round was in the cache at first and $\Phi(t) = \Sigma w_i(t) - s/n$, we obtain $r > m \log(n/s)$ see Appendix 10.2 and Conjecture 1).

6 EVALUATION

Layered method. We look here at the average number of encryption layers e a record has before being decrypted. Making the assumption that the record access distribution is uniform, we can represent the problem of accessing all records at least once as a coupon collector problem. In that case, the average number of evictions before all records have been fetched once is $E[e_{all}] \leq (n/s) \cdot H_n$ with H_n the n^{th} harmonic number. The expected number of encryption layers per record before decryption is however $E[r] \leq r/s \cdot \left(\frac{n+1}{2} \cdot (H_n - 1/2) + 1/2\right)$. For $n = 10^6$ and $s = \sqrt{n}$, we have $E[e] \approx 15 \cdot 10^3$ and $E[r] \approx 7 \cdot 10^3 \cdot r$.

Proof. Lets τ_n be the random number of coupons collected when the first set contains every n types. We have, $E[\tau_n] = n \sum_{i=1}^{n} \frac{1}{i} = n \cdot H_n$. Since we fetch s unique records per eviction (we cannot fetch a record already in the stash), the previous result is an upper bound of the number of requests needed and so the expected number of eviction is $E[e_{all}] \leq n/s\ H_n$.

We now want to find the average number of encryption layers per record before decryption, this is equivalent to finding the average number of evictions before a record is deciphered. Hence we have, $E[e] \leq r/s \cdot \sum_{i=1}^{n} E[\tau_i] = r/s\ \sum_{i=1}^{n} \left(\frac{(n+1-i)(n+i)}{2} \cdot \frac{1}{i}\right)$ from which can be calculated the result presented earlier. □

To reduce these numbers, we can modify the access method as follows. When the client requests a record from the database, d other records are chosen uniformly at random from the set of unaccessed records. These records are then fetched, their encryption is refreshed as written previously and the client overwrites with these records their older version on the database. Doing so, with d high enough, yields a better approximation of the uniform distribution assumption and we would obtain $E[e_{all}] \leq n/(sd) \cdot H_n$ and $E[e] \leq r/(sd) \cdot \left[\frac{n+1}{2} \cdot (H_n - 1/2) + 1/2\right]\ H_n$. With $d = \sqrt{n}$, we now have $E[e_{all}] \leq 15r$ and $E[e] \leq 7r$.

Another method to reduce the decryption cost would be to reinitialize the database periodically, for instance every e eviction. Doing so, the client would only need to decipher each record a maximum of $e \cdot m$ times for the Cascade architecture and $e \cdot r$ times for the Parallel architecture during the reinitialization process and in the decryption method.

7 COMPARISON

We can find in Table 2 and Table 3 the cost comparisons of the different Mix-ORAM designs. We did not include the public permutation costs in the Parallel cases as they can be done offline, or during the records' exchange at each round, since the permutation is done on the range of indices and not on the data. We can see that the Layered method is more efficient than the Rebuild one in theory, however we have to take into account in practice the added cost due to the fact that the whole database may not fit in the cache of the mixes (the time needed to fetch the records from the main memory). Moreover, the client incurs higher costs, both in term of memory and computation, with the Layered method. Comparison of our schemes. The mix cost in the Cascade architecture are quadratic in the number of mixes m while they could be considered independent of m in the Parallel case. Hence, the Parallel architecture, even if it has a higher number of rounds, can still be faster than the Cascade depending on the siwe of the network and of the cache for the Layered method.

Comparing the computation and communication costs of our designs to existing eviction schemes would be interesting but delicate as we take into account the fact that the mixes may have faster processor or larger RAM and that the bandwidth in the mix-net may be higher than the one between the client and the ORAM process thus speeding the eviction. The total communication or computation cost of each of our design is indeed higher than regular evictions' such as Melbourne's [28]. However, in our cases, the client only needs to preprocess the database once, and with the periodic reinitialization of the database in the Layered approach, and has manageable additional costs for the lookup and description of a record. These costs do not compare with the overhead of the periodic eviction incurred by the client in the non delegated schemes eviction.

