Raphael R. Toledo and George Danezis

Mix-ORAM: Using delegated shuffles.

Abstract: Oblivious RAM is a key technology for securely storing data on untrusted storage but is commonly considered impractical due to its high overhead. Indeed, to ensure full privacy to the user, the database has to be periodically randomized, i.e. re-encrypted and shuffled, by the client. To attain full privacy, the computation and communication costs the client incurs are in reality super linear in the number of records, this very price deters the use of ORAM for most applications. We propose in this paper to increase ORAM's appeal by delegating the randomization process to semi-trusted third parties such that the clients are oblivious to the corresponding overhead. To do so, we present four different designs inspired by mix-net technologies and evaluate them. Results results results

Keywords: Oblivious RAM, Mix-net, Private Queries

1 Introduction

Cloud technologies offer the ability to save impressive amounts of data safely and privately on remote servers. To do so, not only the data integrity must be preserved but also the confidentiality of the data content and meta data from both external adversaries and the cloud itself. Cryptographic measures are thereby taken such as secure communication channels, user authentication, data encryption and integrity checking. These actions, however, do not prevent the leakage of all meta data: the server owner can monitor user activities and watch which records were fetched.

Oblivious RAM (ORAM) [15], or Oblivious Storage (OS) [7], precisely prevents an adversary from observing the record access, and do so by introducing a pairing of virtual and real, or remote and local, indices. In these schemes, the records are encrypted and permuted before being uploaded to the untrusted storage and the information to decrypt the records,the encryption keys, and make the matching between the remote and local indices, a pseudo random generator seed for instance, is saved locally. When the user seeks a given record, the local client computes with the seed the corresponding remote index, fetches the encrypted data block obliviously and decrypts it. After a number of accesses, in order to render any leaked information obsolete, the database is fetched and before being uploaded back to the remote server, encrypted and shuffled once again during a so called eviction process before.

This randomization is the main bottleneck of ORAM. Indeed, we assume in such design that the number of records stored remotely is orders of magnitude higher than the client storage. As every record must be re-encrypted and permuted locally to ensure nothing is leaked, the client has to download and process the database by chunks, and do so several times to undeniably hide from the adversary the records' ordering. Thus as the the database size grows, the randomization cost rises super linearly.

Mix networks [8] are a solution widely applied in anonymity systems to obliviously shuffle packets. A batch of packets is processed by the mix-net in that each mix re-encrypts, or decrypts, and shuffles the packets before sending them to the next mix. For the adversary to link the mix-net's input and output, all of the mixes used in the shuffling have to be compromised. As the mixes perform all the operations needed in ORAM, mix-nets inspired designs could thus address the delegation of ORAM's randomization process but at high price since the used of several mixes is needed to ensure the confidentiality of the randomization and that the computation and communication costs are linear in the number of mixes. If using these designs, methods to amortize the eviction must be taken into account to make the randomization practical.

In this work, we present several distributed systems' designs inspired by mix-nets to safely delegate ORAM's randomization process to semi-trusted third parties. The advantages of such practices are, besides the reduction of the client computation, the possibility to delay the eviction to quieter times, the database availability during the eviction process regardless of the ORAM design and the independence from centralized parties.

Raphael R. Toledo: University College London (r.toledo@cs.ucl.ac.uk)
George Danezis: University College London (g.danezis@ucl.ac.uk)

Our contributions are listed as following:

- We present and motivate the use of mix-net with ORAM.
- We present a number of designs, improve them with load balancing via parallel mixing and compare their costs and efficiency.
- We introduce a new security definition of ORAM's eviction.

We present the related work, the ORAM model, its associated threat model and explaining the different costs in Section 2 and 3, we use random transposition shuffles them in ORAM and together with a mixnet and discuss various optimizations in Section 4. We then present our implementation and compare the costs with several designs in Section 6. We finally evaluate our schemes and discuss about the advantages and drawbacks of using mix-nets.

2 Related Work

ORAM. ORAM technologies were first presented by Goldreich and Ostrovsky in 1990 [28] to prevent reverse engineering and protect softwares run on shielded CPU, the concept was update in 2011 with Oblivious Storage (OS) [7] in the context of untrusted remote storage. Since then, several types of enhancement have been proposed including *data structures* diversification [16, 31–33] with trees, partitions and hierarchichal solutions appearing, the use of more and more sophisticated *security definitions* with statistical security [1, 10] and differential privacy [34], and the revision of *item lookups* with cuckoo hashing [30] and bloom filters [36]. Most ORAM constructions are based on a single client-server model, but multi-user designs were gradually introduced as [4] in 2016 which also provides user anonymity, some relying on access control [14] or group access [20]. The eviction process is one of the principal problems of ORAM, the clients have to re-encrypt and process the whole database for extensive period of time during which record access is usually not possible, some designs permitting read while shuffling as [7].

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 on AKS [2] which unfortunately were proved to be impractical because of the high number of I/Os, Batchers using $\mathcal{O}(n \log n)$ I/Os and AKS having a high constant factor, but also more recent and efficient ones [29]. The randomized Shellsort [17] is an elegant simple data-oblivious version of the Shellsort algorithm running in $\mathcal{O}(n \log n)$ time that sort with high probability. The Zig Zag sort [18], presented in 2014, is the deterministic data-oblivious variant of the Shellsort with running time of $\mathcal{O}(n \log n)$. Lastly, the bucket sort Algorithm was both studied in Melbourne shuffle [27] where a user rely on temporary arrays and dummies and in [19] with the use of a parallel mixnet.

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, 26]. Mix-nets' main goal is to give users some anonymity by hiding the correspondence between the incoming users' packets and the mix-nets output. To do so, the users' packets go through several mixes which permute them and refresh their encryption. Either reencryption [35] 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 as the packets come from a single user.

3 Preliminaries

3.1 Model

ORAM systems actually rely on two types of data arrays. One we call database and comprises the user's encrypted records and some dummies and the other one we call cache, e.g. the shelter in [15], of lesser size and used to store the fetched records in order to hide the number of times they were accessed.

Access method: The client first downloads the cache and checks whether the desired record is present, if so a dummy is fetched from the database else the desired record. Finally, the fetched element is encrypted and updated in the cache which is sent back to the ORAM.

Eviction method: When the cache is full, the client empties it by starting the eviction process. To do so, the client first *rebuilds* the database by merging obliviously the cache and the database and afterwards, starts the *oblivious shuffle*.
The rebuild phase consists in placing obliviously records from the cache back to the ORAM database. The client begins by downloading the cache, then re-encrypts the records and discards any dummies contained within it. It next fetches all the records previously accessed from the database, and overwrites them one by one with their cached version or with a dummy before updating them back in the database. The cache is now empty.
The client starts the oblivious shuffle by selecting a set of mixes from the mix-net and sending to them randomization instructions containing the list of participating mixes and the seeds used for shuffling and encrypting. When receiving the records, the first mixes fetch the database, encrypt the records, shuffle and transmit them to the next mixes according to these instructions.

We consider here 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 with memory of n/m data blocks, and a client with memory of s data blocks. The ORAM server, the mixes and the client additionally have a small memory of capacity $\mathcal{O}(m)$ to store extra information about permutation and encryption.

