Prof George Danezis and Raphael R. Toledo

Mix-ORAM, Towards a delegated shuffle

Abstract: Oblivious RAM systems suffer of impracticality, notably due to the cost of the randomization process in the eviction phase. We present in this paper an ORAM eviction relying on amortizable shuffle and mix-net technologies to make it more usable.

Keywords: ORAM, Mix-nets

1 Introduction

Cloud technologies provide a range of services using remote servers, including website hosting, computation, and data storage. One of the main issues for the storage providers is to guarantee the users that the data is protected, not only from external adversaries but also from itself, and doing so for reasonable costs. Cryptographic functions are thus used among other things to create secure communication channels, to authenticate users, and to encrypt and check the data integrity. These solutions however do not prevent the leakage of some information and its interception. One can for instance count the number of packets exchanged by two parties, even when using anonymity systems [16], and guess which file was accessed thanks to a-priori knowledge, for instance the file size distribution, the file access frequency distribution, etc. .

Oblivious RAM (ORAM) [11], or Oblivious Storage (OS) [6], are key technologies to answer such problem. In these designs, the access methods are made independent of the input, the sought records, to hide the access pattern. The data-blocks are also periodically randomized to mitigate the information leakage due to data retrieving. If improvements have been made on the data retrieval costs thanks to structural change in the data representation, considering trees [24] or rings [21], and the shuffling algorithm, in particular with the introduction of dummies [17], the eviction problem however still remain a challenge as the whole database has to be re-encrypted and sorted again frequently. The database is consequently unavailable for long periods of time during which the user processes heavy computing. In 2013, Stefanov and Shi [22] proposed a multi-cloud oblivious storage where the eviction algorithm was delegated thanks to the use of several semi-trusted clouds. De Capitani di Vimercati in 2015 [8] used dynamic distributed data allocation on three servers. However, the practicality of owning, managing and storing data on more than one server was not discussed in these papers.

We propose here the use of an independent mix-net to execute the eviction algorithm without relying on any specific ORAM server design. Mixing networks is a solution widely applied in Anonymity system to obliviously shuffle packets. A batch of packets is received by the mix-net where it is re-encrypted and shuffled by each mix such that the output of one mix is the next oneâĂŹs input. For the adversary to link the input and output, all of the mixes used in the shuffling have to be compromised. Mix-nets could thus address ORAM constraints but at high cost since the re-encryption cost is linear in the number of mixes. If using mix-net, methods to amortize the shuffle have to be taken into account to make ORAM practical.

1.1 Our contributions

In this work, we focus on the eviction algorithm. We present a new security definition for the oblivious sorting algorithm and suggest to leave the eviction to untrusted Raphael: semi-trusted? third parties. The advantages of such practices are the delegation of the shuffling, the possibility to delay the latter to quieter times and the independence from centralized parties.

- We present the first ORAM eviction relying on a mix-net.
- We give a new security definition of ORAM's eviction.
- We present an amortizable shuffle.

Prof George Danezis: University College London, E-mail: g.danezis@ucl.ac.uk
Raphael R. Toledo: University College London, E-mail: r.toledo@cs.ucl.ac.uk

1.2 Content

After presenting the related work, the ORAM model, its associated threat model and explaining the different costs in Section. 2, we introduce Random Transposition Shuffles, how to use them in ORAM and together with a mixnet and discuss of various optimizations in Section. 3. We finally evaluate our schemes with standard ORAM model and discuss about the advantages and drawbacks of using mix-nets. TODO: Implementation and Performances.

2 Preliminaries

2.1 Related Work

2.1.1 ORAM

ORAM technologies were first presented by Goldreich and Ostrovsky in 1990 [18] to prevent reverse engineering and protect softwares thanks to shielded CPU. Since then, several types of enhancement have been proposed including data structures [12, 21, 23, 24] with trees, partitions and hierarchichal solutions, security definitions with statistical security [1, 7] and differential privacy [26], and item lookup with cuckoo hashing [20] and bloom filters [27]. If most ORAM constructions are based on a single client server model, multi-user designs were gradually introduced as [4] in 2016 which also provides user anonymity, some relying on right delegation [10] or group access [15]. As we stated before the eviction process is one of the principal problem 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 [6].