	Cascade - Layered	Cascade - Rebuild
Mix memory	n	n
Mix Encryption cost	$m \cdot n$	$4m \cdot n$
Mix Permutation cost	$mn \cdot C_{\Pi}(n)$	$2mnC_{\Pi}(n)$
Mix Communication cost	$(m+1) \cdot C_{com}(n)$	$3m \cdot C_{com}(n)$
Client Lookup overhead	$O(1)$	$m \cdot C_{\Pi}(n)$
Client Decryption overhead	$\sim \frac{nm}{2s} H_n$	$2m$
Client Storage overhead	$n\log(n) + 2\kappa m$	$2\kappa m$

Table 2: Cost comparison of the designs with C_E the cost of 1 encryption, $C_{\Pi}(x)$ the permutation cost and $C_{com}(x)$ the communication cost of x records in the scheme.

	Parallel - Layered	Parallel - Rebuild
#Rounds (r)	$\frac{m}{2}\log\left(\frac{n}{s}\right)$	$2m\log(n)$
Mix memory	n/m	n/m
Mix Encryption cost	$\frac{n}{2}\log\left(\frac{n}{s}\right)$	$n(4\log(n)+2)$
Mix Permutation cost	$m\log\left(\frac{n}{s}\right) \cdot C_{\Pi}\left(\frac{n}{m}\right)$	$8m\log(n) \cdot C_{\Pi}\left(\frac{n}{m}\right)$
Mix Communication cost	$m(r+2) \cdot C_{com}\left(\frac{n}{m}\right)$	$m(2r+m+2) \cdot C_{com}\left(\frac{n}{m}\right)$
Client Lookup overhead	$O(1)$	$m \cdot [C_{\Pi}\left(\frac{n}{m}\right) h + 2C_{\Pi}(n)]$
Client Decryption overhead	$\sim \frac{nr}{2s} H_n$	$m + r$
Client Storage overhead	$n\log(n) + 2\kappa(m+1)$	$2\kappa(m+1)$

Table 3: Cost comparison of the designs with C_E the cost of 1 encryption, $C_{\Pi}(x)$ the permutation cost and $C_{com}(x)$ the communication cost of x records in the scheme.

8 ACKNOWLEDGEMENT

Danezis was supported by H2020 PANORAMIX Grant (ref. 653497) and EPSRC Grant EP/M013286/1; and Toledo by Microsoft Research.

9 CONCLUSION

We presented in this paper a novel ORAM eviction system where the randomization, more specifically the oblivious shuffle, is delegated to a semi-trusted mix-net. Doing so, the client is alleviated from the main overhead of the ORAM technology at the cost of reasonable additional costs for the record lookup and decryption. Very thin clients can thus accede to the ORAM technology as only a few group elements are needed for fetching any records. The database is moreover accessible and can be made available during the eviction of the records, and this independent of the structure of the underlying ORAM server, making the ORAM technology more portable.

REFERENCES

[1] Miklós Ajtai. 2010. Oblivious RAMs without cryptographic assumptions. In *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM, 181–190.
[2] Miklós Ajtai, János Komlós, and Endre Szemerédi. 1983. An 0(n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*. ACM, 1–9.
[3] David Aldous and Persi Diaconis. 1986. Shuffling cards and stopping times. *The American Mathematical Monthly* 93, 5 (1986), 333–348.
[4] Michael Backes, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. [n. d.]. Anonymous RAM. ([n. d.]).
[5] Kenneth E Batcher. 1968. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. ACM, 307–314.
[6] Stephanie Bayer and Jens Groth. 2012. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 263–280.
[7] Dan Boneh, David Mazieres, and Raluca Ada Popa. 2011. Remote oblivious storage: Making oblivious RAM practical. (2011).
[8] David L Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (1981), 84–90.
[9] Joan Daemen and Vincent Rijmen. 2013. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media.
[10] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. 2011. Perfectly secure oblivious RAM without random oracles. In *Theory of Cryptography Conference*. Springer, 144–163.
[11] George Danezis, Roger Dingledine, and Nick Mathewson. 2003. Mixminion: Design of a type III anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*. IEEE, 2–15.
[12] George Danezis and Ian Goldberg. 2009. Sphinx: A compact and provably secure mix format. In *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 269–282.
[13] George Danezis and Ben Laurie. 2004. Minx: A simple and efficient anonymous packet format. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM, 59–65.
[14] Martin Franz, Peter Williams, Bogdan Carbunar, Stefan Katzenbeisser, Andreas Peter, Radu Sion, and Miroslava Sotakova. 2011. Oblivious outsourced storage with delegation. In *International Conference on Financial Cryptography and Data Security*. Springer, 127–140.
[15] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on*