3.2 Security definitions and Threat model

We presume here of the existence of motivated adversaries trying to subvert a target user's privacy and perhaps compromise his data integrity. We assume the target user utilizes an ORAM system, compliant with the Privacy Definition 1 introduced by Stefanov et al. [32], to protect his data. We futhermore assume that all communication between the client, ORAM server and mixes may be intercepted as in the *global passive adversary* assumption however only message timing, volume and size from honest parties can be known thanks to packet encryption. Finally, we suppose the adversaries have corrupted a number of machines to achieve their goal, unless said otherwise, the ORAM server and all but one mix are considered compromised. We will assume that the compromised machines behave in a *honest but curious* way in that every operation is correctly performed but passively recorded and shared with the adversaries.

Privacy Definition 1. *Let's denote a sequence of k queries $seq_k = \{(op_1, ad_1, data_1),\ ..., (op_k, ad_k, data_k)\}$, 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 their lengths are equal ($k = k'$).*

This work focuses on the ORAM eviction process and more precisely on the randomization phase where sequences of data-blocks are shuffled in an oblivious manner in order to hide the records indexes after access information has leaked. For instance, in the Square Root solution [28] the problem refers to the eviction of the shelter in the database and the upper partitions in a lower one in the Hierarchical case [16]. We evaluate our designs with the following Security Game and consider the adversaries have won the game when discovering the ordering of the remote records.

Security Game. *An adversary gives to the user two ORAM query sequences seq_k and seq'_k of the same size. The user chooses randomly one of the two, executes it and start the eviction. At the end of the eviction the adversary picks which ORAM query sequence was chosen. The adversary wins if the right sequence is chosen with probability higher than* $\Pr = \frac{1}{2} + \epsilon$, *with* $\epsilon \ll \frac{1}{2}$.

Costs. We denote the communication costs by ω for the number of *bits* sent by the ORAM, γ by the client, and by μ sent per mix. We classify the computation cost under its function (encryption, shuffle and secret generation), and the machine doing the computation.

3.3 Cryptographic Primitives

PRG & Seeds. ORAM systems make use of pseudo random generators (PRG) and seeds to save the matching between 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 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 seed and stretches it into a long pseudo random stream.

Encryption. ORAM designs heavily rely on encryption mechanism to obfuscate the database records as the records must be re-encrypted during the eviction and access processes.

Advanced Encryption Standard (AES) [9] was conceived for high speed and low RAM requirements. It can present throughput over 700 MB/s per thread on recent CPUs such as the Intel Core i3 [25] which makes it the ideal choice for ORAM.

To minimize the size of the instructions stored and sent by the client, we make use of elements of a cyclic group of prime order satisfying the decisional Diffie-Hellman assumption. The mixes make use of these elements and derive the different permutation seeds and encryption keys with the a key derivation function, such as the HKDF [**?**], and refresh the them at each round by blinding them with the shared secrets as in [12].

4 Mix-ORAM

In this section, we first introduce the different encryption methods we will use to encrypt the records during the eviction. We then present a simple but impractical Mix-ORAM case before optimizing it with the use of distributed shuffle algorithms.

4.1 Mix and User encryption methods

We present in this work different ways to delegate the eviction process to a semi-trusted mix-net proposing two re-encryption methods. For each method, we show how the mix-nets encrypt the records and how the client recovers a record plain text. We make the assumption that all data has first been encrypted by the user before using the ORAM.

Layered method. In the Layered method, we use the data structure shown in Table 1 composed of an IV token, and a label appended to the record, e.g. the record real indices. The underlying principle of the layered method is to let the mixes encrypt the records with AES in CBC mode thanks to IV tokens during the eviction and to let the client decrypts the requested records after the access. To maintain a low number of encryption layers on top of the records, we propose to modify the access method to fetch more records from the database.

IV token	**data = 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.

During the eviction, the mixes use the IV token as seed to generate an initialisation array. AES-CBC is then used to encrypt the data with the IV and the secrets shared with the client. The $8 \cdot \lceil \log(n)/8 \rceil + b$ first bit of the data is then used as a seed to generate an IV with which the IV token is encrypted under AES-CBC and with the shared secrets. Once all elements haven been re-encrypted, they are forwarded to the next mixes.

To decrypt a record, the client make uses of the trial error process written in Algorithm 4.3. Using the shared secrets, the IV token is first deciphered then the data, these two operations are repeated r times in total. The client then uses its own secret key to decipher the IV and the data another time, if the label is the right one, the process stops, if not the client cancels the last decryption and starts again. To encrypt a record, the client generates the IV token and label, encrypts the data with its own secret key and the IV produced from the IV token, and encrypts the IV with its own secret key and an IV produced from the data.

As the number of encryption layers per record may vary, we delay the decryption of the fetched record to avoid a timing attack. When retrieving a record, before sending the cache back to the ORAM server, the client encrypts the retrieved record with its own key and updates it back in the cache before uploading the latter. After doing so, the client locally decrypts the previous encryption and starts the decryption method described above. Once the plain text is recovered, a new IV token is generated, the label is appended to the record, the data is encrypted with the IV produced from the IV token and the private key. The next time the cache is fetched, the cached version of the record is overwritten with the new version.

We now look at the average number of encryption layers a record has before being decripted. 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, we expect $E[e] \leq (n/s) \cdot H_n)$ evictions before all records have been fetched once with H_n the n^{th} harmonic number, and the expected number of layers per card is $E[r] \leq r/s \cdot \left(\frac{n+1}{2} \cdot (H_n - 1/2) + 1/2\right)$. 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 $E[e] \leq n/s\ H_n$.

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, $E[r] \leq r/s \cdot \sum_{i=1}^{n} E[\tau_i] = r/s \ \sum_{i=1}^{n}\left(\frac{(n+1-i)(n+i)}{2}\cdot\frac{1}{i}\right)$ from which can be calculated the result presented earlier. □

To reduce these numbers, we modify the access method as written in Algorithm 4.4. 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, choosing a parameter d high enough yields a better approximation of the uniform distribution assumption. We would have thus $E[e] \leq n/(sd)\cdot H_n$ and $E[r] \leq r/(sd)\cdot\left[\frac{n+1}{2}\cdot(H_n - 1/2) + 1/2\right]\ H_n$. With $d = \sqrt(n)$, we now have $E[e] \leq 15$ and $E[r] \leq 7$.

Algorithm 4.1: Layered encryption primitive

Input: Record rec;
Encryption key k;

$IV \leftarrow urandom(rec.token \cdot k, 128)$;
$rec.data \leftarrow enc\,(k,\ IV,\ rec.data)$;
$IV \leftarrow urandom(rec.data \cdot k, 128)$;
$rec.token \leftarrow enc\,(k,\ IV,\ rec.token)$;

Output: rec

Algorithm 4.2: Layered decryption primitive

Input: Record rec;
Decryption key k;

$IV \leftarrow urandom(rec.data \cdot k, 128)$;
$rec.token \leftarrow dec\,(k,\ IV,\ rec.token)$;
$IV \leftarrow urandom(rec.token \cdot k, 128)$;
$rec.data \leftarrow dec\,(k,\ IV,\ rec.data)$;

Output: rec

Algorithm 4.3: Layered Decryption algorithm

Input: Record and index $record,\ index$;
Shared encryption keys $k_{mix,eviction,round}$;
Private key prv;
Number of rounds r;
Permutation seeds σ;