2.1.2 Shuffling and Sorting

Shuffle and sorting algorithms are a thoroughly researched subject central to ORAM for the randomization process. However most of them are not oblivious in that the memory accesses done by algorithm depends on the input. Examples of oblivious sorting algorithms include sorting networks such as Batcher's [5] and the ones based on AKS [2] which were proved to be impractical because of their 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 [19]. The randomized Shellsort [13] 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 [14], presented in 2014, is the deterministic data-oblivious variant of the Shellsort with running time of $\mathcal{O}(n \log n)$. Lastly, the Melbourne shuffle [17] is an efficient variant of the bucket sort Algorithm relying on temporary arrays and dummies to hide the item redistribution.

2.1.3 Mix-nets

TODO: Tor [9]. We present here a new type of mix-net as the messages come from the same source and that the mix-net client needs to store some information to remember the output ordering.

2.2 Model

We consider on one side, an ORAM remote server, defined as follows, with memory capacity of $\mathcal{O}(n)$ b-bit long blocks, a mix-net composed of m mixes and on the other side, a client with memory of s-block capacity with $s << n$.

Privacy Definition 1. *ORAM Security Definition. Let* $seq_k = ((op_1, addr_1, data_1), \ldots, (op_k, addr_k, data_k))$ *denote a query sequence of length* k*, where* op *denotes a read or write operation and* $data$ *the block to write 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* $k = k'$*.*

We consider here the use of ORAM on the cloud. As such, the cloud primitives are available at the server side:

- get(index): Returns the record located at *index*.
- get_range($index_i$, $index_j$): Returns the records comprised between $index_i$ and $index_j$.
- put(index, data_block): Writes the data block at *index*, returns an error if the data block is too large.
- put_range($index_i$, $index_j$, data_blocks): Writes the data blocks between $index_i$ and $index_j$, returns an error if the data block are too large.

The remote server can additionally includes a small memory of capacity $\mathcal{O}(m)$ to store the seeds and shuffle status. In that case, the following operations are available at the client :

- shuffle_db(seed): Shuffles the database with the given seed.
- shuffle_range($index_i$, $index_j$, seed): Shuffles the range of records with the given seed.
- shuffle_status(): Returns the shuffle status that is whether a shuffle has been set, if it is delayed to quiet time, whether it is being processed and at what round it is.
- set_shuffle(seeds, nb_mix, rounds, (delay)): Set the next shuffle overwriting its current parameters.

Raphael: Not used so far.

2.3 Threat model and Security definition

The ORAM design assumes the existence of motivated adversaries trying to subvert the user's privacy and perhaps his data integrity. The adversaries might have different goals such as identifying a user data access or matching encrypted data-blocks with known plain text for further purposes such as selective data corruption or monitoring.

We assume that all communication between the different parties (user, 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. In addition, we assume the adversaries may have corrupted the ORAM server and a part of the mix-net to achieve his goal. We will assume that the compromised machines present a *honest but curious* behaviour in that every operation is correctly performed but passively recorded until the Discussion section (c.f. Sec. 5) where we will examine packet injection, dropping and delaying.

This work focuses on the ORAM eviction process where sequences of data-blocks are merged in an oblivious manner in order to hide the records indexes after access information has leaked. For instance, in the Square Root solution the problem refers to the eviction of the shelter in the database and the upper partitions in a lower one in the Partition cases.

Privacy Definition 2. *An ORAM eviction process is considered oblivious if an adversary cannot distinguish after eviction whether a specific record has been retrieved.*
That is to say, for the eviction of s elements in an array of n elements, and with $\epsilon = \mathcal{O}\left(\frac{1}{n}\right)$, we have,

$$\forall i \in [\![1, n]\!],\ Pr(index_i \text{ was retrieved}) = \frac{s}{n} + \epsilon$$

2.4 Costs

We denote the communication costs by $C_{com,o}$ for the number of *bits* sent by the ORAM, $C_{com,c}$ by the client, and by $C_{com,m}$ sent by the mixes. We will call C_{cmp} the computation cost derived from the eviction.