Theory of computing. ACM, 182–194.
[16] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.
[17] Michael T Goodrich. 2010. Randomized shellsort: A simple oblivious sorting algorithm. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1262–1277.
[18] Michael T Goodrich. 2014. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in O(n log n) time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM, 684–693.
[19] Michael T Goodrich and Michael Mitzenmacher. 2012. Anonymous card shuffling and its applications to parallel mixnets. In *International Colloquium on Automata, Languages, and Programming*. Springer, 549–560.
[20] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. 2012. Privacy-preserving group data access via stateless oblivious RAM simulation. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 157–167.
[21] Jens Groth and Steve Lu. 2007. A non-interactive shuffle with pairing based verifiability. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 51–67.
[22] Jens Groth and Steve Lu. 2007. Verifiable shuffle of large size ciphertexts. In *International Workshop on Public Key Cryptography*. Springer, 377–392.
[23] Markus Jakobsson, Ari Juels, and Ronald L Rivest. 2002. Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking.. In *USENIX security symposium*. San Francisco, USA, 339–353.
[24] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to modern cryptography*. CRC press.
[25] Hugo Krawczyk. 2010. Cryptographic extraction and key derivation: The HKDF scheme. In *Annual Cryptology Conference*. Springer, 631–648.
[26] Grant McWilliams. 2014. Hardware aes showdown-via padlock vs intel aes-ni vs amd hexacore. (2014).
[27] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. 2003. Mixmaster protocolâĂŤversion 2. *Draft, July* 154 (2003).
[28] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. 2014. The Melbourne shuffle: Improving oblivious storage in the cloud. In *International Colloquium on Automata, Languages, and Programming*. Springer, 556–567.
[29] Rafail Ostrovsky. 1990. Efficient computation on oblivious RAMs. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 514–523.
[30] Michael S Paterson. 1990. Improved sorting networks with O(log N) depth. *Algorithmica* 5, 1-4 (1990), 75–92.
[31] Benny Pinkas and Tzachy Reinman. 2010. Oblivious RAM revisited. In *Annual Cryptology Conference*. Springer, 502–519.
[32] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. 2014. Ring ORAM: Closing the Gap Between Small and Large Client Storage Oblivious RAM. *IACR Cryptology ePrint Archive* 2014 (2014), 997.
[33] Emil Stefanov, Elaine Shi, and Dawn Song. 2011. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652* (2011).
[34] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: an extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 299–310.
[35] Sameer Wagh, Paul Cuff, and Prateek Mittal. 2016. Root ORAM: A Tunable Differentially Private Oblivious RAM. *arXiv preprint arXiv:1601.03378* (2016).
[36] Douglas Wikström and Jens Groth. 2006. An adaptively secure mix-net without erasures. In *International Colloquium on Automata, Languages, and Programming*. Springer, 276–287.
[37] Peter Williams, Radu Sion, and Bogdan Carbunar. 2008. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 139–148.

10 APPENDIX

10.1 Proof k-RTS

Proof. To prove the upper bound, we use Diaconis et al. method [3] which consists in marking cards depending on whether they have already been picked or not. Let's define τ the stopping time, i.e. the time when every card has been marked and τ_i the number of transpositions before i cards have been marked. The τ_i are independent geometric variables with probability of success p_t as implied by the game rules. We thus have,

$$\begin{aligned} p_t &= \sum_{i=1}^{min(k,n-t)} \binom{k}{i} \cdot \binom{t+1}{i} \cdot \binom{n-t}{i} \cdot \binom{n}{i}^{-2} \\ &= \frac{1}{n^2} \cdot \left(k \cdot (t+1) \cdot (n-t) + \alpha_{n,t,k}\right) \end{aligned}$$

With $\alpha_{n,t,k} = O\left(n^{-k}\right)$ positive.