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

Output: rec

Algorithm 4.4: Layered access method

Input: Real record index $index$;
Operation and data $op,\ towrite$;
Encryption keys k;
Permutation seeds σ;
Saved records $recs$;
Number of rounds r;

$j \leftarrow recover\_index(\sigma, index)$;
$cache \leftarrow fetch\_cache()$;
forall the $rec \in cache$ **do**
  $cache[rec] \leftarrow decrypt(k,\ r,\ \sigma,\ cache[rec])$;
if $record \in cache$ **then**
  $record \leftarrow find(record)$;
  $tofetch \leftarrow choose\_dummies(cache, d + 1)$;
else
  $tofetch \leftarrow choose\_dummies(cache, d)$;
  $tofetch \leftarrow tofetch \cup \{j\}$;
$record,\ d \leftarrow fetch(tofetch)$
$update\_cache(recs, encrypt(record))$;
$send\_cache()$;
if $op == read$ **then**
  $d \leftarrow d \cup \{record\}$;
else
  $d \leftarrow d \cup \{towrite\}$;
forall the $rec \in d$ **do**
  $d[rec] \leftarrow decrypt(k,\ r,\ \sigma,\ d[rec])$;
  $save(encrypt(k,\ d[rec]))$;

Output: record

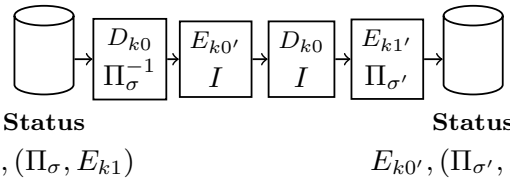

Fig. 1. Eviction under the Rebuild method.
The first row denotes the encryption (E) and decryptions (D) while the second the permutations (Π) and I for the identity.

Rebuild method. The rebuild method aims at replacing all the mix encryption layers with new ones in a manner such that the intermediaries never see the underlying user encryption. In order to achieve this, the encrypted records are first encrypted with the mix encryption keys twice. That way, each mix can either decrypt or re-encrypt the cipher-texts and there will always remain at least one layer of mix encryption. Contrary to the Layered method, the rebuild method does not rely on a data structure, and the encryption used depends on the records order as we use AES in Counter mode (AES-CTR).

Before sending the records to the untrusted storage, the client encrypts the plain-text thrice. The records are always encrypted with AES in CTR mode and we use as counter the record current index. The first time the record are encrypted with the client own private keys, the second time with the shared secrets of every mix, and finally at the last time the records are permuted with permutation seeds and encrypted with the shared secrets of every mix keys at the same time such that the counters vary. For the sake of conciseness, we summarize the state of the database with the status $E_{k0}, (\Pi_\sigma, E_{k1})$ that must be understood as the database was first encrypted with the encryption keys $k0$ and then at the same time permuted and encrypted with the permutation seeds σ and the encryption keys $k1$.

During the eviction, the mixes first unravel the last layer (Π_σ, E_{k1}) by executing $(\Pi_\sigma^{-1}, D_{k1})$. The database is now in the original sequence and encrypted under E_{k0}. The mixes then encrypt the records with $E_{k0'}$ before decrypting with D_{k0} thanks to AES CTR commutativity and the invariant counter. The database is now in the original sequence and encrypted under $E_{k0'}$. Finally, the mixes encrypt and permute at the same time the records executing $(\Pi_{\sigma'}, E_{k1})$, the database is now permuted in the random order $\Pi_{\sigma'}$ and encrypted under $E_{k0'}, (\Pi_{\sigma'}, E_{k1'})$. A scheme summarizing the eviction can be seen in Fig. 1.

When retrieving a record, as shown in Alg 4.6, the client has to compute 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 as written in Alg 4.5. 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. After updating the cache with the encryption of the record to read or write, the latter is sent back.

Algorithm 4.5: Rebuild decryption algorithm

Input: Number of rounds and mixes $r,\ m$;
Encryption keys $k_{mix,round}$;
Mix and Indices $mix,\ round$;

forall the $i \in [\![1, r]\!]$ **do**
  $record \leftarrow dec(record, k_{mix[r-i],r-i}, ind[r-i])$;
forall the $i \in [\![1, m]\!]$ **do**
  $record \leftarrow dec(record, k_{i,0}, index)$;

Output: record

Algorithm 4.6: Renovating access method

Input: Encryption keys $k_{mix,round}$;
Permutation seeds σ;
Record and index $record,\ index$;
Operation and data $op,\ towrite$;
Number of rounds and mixes $r,\ m$;

$indices \leftarrow \{\}$;
$mix \leftarrow \{\}$;
forall the $i \in [\![1, r]\!]$ **do**
  $mix,\ indices \leftarrow$
  $recover\_indices(index,\ r,\ \sigma)$;
$cache \leftarrow fetch\_cache()$;
if $record \in cache$ **then**
  $record \leftarrow find(record)$;
  $tofetch \leftarrow choose\_dummies(cache, 1)$;
else
  $tofetch \leftarrow indices[r]$;
$record, d \leftarrow fetch(tofetch)$;
$record \leftarrow decrypt(record, k, \sigma)$;
if $op == read$ **then**
  $update\_cache(encrypt(record, k, \sigma))$;
else
  $update\_cache(encrypt(towrite, k, \sigma))$;
$send\_cache()$;

Output: record

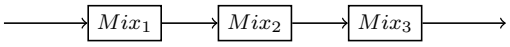

Fig. 2. Cascade architecture with three mixes.

4.2 A simple Mix-ORAM

We consider here the mix-net in a classic cascade configuration as depicted in Fig 2 and present two designs with either the use of the layered encryption method or the rebuild method. We also want to remind that we consider the database to always be permuted according to a number of seeds, denoted Π_σ, the eviction goal being to obviously sort the database to a new state $\Pi_{\sigma'}$.

During the eviction process, the whole database makes a round trip through the mix-net before being updated back to the database. During the way in, the mixes cancel the previous permutation the database has incurred thanks to the old seeds σ given by the client. While during the way out, the mixes permute the database according to the new ones σ as shown in Alg 4.7. At the end of the eviction, or after verification if any, the old seeds are overwritten with the new ones.

When retrieving a record, the user recovers the remote index thanks to the last used seeds. We wrote in Algorithm 4.9 a method to compute all intermediary indices and mixes, the last index being the one sought.

Algorithm 4.7: Cascade sort for mix i

Input: Old and new private seed $\sigma_i,\ \sigma'_i$;
Data $data$;
if in **then**
  $data \leftarrow \Pi^{-1}_{\sigma_i}(data)$;
else
  $data \leftarrow \Pi_{\sigma'_i}(data)$;
Output: $data$

Algorithm 4.8: Cascade Index Lookup

Input: Seeds σ;
Number of records and mixes $n,\ m$;
Record index $index$;
$mixes \leftarrow \{,\}$;
$indices \leftarrow \{,\}$;
forall the $i \in [\![1, m]\!]$ **do**
  $index \leftarrow \Pi_{\sigma_i}(n,\ index)$;
  $mixes \leftarrow mixes \cup \{i\}$;
  $indices \leftarrow \cup\{index\}$
Output: $mixes,\ indices$

Layered Cascade. Before the eviction process starts, the records are in the following state: permuted accordingly to the old seeds σ_i and encrypted under the keys k_i. The design refreshes the permutation, the database being lastly permuted with σ', and add new layers of AES-CBC encryption, the database being encrypted with both k_i and k'_i, as shown in Fig 3. The first time the database passes through a mix mix_i, mix_i undoes the permutation inherited from the previous eviction, i.e. mix_i performs $\Pi^{-1}_{\sigma_i}(DB)$ denoted by σ_i in the figure. The mix mix_i also encrypts the records with the key k_i as detailed before in Algorithm 4.1. The second time, the mix mix_i performs $\Pi_{\sigma'_i}(DB)$, and encrypts every record with the new encryption key k'_i.