2.5 Cryptographic Primitives

2.5.1 PRG & Seeds

A pseudo-random generator (PRG) is a deterministic function that produces pseudo-random strings, it can be instantiated with a small size parameters, the seeds, to produce the same sequence of characters on any machine provided the latter have functions with the same versions. ORAM designs frequently uses PRGs during the eviction algorithm to shuffle the array and store the mapping between indexes. In the case of classic client-server configuration, the client has stored the previous seed and the new one and has to permute obviously the records such that the new ordering is not revealed. This can be done easily thanks to the user storage: the user can download the whole database, sort it thanks to the indexes, re-encrypt it and upload it back. With the introduction of the mix-net, we have a shift of paradigm: the permutations are now executed by the untrusted mix-net. Thereby, seeds are provided to each mix without any information about the indexes so that the adversaries only have a partial view of the permutation process. For the user to remember as little information as possible, we could find seed combinations resulting in a simple result. However, because the symmetric group is not commutative and the seeds space is not structured, finding such combinations can only be done through heavy computation. That is why every eviction process will now be split in two phases, the unwrapping of the previous permutation and the wrapping to the next one.

2.5.2 Encryption

ORAM designs heavily rely on encryption mechanism to obfuscate the database records. Every record that is fetched must be re-encrypted before uploading it back to the database, which implies encryption cost of the order of $\mathcal{O}(n)$ for the eviction process and of variable orders, usually between $\mathcal{O}(1)$ and $\mathcal{O}(\log n)$, for the sole access.
Advanced Encryption Standard (AES) was conceived for high speed and low RAM requirements. It can present throughput over 700MB/s per thread on recent CPUs such as the Intel Core i3 or the AMD's models APU and FX. Furthermore, AES in Counter mode (AES-CTR) encryption and decryption operations are commutable and commutative. AES-CTR can thus help in the eviction process with re-encrypting the datablocks with a new key before decrypting the older encryption layer such that the plain text is never revealed.

3 Mix-ORAM

After looking at a simple but impractical Mix-ORAM we will look closely at the eviction process and the use of shuffle algorithms, we will then apply the result to a traditional client-server ORAM and finally to the mix-ORAM.

3.1 A simple Mix-ORAM

We consider here the basic case where the whole database is sent through a cascade mix-net during the eviction process as shown in Fig.1. Each of the mixes re-encrypts and permutes every record in a way such that the user only has to remember a constant amount of information, that is to say $\mathcal{O}(m)$ permutation seeds (S_i) and encryption keys (k_j).

To do so, because the symmetric group is not commutative, the mixes first undo the previous permutation (executing $\Pi_{S_m^{-1}\circ....\circ S_1^{-1}}$) before sorting the records (executing $\Pi_{S'_1\circ....\circ S'_m}$) to the new ordering. We also consider reversing the mix-net flow during the second step, the records now going from the last mix to the first before being uploaded on the database, to avoid unnecessary communication costs and private information transfer between mixes.
The security guarantees rely on the facts that no private

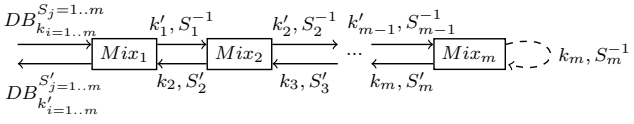

Fig. 1. A simple Mix-ORAM

information (seed, encryption key) is leaked between mixes and that the encryption is not deterministic.

Alongside the database records, the user need to send the encrypted seeds and re-encryption keys. Hence if we denote by x the size in bits of a seed and encryption key, the total communication cost is of the order $\mathcal{O}(m \cdot N)$, more precisely $C_m = 2mN + m \cdot (2m - \frac{1}{2}) \cdot x$. The computation cost is of the order $\mathcal{O}(mN)$, the encryption process comprising most of it.

This scheme is not efficient as instead of re-encrypting each record twice as in the Melbourne Shuffle, $2 \cdot m$ encryption per element is necessary to achieve perfect security. The whole database also has to be passed through every node twice which incur severe delays. Parallelization could help increase the efficiency of the mix-net by distributing the workload among mixes, however the cascade configuration would have to change. We can try to limit the number of seeds and encryption keys the user has to store thanks to reuse. However doing so, an adversary can discover in some cases, the probability depending on the number of compromised mixes and number of reuse, what permutations or encryptions was done by brute-forcing on a small universe of orderings.

3.2 Parallelizing the Eviction process.

In this section we present random transposition shuffles (RTS) and argue their use enable the eviction process to be amortized over time. We first present the mixing time of k-RTS before introducing ORAM assumptions to reduce even more the expected time to achieve randomness to the eyes of the adversary.

3.2.1 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 permutes locally k distinct cards, to make the scheme more efficient.

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*

$$E(\tau) < \frac{2}{k}\cdot\frac{n^2}{n+1}\cdot(\log(n)+\mathcal{O}(1))$$
$$\sigma^2(\tau) < \frac{4}{3}\cdot\pi^2\cdot\left(\frac{n}{k}\right)^2$$

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.