We can thus rewrite τ's expectation as following.

$$\begin{aligned} E(\tau) = E\left(\sum_{i=0}^{n-1} \tau_i\right) &= \sum_{t=0}^{n-1} \frac{1}{p_t} < \sum_{t=0}^{n-1} \frac{n^2}{k \cdot (t+1) \cdot (n-t)} \\ &< \frac{2}{k} \cdot \frac{n^2}{n+1} \cdot \left(\ln(n) + \gamma + O(\frac{1}{n})\right),\ \gamma = \lim_{n\to\infty} H_n - \ln(n) \end{aligned}$$

□

10.2 Proof of Oblivious Merge

Proof. We want to find the mixing time $\tau(\epsilon)$ of our oblivious merge of two sets of indistinguishable elements. To do so, we use the bound of the mixing time of an irreducible ergodic Markov Chain, where $p = \frac{1}{|V|}$, with the volume $V = \binom{n}{s}$, and $1 - \lambda^*$ is the spectral gap, we thus have,

$$\frac{\lambda^*}{1-\lambda^*} \cdot \log\left(\frac{1}{2\epsilon}\right) \leq \tau(\epsilon) \leq \frac{1}{1-\lambda^*} \cdot \log\left(\frac{1}{2\epsilon \cdot \sqrt{p}}\right)$$

We now represent the arranging of merge of the 2 distinct sets by the graph $\mathcal{G}$, a k-regular graph with v vertices corresponding to the different orderings and the undirected edges to transpositions of two elements. By definition, the eigenvalues of the transition matrix of the $\mathcal{G}$ are $k = \lambda'_0 > \lambda'_1 \geq \ldots \geq \lambda'_{n-1}$, and we have,

$$\text{diam}\,(\mathcal{G}) \leq \frac{log(v-1)}{log(\frac{k}{\lambda'^*})} + 1 \text{ with } \lambda'^* = max_{i\neq 0}(\lambda'_i) = k \cdot \lambda^*$$

From which we can deduce that $\lambda^* \geq \left(\binom{n}{s} - 1\right)^{\frac{1}{1-s}} \geq \left(\frac{n \cdot e}{s}\right)^{\frac{s}{1-s}}$ since $diam\,(\mathcal{G}) = s$ the diameter of the graph, $v = \binom{n}{s}$ the number of vertices and $k = s \cdot (n-s)$.

To find an upper-bound of λ^*, we will now look at spectral gap bounding. Let's $\mathcal{G}_{0,1} = \{0,1\}^n$ be the group of elements with the XOR operation and $\mathcal{S} = \{x \in \mathcal{G},\ weight(x) = s\}$ the symmetric subset of $\mathcal{G}$ of n-binary array with s 1s and $n-s$ 0s. We call $Cay_{n,s} = Graph\left(\mathcal{G}_{0,1}, \mathcal{S}\right)$ the Cayley graph generated from these structures.

Lemma 10.1. *Let $\mathcal{G}$ be a finite Abelian group, $\chi : \mathcal{G} \to \mathbb{C}$ be a character of $\mathcal{G}$, $\mathcal{S} \subseteq \mathcal{G}$ be a symmetric set. Let M be the normalized adjacency matrix of the Cayley graph $G = Cay(\mathcal{G}, \mathcal{S})$. Consider the*

vector $x \in \mathbb{C}^{\mathcal{G}}$ such that $x_a = \chi(a)$. Then x is an eigenvector of G, with eigenvalue

$$\frac{1}{\mathcal{S}} \sum_{s \in \mathcal{S}} \chi(s)$$

Theorem 10.2. *The Cayley graph $Cay_{n,s}$ has for eigenvalues $\mu_0 = 1 > \mu_1 \geq \ldots \geq \mu_{n-1}$ with,*

$$\mu_r = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{min(r,n-r)} (-1)^i \binom{r}{i}\binom{n-r}{s-i}$$

Proof. $\forall r \in \{0,1\}^n$, with $\chi_r(x) = (-1)^{\sum r_i \cdot x_i}$, we have,

$$\mu_r = \frac{1}{\mathcal{S}} \sum_{s \in \mathcal{S}} \chi(s) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (-1)^{\sum r_i \cdot s_i}$$
$$= \frac{1}{|\mathcal{S}|} \sum_{i=1}^{min(r,s)} (-1)^i \binom{r}{i}\binom{n-r}{s-i}$$

■

Remark. We recognize here the Vandermonde identity with alternating numbers. We argue that the eigenvalues of the Cayley graph $Cay_{n,s}$ are all positive as the smallest eigenvalue is null. For $r = n - r$, the expression simplifies to $\mu_r = \binom{r}{\frac{n}{2}}$ if n even, 0 otherwise. For $r = 1$, the expression simplifies to $\mu_1 = 1 - 2 \cdot \frac{s}{n}$, the spectral gap of $Cay_{n,s}$ is thus equal to $2 \cdot \frac{s}{n}$.