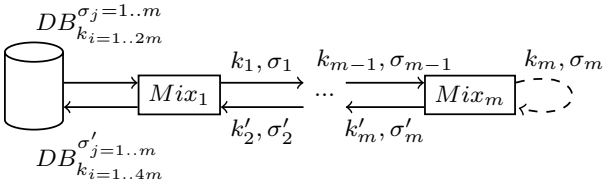

Fig. 3. Layered simple Mix-ORAM eviction.

Mix instructions. The client sends to every mix three elements of a cyclic group of prime order satisfying the decisional Diffie-Hellman Assumption as in Sphinx [12] α_{old} to undo the old permutations, α_{new} to perform the new ones and β to encrypt the records. Alongside these elements, are sent the signed list of mixes ($ports$, ips) and the database access information db consisting of the IP addresses and access token. The client thus send to each mix mix_i:

$$db,\ \alpha_{i,\ old},\ \alpha_{i,\ new},\ \beta_i,\ (ports,\ ips)$$

Let g be a generator of the prime-order cyclic group $\mathcal{G}$ satisfying the Diffie-Hillman Assumption and q the prime order of $\mathcal{G}$. We assume that each mix mix_i has a public key $y_i = g^{x_i} \in \mathcal{G}^*$ with $x_i \in_{\mathbb{R}} \mathbb{Z}_{||}$ alongside the presence of a Public Key Infrastructure to distribute an authenticated list of all (mix_i, y_i). To generate the αs and βs, the client pick at random in $\mathbb{Z}_q$ for each mix mix_i the elements z_i and z'_i. The group elements α, β, shared secrets sk and ss are generated and the encryption keys and permutation seeds are derived as follows:

$$\alpha_i = g^{z_i},\ sk_i = y_i^{z_i},\ k = hkdf(sk_i)$$
$$\beta_i = g^{z'_i},\ s\sigma_i = y_i^{z'_i},\ \sigma = hkdf(ss_i)$$

Mix operations. In this simple scheme, the mix mix_i receives a list of encrypted records from the mix mix_j (or fetch the database if $i = 0$). The mix mix_i encrypts the records with the k_i. If the mix mix_j is before mix_i in the lists of mixes $(ports,\ ips)$ $(j = i - 1)$, the records are permuted with the seed $\sigma_{i,old}$ to cancel $\Pi_{\sigma_{i,old}}$ and sent to mix_{i+1}. If $j = i + 1$, the records are permuted with $\sigma_{i,new}$ and sent to mix_{i-1} (or the database if $i = 0$).

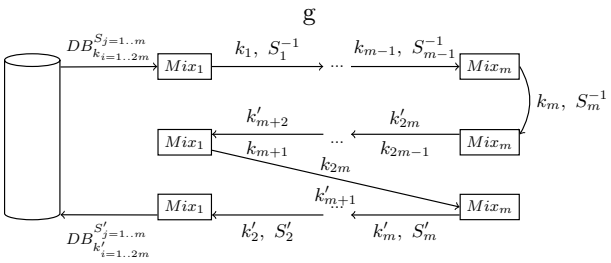

Fig. 4. Renovating a simple Mix-ORAM.

Algorithm 4.9: Layered Cascade mix operation for mix mix_i.

Input: Group elements $\alpha_i,\ \beta_i$;
List of mixes $(ips,\ ports)$;
Data $data$;
Private key x_i

$s\sigma_{i,old} = \alpha_{i,old}^{x_i}$;
$s\sigma_{i,new} = \alpha_{i,new}^{x_i}$;
$sk_{i,} = \beta_i^{x_i}$;
$k_i = hkdf(sk_i)$;
$\sigma_{i,old} = hkdf(s\sigma_{i,old})$;
$\sigma_{i,new} = hkdf(s\sigma_{i,new})$;
forall the $d \in data$ **do**
  $d \leftarrow encrypt_cbc(k_i, d)$;

Output: $mixes,\ indices$

Costs. As the whole database is sent through the mix-net twice, the mix communication cost is $2 \cdot m \cdot n \cdot b$. The client communication is $m \cdot (3 \cdot 32 + (m+1) \cdot (ip + ports) + token)$. The live computation cost is the encryption and shuffle, that is to say $4 \cdot m \cdot n \cdot c_{CBC}$, as the keys and seeds can be derived while the database is being received. TODO: .

Rebuild Cascade. Each of the mixes decrypt and unwrap the records in the first phase then wrap and encrypt in the second, an blinding phase is added between the two during which the records are only encrypted then decrypted thanks to AES-CTR commutativity as depicted in Fig 4.

Message Format. We derive the keys and seeds as previously however we produce twice more shared keys than usual for the wrapping and unwrapping phases. The client only need to send to the mixnet session keys to access the database $(id,\ token)$, the client secrets used to compute the shared secret $(\alpha_{i,old},\ \alpha_{i,new})$, and the signed next mix addresses $(ports, ips)_{signed}$. The message format is then:

$$E_{pub,\ mix_i}\ (id,\ token,\ \alpha_{i,old},\ \alpha_{i,new},\ (ports,\ ips)_{signed})$$

Costs. For 128-bit security, group elements can be expressed in just 32 bytes. We have $c_{cmp} = 3 \cdot m \cdot n \cdot c_{CTR}$, $c_{client} = x$, $c_{mix} = 3n$. The total communication cost is $c_{com} = 3\ cdotm \cdot n + c_{client}$.

These schemes are not efficient as instead of re-encrypting each record twice as in the Melbourne Shuffle, at least $2 \cdot m$ encryption per element are necessary to achieve perfect security. The whole database also goes through every node twice which incur severe delays. To increase the mix-net efficiency, we study in the following sectin parallelization to distributing the workload among mixes.

4.3 Parallelizing the Eviction process.

In this section, we replace the cascade configuration of the mix-net with a parallel one as depicted in Fig 5 and use random transposition shuffles (RTS) to model the parallelization. We also calculate the number of rounds needed to reach perfect security by presenting firstly the mixing time of k-RTS before introducing ORAM assumptions to reduce the expected time to achieve randomness.

Parallel mix-net ...

The records are permuted locally and globally ...

The user when retrieving a given record needs to ...