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

We can thus rewrite τ's expectation as following.

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

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

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,

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

. □

An oblivious k-RTS implies computation and communication cost of the order of $\mathcal{O}(n \cdot \log(n))$.

3.2.2 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, 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*

$$\tau(\epsilon) \geq \frac{1}{(\frac{n\cdot e}{s})^{\frac{s}{s-1}} - 1} \log\left(\frac{1}{2\epsilon}\right)$$

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

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:

$$log(\frac{k}{\lambda'^*}) = log(\frac{1}{\lambda^*}) \leq \frac{\log(v-1)}{\text{diam}(\mathcal{G}) - 1}$$

$$log(\lambda^*) \geq \frac{log(v-1)}{1 - \text{diam}(\mathcal{G})}$$

$$\lambda^* \geq (v-1)^{\frac{-1}{\text{diam}(\mathcal{G})-1}}$$

$$\lambda^* \geq \left(\binom{n}{s} - 1\right)^{\frac{1}{1-s}} \geq \left(\frac{n \cdot e}{s}\right)^{\frac{s}{1-s}}$$

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

$$\Delta = \frac{1}{(\lambda^*)^{-1} - 1}$$

$$\geq \frac{1}{\left(\frac{n\cdot e}{s}\right)^{\frac{s}{s-1}} - 1}$$

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 subgraph 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 $\frac{1}{1-\lambda*} \leq \frac{n}{2 \cdot s}$ □

Security Theorem 3. *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*

$$\tau(\epsilon) \geq \frac{1}{k} \cdot \frac{1}{(\frac{n \cdot e}{s})^{\frac{s}{s-1}} - 1} \log\left(\frac{1}{2\epsilon}\right)$$

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

3.2.3 Amortizable sort

We now consider the mix-net as a collection of isolated mixes. When performing an eviction, we attribute to each mix $k = \frac{n}{m}$ distinct indexes and provide them with lists of $\mathcal{O}\left(m \cdot \log\left(\frac{n}{s}\right)\right)$ of private and public seeds and encryption keys.

The unwrapping and wrapping phases consist of $\mathcal{O}\left(m \cdot \log\left(\frac{n}{s}\right)\right)$ rounds. In the unwrapping phase as shown in the Algorithm. 3.1, the records are located in the database with the old public seeds, inversely permuted with the old private seeds and re-encrypted with the new encryption key. In the wrapping phase as shown in the Algorithm. 3.2, the records from the attributed section of the database are download, permuted according to the new seeds, decrypted with the old encryption keys and uploaded to the database according to the new public seeds as shown in Fig. 2.

3.3 Parallel Mix-ORAM

TODO: eviction as in 3.2.3, new lookup, message format...

The client thus provides to each mix the *index* and range $k = \frac{n}{m}$ for the record allocation, $\frac{3}{2} \cdot m \cdot \log\left(\frac{n}{s}\right)$ seeds in total and $m \cdot \log\left(\frac{n}{s}\right)$ encryption keys and store in total $\mathcal{O}\left(m \cdot \log\left(\frac{n}{s}\right)\right)$ bits. To find the ORAM index of a precise record, the client performs the index lookup function written in Algorithm. 3.3.

Algorithm 3.1: Unwrapping process for the mix_i during $round_j$

Input: Seeds $old\_S_{pub,j}$, $old\_S_{prv,i,j}$;
Encryption key $new\_key_{i,j}$;
Number of records k;
First index $index$;

$Idxs \leftarrow \{\Pi^{-1}_{old\_S_{pub,j}}(range(index, index+k))\}$;
$Records \leftarrow \{\}$;
$i \leftarrow 0$;
while $i < k$ **do**
  $Records.append(get(Idxs[i]))$;
  $i \leftarrow i+1$;
$Records \leftarrow \{\Pi^{-1}_{old\_S_{priv,i,j}}(Records)\}$;
forall the $record \in Records$ **do**
  $record \leftarrow encrypt_{new\_key_{i,j}}(record)$;
$put\_range(index, index+k, Records)$;
del $old\_S_{pub,j}$, $old\_S_{priv,i,j}$;

Algorithm 3.2: Wrapping process for the mix_i during $round_j$

Input: Seeds $new\_S_{pub,j}$, $new\_S_{prv,i,j}$;
Encryption key $old\_key_{i,j}$;
Number of records k;
First index $index$;