We notice that the first graph $\mathcal{G}$ actually is a sub-graph of $Cay_{n,s}$ and as such the adjacent matrix of the first graph is included in the second's. For $s > 1$, $Cay_{n,s}$ is divided in two sub-graphs representing the cosets of $\{0,1\}^n$ as $\mathcal{S}$ is not a generating group of $\mathcal{G}_{0,1}$, $\mathcal{G}$ is only contained in one of the sub-graphs. We use the Cauchy's Interlace Theorem to bound the eigenvalues of $\mathcal{G}$ with the ones of $Cay_{n,s,}$.

Theorem 10.3. *Let M be a Hermitian $n \times n$ matrix with eigenvalues ${\mu'}_0 \geq \ldots \geq {\mu'}_{n-1}$ and N a $m \times m$ sub-matrix of M with eigenvalues ${\lambda'}_0 \geq \ldots \geq {\lambda'}_{m-1}$, we have*

$${\mu'}_i \geq {\lambda'}_i \geq {\mu'}_{n-m+i+1}$$

We are here only interested in an upper-bound of $\lambda*$, as we have $\mu_{2^n+2-\binom{n}{s}} \leq \lambda_1 \leq 1 - 2\frac{s}{n}$ and $0 \leq \lambda_n \leq \mu_2$, $\lambda* \leq 1 - 2\frac{s}{n}$. We thus have $\frac{1}{1-\lambda*} \leq \frac{n}{2 \cdot s}$ and $\log\binom{n}{s} \approx s(\log(n/s - 0.5) + 1) - 1/2\log(2\pi s)$ when $n \gg s$ from which we derive the final result. □

10.3 Proof of Parallel mix-net

Proof. This proof is derived from Goodrich et al [20] who bounded the closeness of a shuffle to the uniform distribution using a compromised parallel mix-net.

Let $w_i(t)$ the probability the i^{th} record at the t^{th} round was the first record at start, the sum of square metric $\Phi(t) = \Sigma_{i=1}^{n}(w_i(t) - 1/n)^2$, n the number of cards, m the number of mixes out of which m_a are corrupted and $k = n/m$.

We have by recurrence that the potential $\Delta\Phi^*$ changes when a group of K card is shuffled during a round as following : $\Delta\Phi^* = \Sigma_{1 \leq i \leq n}(w_i - w_j)^2$. Thereby,

$$E[\Delta\Phi] = \frac{m}{n} \sum_{1 \leq i \leq n} \Pr((i,j) \text{ in the same honest mix})(w_i - w_j)^2$$
$$= \frac{k-1}{k(n-1)} \cdot \frac{m - m_a}{m} \sum_{i<j} (w_i - w_j)^2$$
$$E[\frac{\Delta\Phi}{\Phi}] = \frac{(m-m_a)(k-1)}{2n(n-1)} \frac{\sum_{i,j}((w_i - 1/n) - (w_j - 1/n))^2}{\sum_k (w_k - 1/n)^2}$$
$$= \frac{(m-m_a)(k-1)}{n-1} \text{ since } \sum_k w_k - 1/n = 0$$

We thus find,

$$E[\Phi(t+1)] = (1 - \frac{(m-m_a)(k-1)}{n-1})E[\Phi(t)]$$
$$E[\Phi(t)] = (1 - \frac{(m-m_a)(k-1)}{n-1})^t$$

We want to find the conditions on c such that the corrupted parallel mix-net can mix in $t = bc\log(n)$ such that $E[\Phi(t)] < n^{-b}$.

$$E[\Phi(t)] = (1 - \frac{(m-m_a)(k-1)}{n-1})^t < n^{-b}$$
$$c \cdot \log(1 + \frac{1}{\frac{n-1}{(m-m_a)(k-1)} - 1}) > 1$$

Using Taylor series, assuming that $n - 1 \gg (m - m_a)(k-1)$, we finally get

$$c \cdot (\frac{1}{\frac{n-1}{(m-m_a)(k-1)} - 1} + o(n/k)) > 1$$
$$c > \frac{n-1}{(m-m_a)(k-1)} - 1 \simeq \frac{m}{m-m_a} - o(1)$$

Thus, when shuffling n cards with n a parallel mix-net composed of m mixes out of which m_a were compromised, we need $t > b \cdot \frac{m}{m-m_a}\log(n)$ rounds before the expected sum of squares error $E[\Phi(t)]$ between the card assignment probabilities and the uniform distribution is at most $1/n^b$ for any fixed $b > 1$. □