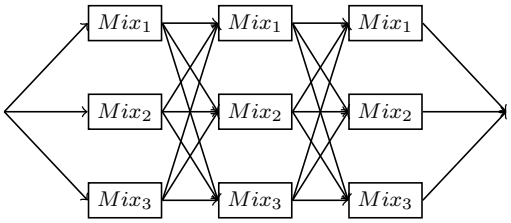

Fig. 5. A Parallel Mixnet

Algorithm 4.10: Parallel sort for mix i during round j

Input: Seeds $S_{pub,old},\ S_{prv,old,i}$;
Number of records n;
Data $data$;

$index \leftarrow \Pi^{-1}_{S_{pub,old,j-1}}[\![1,n]\!]$;
$rec \leftarrow split(index,\ k)$;
forall the $u \in [\![1, n/k]\!]$ **do**
  $rec[u] \leftarrow [Rec[u][v]\ if\ rec[u][v] \in [\![i \cdot k,\ (i+1) \cdot k]\!]\ for\ v \in [\![1,\ k]\!]]$
$data \leftarrow [data\ for\ (rec, data)\ \in sorted(zip(rec,\ data))]$;
$data \leftarrow \Pi^{-1}_{S_{prv,old,j}}(data)$;
Output: $data$

Algorithm 4.11: Parallel permutation for mix i during round j

Input: Seed $S_{pub,new,j},\ S_{prv,new,i,j}$;
Number of records n;
Number of record per mix k;
Data $data$;

$index \leftarrow \Pi_{S_{pub,new,j-1}}[\![1,n]\!]$;
$rec \leftarrow split(index,\ k)$;
forall the $u \in [\![1, n/k]\!]$ **do**
  $rec[u] \leftarrow [Rec[u][v]\ if\ rec[u][v] \in [\![i \cdot k,\ (i+1) \cdot k]\!]\ for\ v \in [\![1,\ k]\!]]$
$data \leftarrow [data\ for\ (rec, data)\ \in sorted(zip(rec,\ data))]$;
$data \leftarrow \Pi_{S_{prv,new,i,j}}(data)$;
Output: $data$

Algorithm 4.12: Parallel Index Lookup

Input: Seeds $S_{pub},\ S_{prv}$;
Number of records per mix k;
Number of rounds r;
Index to retrieve $index$;

forall the $i \in [\![1, r]\!]$ **do**
  $mix \leftarrow index/k$;
  $idx \leftarrow \Pi^{-1}_{S_{prv,mix,i}}(index\ \%\ k)$;
  $index \leftarrow \Pi^{-1}_{S_{pub,i}}(k \cdot mix + idx)$;
Output: $index$

4.3.1 k-Random Transposition Shuffle.

Random Transposition Shuffles (RTS) are widely used examples 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 arranging is possible.
We can already see why RTS are natural candidates for amortized ORAMs : if they can be broke down in independent rounds which can be spread over several entities and time, so can a randomization process based on them. Furthermore, having the client (player) permuting the data blocks (cards) locally is enough to make RTS oblivious to the eyes of an adversary. Diaconis in 1986 [3] has proved that the RTS mixing time of a deck of n cards is of the order of $n\log(n)$. We thus 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 which leads to a different probability distribution. 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*

$$E(\tau) < \frac{2}{k} \cdot \frac{n^2}{n+1} \cdot (\log(n) + \mathcal{O}(1))$$

Proof. The proof can be found in Appendix 10.1. □

PRG seeds actually do not ensure strict transpositions between elements but permutations, that is to say the number of transpositions done while using PRG seeds can be greater than the ones we considered. Hence, the uniform distribution is reached even more quickly. We thus consider that an oblivious k-RTS implies computation and communication cost of the order of $\mathcal{O}\left(\frac{n}{k} \cdot \log(n)\right)$.

When different parties, the m mixes in our case, perform in parallel the k-RTS, we can improve by another factor m the eviction computation time. However, as some mixes can have been compromised by the adversary we ask each mix to perform $2m\log n$ rounds.

4.3.2 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, less rounds are necessary to correctly shuffle the database.

Indeed, this assumption significantly reduces the number of possible orderings in the adversarial view. We can prove, thanks to the Bars and Stripes theorem, that there now are $\binom{n}{s}$ orderings instead of $n!$.

We now consider the RTS process in that scenario and suppose the records (the bits) are re-encrypted before being permuting 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*

$$\tau(\epsilon) \leq \frac{n}{2} \cdot \left[\left(\log\left(\frac{n}{s} - 0.5\right) + \mathcal{O}(1)\right) - \frac{2}{s}\log(4 \cdot \epsilon)\right]$$

Proof. The proof can be found in Appendix 10.2. □

Security Conjecture 1. *A k-oblivious Merge (k-OM) of 2 indistinguishable sets of respective size n and s requires τ rounds doing of k-RTS such that any arranging is possible, with*

$$\tau(\epsilon) \leq \frac{n}{2 \cdot k} \cdot \left[\left(\log\left(\frac{n}{s} - 0.5\right) + \mathcal{O}(1)\right) - \frac{2}{s}\log(4 \cdot \epsilon)\right]$$

Proof. We could see that permuting k elements in RTS decreased the mixing time by at least a factor k, and does so independently of the items to shuffle. Thereby the decrease should still be relevant here. □

4.4 Parallel Mix-ORAM

We now consider the mix-net as a collection of isolated mixes. The unwrapping and wrapping phases consist of $r = \frac{m}{2} \cdot \log\left(\frac{n}{s}\right)$ rounds as depicted in Fig 6.

Using AES-CBC. In this case, the mixes keep encrypting the records with the appended IVs during r rounds of wrapping and unwrapping.

Message Format. The client only need to send to each mix the signed message containing session keys to access the database (id, $token$), information to compute which records it has been attributed ($index$, $range$), the client's secret used to compute the shared secret (α_i, α'_i, α_{pub}, α'_{pub}), the number of rounds r, and the signed ordered ($ports$, ips) of the mixes participating in the eviction. The message format is then :

$$E_{pub,\ mix_i}\big(id,\ token,\ index,\ range,\ \alpha_i,\ \alpha'_i,\ \alpha_{pub},\ \alpha'_{pub},\ r,\ (ports,\ ip$$

Costs. For 128-bit security, group elements can be expressed in just 32 bytes, we have thus a maximum message size of $|addr| + XXX + 3 \cdot 8 \cdot 32 + XXX$. We have $c_{cmp} = 2 \cdot r \cdot n \cdot c_{CBC}$, $c_{client} = x$, $c_{mix} = 2 \cdot r \cdot n$ and $c_{com} = 2 \cdot r \cdot n + c_{client}$.

Using AES-CTR. When performing an eviction, the client assigns to each mix $k = \frac{n}{m}$ distinct indexes together with a list of shared secrets from which will be derive public and private permutation seeds and encryption keys.
At first, the mixes fetch their allocated records from the database. They then unwrap the previous permutation: the last wrapping permutations are undone in reverse order thanks to the old private seeds and the records are decrypted with the old encryption keys before being distributed to all mixes according to the inverse permutation of the old public seeds. Then starts the wrapping, where the records are permuted according to the new seeds, encrypted with the new encryption keys and distributed to the mixes according to the new public seeds. Between the wrapping and unwrapping phases, the mixes exchanged in a cascade fashion their records and encrypt them with both the new and old key.

Message Format. The client send the same data as in the AES-CBC case. The message format is then :

$$E_{pub,\ mix_i}\big(id,\ token,\ index,\ range,\ \alpha_i,\ \alpha'_i,\ \alpha_{pub},\ \alpha'_{pub},\ r,\ (ports,\ ip$$

Costs. For 128-bit security, group elements can be expressed in just 32 bytes, we have thus a maximum message size of $|addr| + XXX + 3 \cdot 8 \cdot 32 + XXX$. We have $c_{cmp} = 2 \cdot (r+m) \cdot c_{CTR}$, $c_{client} = x$, $c_{mix} = (2r+m) \cdot n$. The total communication cost is $c_{com} = (2r+m) \cdot n + c_{client}$.

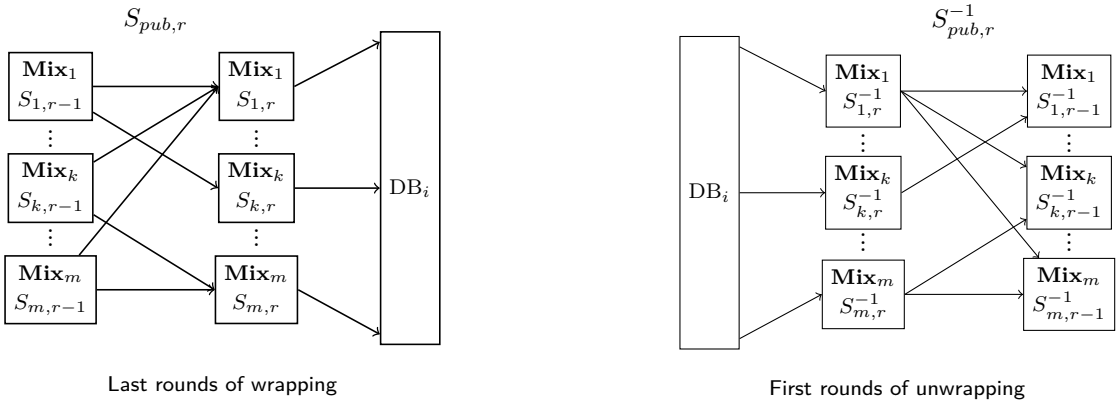

Fig. 6. Wrapping and corresponding unwrapping.

5 Security Proof

We first remark that all of the eviction metadata is independent of data content, it is entirely determined by the sole parameter n and is the same for any access. The data content is also never revealed to the adversary and is refreshed upon reception by every mix. We argue that the adversary can win the game in only two ways: decrypting the records or finding out the record order before eviction.

5.1 In Search of the Lost Order.

The client has sent to each mix private, and potentially public, group elements chosen as random from which the permutation keys are retrieved. The randomness of the seeds was increased thanks to the use of a key derivation function.

Cascade mixnet. In this architecture, the whole database is sent from a mix to another. To know the index of any block, the adversary needs to retrieve the group elements sent to the uncompromised mixes.

Parallel mixnet. In this architecture, chunks of the database are exchanged between mixes. The adversary can benefit of it as some records may never go to uncompromised mixes.
Goodrich in 2012 [19], proved the security of such scheme using the sum of squares metric. If we note by $p(i,t)$ the probability the adversary thinks that the a particular card is at index i at round t, $m - m_a$ the number of uncorrupted server(s) and by $\Phi(t) = \sum_i^n \left(p(i,t) - 1/n\right)^2$ the sum of squares metric, we have that

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

To have $\Phi(t) \leq n^{-c}$ with $m - m_a = 1$, we need $cm \log n \leq r \leq 2cm \log(n)$, the upper bound being the value we propose in Section 4.3.1. The value we propose in Theorem 1, $\frac{m}{2} \cdot log\left(\frac{n}{s}\right)$, accounts for the indistinguishability assumption we made: we do not want to hide the position of every record, only the position of s of them (the accessed ones). ?

5.2 In Search of the Lost Key.

We assume here the adversary knows the invariant position of some of the records he asked the user to query.
The CBC case. In the CBC case, the blocks and IVs are encrypted again and again with AES in CBC mode. If a block was accessed, it is however encrypted only once by the user before the eviction.

The CTR case. In the case of CTR, the adversary only needs to discover the uncompromised mixes' bindings.

uncompromised mix blindings are never known by the adversary as the records are encrypted with AES-CTR when being (un)blinded. Thereby, the records after the unwrapping are undistinguishable from random arrays, and the adversary cannot see which records were accessed.

6 Evaluation

6.1 Implementation and Benchmark

6.2 Performances

performance per enc, performance per design, performance with implementation

6.3 Comparison

Comparison of CBC and tagging

7 Discussion

7.1 Active Adversary

As misbehaving mixes are already discarded in the eviction setup, we could integrate in the client a lightweight local mix reputation system. Randomized Partial Checking

7.2 Differentially Private Oblivious Shuffle

We could ask the mixes to process only a fraction of rounds $r \approx \frac{n}{2 \cdot k} \cdot \log\left(\frac{n}{s} - 0.5\right)$ to increase even more the efficiency of the eviction.

In the worst case scenario where an adversary $\mathcal{A}$ controls all but one mixes, the probability $\mathcal{A}$ knows where a record is at the end of the eviction is $\delta = \Pr(\mathcal{A} \text{ knows a record index}) = \left(1 - \frac{1}{m}\right)^r$ where is r the number of rounds.

To prevent such case, we can make each mix permute every allocated sections of the database and then begins the Oblivious Merge. We would thus add $2 \cdot m^2$ permutations for the whole eviction....

8 Acknowledgement

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

9 Conclusion

- New ORAM with mix-net
- amortizable eviction
- can fetch while eviction running
- multi-user friendly as all the secrets are on the server

References

[1] Ajtai, M.: Oblivious RAMs without cryptographic assumptions. In: Proceedings of the forty-second ACM symposium on Theory of computing. pp. 181–190. ACM (2010)

[2] Ajtai, M., Komlós, J., Szemerédi, E.: An 0(n log n) sorting network. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing. pp. 1–9. ACM (1983)

[3] Aldous, D., Diaconis, P.: Shuffling cards and stopping times. The American Mathematical Monthly 93(5), 333–348 (1986)

[4] Backes, M., Herzberg, A., Kate, A., Pryvalov, I.: Anonymous RAM

[5] Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, spring joint computer conference. pp. 307–314. ACM (1968)

[6] Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 263–280. Springer (2012)

[7] Boneh, D., Mazieres, D., Popa, R.A.: Remote oblivious storage: Making oblivious RAM practical (2011)

[8] Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–90 (1981)

[9] Daemen, J., Rijmen, V.: The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media (2013)

[10] Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly secure oblivious RAM without random oracles. In: Theory of Cryptography Conference. pp. 144–163. Springer (2011)

[11] Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a type iii anonymous remailer protocol. In: Security and Privacy, 2003. Proceedings. 2003 Symposium on. pp. 2–15. IEEE (2003)

[12] Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: 2009 30th IEEE Symposium on Security and Privacy. pp. 269–282. IEEE (2009)

[13] Danezis, G., Laurie, B.: Minx: A simple and efficient anonymous packet format. In: Proceedings of the 2004 ACM workshop on Privacy in the electronic society. pp. 59–65. ACM (2004)

[14] Franz, M., Williams, P., Carbunar, B., Katzenbeisser, S., Peter, A., Sion, R., Sotakova, M.: Oblivious outsourced storage with delegation. In: International Conference on Financial Cryptography and Data Security. pp. 127–140. Springer (2011)

[15] Goldreich, O.: Towards a theory of software protection and simulation by oblivious RAMs. In: Proceedings of the nine-

	C_{comp}	C_{comm}	C_{access}
Cascade - CBC	17.5	19.1	17.5
Cascade - CTR	11.8	12.7	29.3
Parallel - CBC	6.6	5.6	35.9
Parallel - CTR	6.6	5.6	35.9

Table 2. Cost comparison of designs.

	$m = 3$	$m = 5$	$m = 7$
Cascade - CBC	17.5	19.1	17.5
Cascade - CTR	11.8	12.7	29.3
Parallel - CBC	6.6	5.6	35.9
Parallel - CTR	6.6	5.6	35.9

Table 3. Time comparison of designs, with $n = \frac{1024^3}{4 \cdot 1024}$, $s = \sqrt{n}$.

teenth annual ACM symposium on Theory of computing. pp. 182–194. ACM (1987)
[16] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. Journal of the ACM (JACM) 43(3), 431–473 (1996)
[17] Goodrich, M.T.: Randomized shellsort: A simple oblivious sorting algorithm. In: Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. pp. 1262–1277. Society for Industrial and Applied Mathematics (2010)
[18] Goodrich, M.T.: 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. pp. 684–693. ACM (2014)
[19] Goodrich, M.T., Mitzenmacher, M.: Anonymous card shuffling and its applications to parallel mixnets. In: International Colloquium on Automata, Languages, and Programming. pp. 549–560. Springer (2012)
[20] Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-preserving group data access via stateless oblivious ram simulation. In: Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms. pp. 157–167. SIAM (2012)
[21] Groth, J., Lu, S.: A non-interactive shuffle with pairing based verifiability. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 51–67. Springer (2007)
[22] Groth, J., Lu, S.: Verifiable shuffle of large size ciphertexts. In: International Workshop on Public Key Cryptography. pp. 377–392. Springer (2007)
[23] Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: USENIX security symposium. pp. 339–353. San Francisco, USA (2002)
[24] Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press (2014)
[25] McWilliams, G.: Hardware aes showdown-via padlock vs intel aes-ni vs amd hexacore (2014)
[26] Möller, U., Cottrell, L., Palfrader, P., Sassaman, L.: Mixmaster protocolâĂŤversion 2. Draft, July 154 (2003)
[27] Ohrimenko, O., Goodrich, M.T., Tamassia, R., Upfal, E.: The melbourne shuffle: Improving oblivious storage in the cloud. In: International Colloquium on Automata, Languages, and Programming. pp. 556–567. Springer (2014)
[28] Ostrovsky, R.: Efficient computation on oblivious RAMs. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 514–523. ACM (1990)
[29] Paterson, M.S.: Improved sorting networks with O(log N) depth. Algorithmica 5(1-4), 75–92 (1990)
[30] Pinkas, B., Reinman, T.: Oblivious RAM revisited. In: Annual Cryptology Conference. pp. 502–519. Springer (2010)
[31] Ren, L., Fletcher, C.W., Kwon, A., Stefanov, E., Shi, E., van Dijk, M., Devadas, S.: Ring oram: Closing the gap between small and large client storage oblivious RAM. IACR Cryptology ePrint Archive 2014, 997 (2014)
[32] Stefanov, E., Shi, E., Song, D.: Towards practical oblivious RAM. arXiv preprint arXiv:1106.3652 (2011)
[33] Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path oram: an extremely simple oblivious RAM protocol. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 299–310. ACM (2013)
[34] Wagh, S., Cuff, P., Mittal, P.: Root oram: A tunable differentially private oblivious RAM. arXiv preprint arXiv:1601.03378 (2016)
[35] Wikström, D., Groth, J.: An adaptively secure mix-net without erasures. In: International Colloquium on Automata, Languages, and Programming. pp. 276–287. Springer (2006)
[36] Williams, P., Sion, R., Carbunar, B.: 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. pp. 139–148. ACM (2008)

10 Appendix

10.1 Proof k-RTS

Proof. To first prove the upper bound and variance, we use Diaconis et al. results [3] which states that τ defined in the following game is a strong stationary time. In a random transposition shuffle, the cards chosen by the right and left hands at time t are respectively called R_t and L_t. Assuming that when $t = 0$, no card is marked, we mark R_t if R_t was unmarked before and either L_t is marked or $L_t = R_t$. The variable τ represents the time when every card has been marked, we call it the stopping time.

Let be τ_t the number of transpositions after the t^{th} card is marked, up to and including when the $(t+1)^{th}$

is marked.

$$\tau = \sum_{i=0}^{n-1} \tau_i$$

The τ_t are independent geometric variables with probability of success p_t as implied by the game rules. The probability of success corresponds to the probability of marking at least one card, one to t cards exactly. To do so, the right cards must be chosen from the unmarked set, comprising $n-t$ cards at time t, and the left cards from the union of the marked set and the right cards.

$$p_t = \sum_{i=1}^{min(k,n-t)} \binom{k}{i} \cdot \binom{t+1}{i} \cdot \binom{n-t}{i} \cdot \binom{n}{i}^{-2}$$
$$= \frac{1}{n^2} \cdot (k \cdot (t+1) \cdot (n-t) + \alpha_{n,t,k}), \ 0 < \alpha_{n,t,k} = \mathcal{O}\left(n^{-k}\right)$$

We can thus rewrite τ's expectation as following.

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

$$\begin{aligned} var(\tau) &= \sum_{t=0}^{n-1} \frac{1-p_t}{{p_t}^2} < \sum_{t=0}^{n-1} \frac{1}{{p_t}^2} \\ &< \sum_{t=0}^{n-1} \frac{1}{\left(k\frac{(t+1)\cdot(n-t)}{n^2} + \alpha_{n,t,k}\right)^2} \\ &< \sum_{t=0}^{n-1} \frac{1}{\left(k\frac{(t+1)\cdot(n-t)}{n^2}\right)^2} \\ &< 2 \cdot \left(\frac{n}{k}\right)^2 \cdot \left(\frac{n}{n/2}\right)^2 \cdot \sum_{t=0}^{n/2-1} \frac{1}{(t+1)^2} \\ &< \frac{4}{3}\pi^2 \cdot \left(\frac{n}{k}\right)^2 \end{aligned}$$

To now prove the lower bound of τ, we will compare the number of fixed points of a permutation σ, $F(\sigma)$, for our shuffle, the permutation obtained from the identity by applying kt random transpositions $P^{kt}(id, \cdot)$, and the uniform distribution π, or more precisely compare the corresponding probabilities over the set $A = \{\sigma : F(\sigma) \geq \frac{\mu}{2}\}$. We can say that after t shuffles, the number of untouched cards of our shuffle has the same distribution as the number R_{2kt} of uncollected coupon types after $2kt$ steps of a coupon collector chain and that about $P^{kt}(id, \cdot)$ that the associate $F(\sigma)$ is at least as large as the number of cards that were touched by none of the transpositions, i.e. $P^{kt}(id, A) \geq P(R_{kt} \geq A)$.

We know that the R_{2kt} has expectation $\mu = np$ with $p = \left(1 - \frac{1}{n}\right)^{2kt}$, variance $var = np(1-p) < \mu$ and by Chebyshev, we know that $\Pr(R_{2kt} \leq \frac{\mu}{2}) \leq \frac{4}{\mu}$ as $\Pr(|R_{2kt} - \mu| \geq \frac{\mu}{2}) = \Pr(R_{2kt} \geq \frac{3\mu}{2}) + \Pr(R_{2kt} \leq \frac{\mu}{2}) > \Pr(R_{2kt} \leq \frac{\mu}{2})$.

By Markov's inequality we know that $\pi(A) \leq \frac{2}{\mu}$.

As $P^{kt}(id, A) \geq P(R_{kt} \geq A)$, we also have $P^{kt}(id, A^c) \leq P(R_{2kt} \leq A) \leq \frac{4}{\mu}$ which leads to $P^{kt}(id, A) \geq 1 - \frac{4}{\mu}$.

Thus we have $d(t) = ||P^{kt}(id,) - \pi||_{TV} \geq |1 - \frac{4}{\mu} - \frac{2}{\mu}| \geq 1 - \frac{6}{\mu}$.

We want to find the minimum t such that $1 - \frac{6}{\mu} \geq \epsilon$, which is equivalent to $n \cdot \left(1 - \frac{1}{n}\right)^{2kt} \geq \frac{6}{1-\epsilon}$ and to

$$\log\left(\frac{n \cdot (1-\epsilon)}{6}\right) \geq 2 \cdot k \cdot t \cdot \log\left(\frac{n}{n-1}\right)$$

As $\log(1+x) \leq x$, the previous inequality holds if $\log\left(\frac{n\cdot(1-\epsilon)}{6}\right) \geq \frac{2kt}{n-1}$ which means that if $t \leq \frac{n-1}{2k} \cdot \log\left(\frac{n(1-\epsilon)}{6}\right)$ then $d(t) \geq \epsilon$. Thus,

$$\tau(\epsilon) \geq \frac{n-1}{2k} \ln(n \cdot \frac{1-\epsilon}{6})$$

. □

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 $p = \frac{1}{|V|}$ and $1 - \lambda^*$ is the spectral gap,

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

We now want to find a bound for λ^*. We 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 $k = \lambda'_0 > \lambda'_1 \geq ... \geq \lambda'_{n-1}$, and we have,

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

with $diam\,(\mathcal{G}) = s$ the diameter of the graph, $v = \binom{n}{s}$ the number of vertices and $k = s \cdot (n-s)$.

We can thus find a first relation:

$$\begin{aligned} log(\frac{k}{\lambda'^*}) = log(\frac{1}{\lambda^*}) &\leq \frac{\log(v-1)}{\text{diam}\,(\mathcal{G})-1} \\ log(\lambda^*) &\geq \frac{log(v-1)}{1-\text{diam}\,(\mathcal{G})} \\ \lambda^* &\geq (v-1)^{\frac{-1}{\text{diam}(\mathcal{G})-1}} \\ \lambda^* &\geq \left(\binom{n}{s}-1\right)^{\frac{1}{1-s}} \geq \left(\frac{n\cdot e}{s}\right)^{\frac{s}{1-s}} \end{aligned}$$

And can derive the minimum value of $\Delta = \frac{\lambda^*}{1-\lambda^*}$,

$$\begin{aligned} \Delta &= \frac{1}{(\lambda^*)^{-1}-1} \\ &\geq \frac{1}{\left(\frac{n\cdot e}{s}\right)^{\frac{s}{s-1}}-1} \end{aligned}$$

To find an upper-bound of λ^*, we will focus on the 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 $Cay_{n,s} = Graph\,(\mathcal{G}_{0,1}, \mathcal{S})$ the Cayley graph generated from these structures.

Lemma 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 1. *The Cayley graph $Cay_{n,s}$ has for eigenvalues $\mu_0 = 1 > \mu_1 \geq ... \geq \mu_n$ 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,

$$\begin{aligned} \mu_r &= \frac{1}{\mathcal{S}}\sum_{s\in\mathcal{S}}\chi\,(s) \\ &= \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}(-1)^{\sum r_i\cdot s_i} \\ &= \frac{1}{|\mathcal{S}|}\left(|\text{“1”}| - |\text{“}-1\text{”}|\right) \\ &= \frac{1}{|\mathcal{S}|}\sum_{i=1}^{min(r,s)}(-1)^i\binom{r}{i}\binom{n-r}{s-i} \\ &= 1 - \frac{2}{\binom{n}{s}}\cdot\sum_{i=0}^{min(\frac{r-1}{2},\frac{s-1}{2})}\binom{r}{1+2i}\binom{n-r}{s-2i-1} \\ &= \frac{\binom{n-r}{s}}{\binom{n}{s}}\ {}_2F_1(-r,-s,n-2r+1,-1) \end{aligned}$$

■

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 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 2. *Let M be a Hermitian $n \times n$ matrix with eigenvalues $\mu'_0 \geq ... \geq \mu'_{n-1}$ and N a $m\times m$ sub-matrix of M with eigenvalues $\lambda'_0 \geq ... \geq \lambda'_{m-1}$, 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 $\frac{1}{1-\lambda*} \leq \frac{n}{2\cdot s}$ □

10.3 Proof of Fake access

Proof. We want to prove that the average number of fake access is 0 in case of a uniform distribution. To do so, we consider the Markov chain and its Transition Matrix. The transition matrix P represents the s transient

state, in which the stash is not completely filled, and the absorption state in which the stash is full. Thus, P can be decomposed in 4 sub-matrices: the square sub-matrix Q_s representing all the transient state, the column matrix R with the probabilities of transitioning to the absorbing state, the null row matrix and the absorption matrix.

$$\begin{bmatrix} Q_s & R \\ 0_{1\times s} & I_1 \end{bmatrix}$$

To find the average number of steps from one state to the absorbing one, we have solve the following equation, each row corresponding to the average number of steps from the corresponding state (the stashed filled with some records) to the state where the stash is full.

$$\begin{aligned} t &= \left(\sum_{k=0}^{\infty} {Q_s}^k\right) 1 \\ &= (I_s - Q_s)^{-1} 1 \end{aligned}$$

This equation has a solution since $M = I_s - Q_s$ have independent rows and thus an inverse that we call N. By calculus we find that,

$$\begin{aligned} n_{i,j} &= 0 && \text{if } i > j, \\ n_{i,j} &= \frac{1}{m_{i,i}} && \text{if } i = j, \\ n_{i,j} &= -n_{i+1,j} \cdot \frac{m_{i,i+1}}{m_{i,i}} && \text{if } i < j \end{aligned}$$

which can be simplified by

$$\begin{aligned} n_{i,j} &= 0 && \text{if } i > j, \\ n_{i,j} &= \frac{1}{m_{j,j}} \prod_{k=1}^{j-1} \left(-\frac{m_{i,k+1}}{m_{k,k}}\right) && \text{if } i \leq j \end{aligned}$$

We only want to calculate the first solution S_1 from the equation.

$$\begin{aligned} S_1 &= \sum_{j=0}^{s-1} \frac{1}{m_{j,j}} \prod_{k=1}^{j-1} \left(-\frac{m_{i,k+1}}{m_{k,k}}\right) \\ &= \sum_{j=1}^{s-1} \frac{1}{m_{j,j}} \text{ as } m_{i,k+1} = -m_{k,k} \\ &= \sum_{j=0}^{s-1} \frac{1}{1-\frac{j}{n}} = \sum_{j=0}^{s-1} \left(1 + \frac{j}{n-j}\right) \\ &= s + \sum_{j=0}^{s-1} \frac{j}{n-j} \end{aligned}$$

As s steps are required to fill the stash, we thus find the following inequality for the number of fake access f:

$$\frac{s \cdot (s+1)}{2 \cdot n} < f < \frac{s \cdot (s+1)}{2 \cdot (n+1-s)}$$

□