$Records \leftarrow get\_range(idx, idx+k)$;
$Records \leftarrow \{\Pi_{new\_S_{priv,i,j}}(Records)\}$;
forall the $record \in Records$ **do**
  $record \leftarrow decrypt_{old\_key_{i,j}}(record)$;
$Idxs \leftarrow \{\Pi_{new\_S_{pub,j}}(range(index, index+k))\}$;
$i \leftarrow 0$;
while $i < k$ **do**
  $put(Idxs[i], Records[i])$;
  $i \leftarrow i+1$;
del $old\_key_{i,j}$;

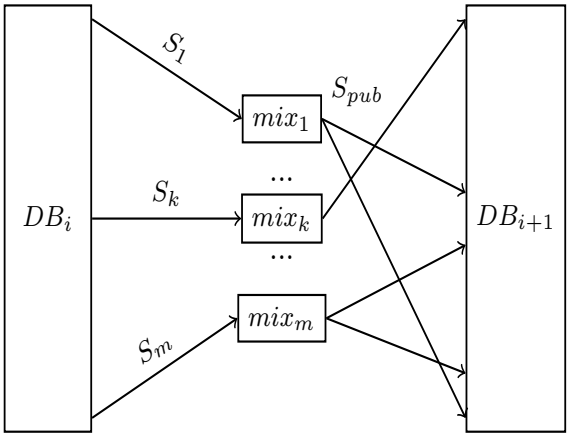

Fig. 2. Wraping in a round of Amortizable sort

Algorithm 3.3: Index Lookup

Input: Seeds new_S_{pub}, new_S_{prv};
Total number of records n;
Number of mixes n;
number of rounds r;
Index to retrieve $index$;

$k \leftarrow \frac{n}{m}$;
$i \leftarrow 0$;
while $i < r$ **do**
  $mix \leftarrow \frac{index}{k}$;
  $subindex \leftarrow index \ \% \ k$;
  $subindex \leftarrow \Pi_new\_S_{prv,mix,i}(subindex)*$;
  $index \leftarrow \Pi_new\_S_{pub,j}(k * mix + subindex)*$;
  $i \leftarrow i+1$;
Output: $index$
* These steps were simplified for easier reading.

3.4 Lowering Eviction Costs

In order to lower the eviction costs sustained by the mix-net, optimizations on each side of the channel can be done.

Raphael: subsections here or in discussion?

4 Evaluation

5 Discussion

- Ads and Cons of mix-nets
- Using proof of shuffle and proof of work to counter injection attacks.
- Using a reputation system alongside to dismiss mixes more quickly.
- Multi-user ORAM
- Lowering the eviction cost
- Fetching k items to postpone the eviction
- Differentially private oblivious shuffle

5.1 Mix-net: ads & cons

Two different methods can be applied for the eviction process depending on the mix-net architecture, in both cases eviction rely on repeatedly fetching k records, refreshing and permuting them before updating them back in the ORAM. If the mix-net is a cascade network, the first mix outputs the resulting array to next mix while repeating the process for distinct records,

otherwise, in the mesh case, the mixes fetch in parallel the records. The cascade case is more advantageous in time as the outputs and inputs of the mixes coincide, hence the communication cost is roughly divided by two. There is furthermore no synchronization needed between the mixes to process different records at the same time. However what is gain in duration is loose in resilience as any mix failure in the cascade mix-net would compromise the whole system. Moreover an attack specific to cascade mix-net also needs to be taken into account: if the first mix and the database are both corrupted, the permutation could be done only on a subset of records so that the adversary knows the positions of a non trivial number of records, as they were not permuted, without the honest mixes and the client realizing it. To counter it, the first mix query should be sent along the records to all the mixes together with an identification proof.

The computation cost of the eviction would thus be of the order of $m \cdot n \cdot \log(n)$ while the communication cost will be either $4 \cdot m \cdot n \log(n)$ or $2 \cdot (m+1) \cdot n \log(n)$ depending on the mix-net architecture.

5.2 Lowering the eviction costs

The client, to lower the eviction costs, could instead of fetching only the requested record query $k-1$ other dummy records. Doing so, we can advance further towards the uniform distribution before the eviction process, measure the distance between the two distributions and reduce the database shuffling cost by having the mix-net performing cheaper and weaker permutations. What's more, requesting more records can help in hiding the access pattern to the adversary has discussed in [25].

For instance if $n = 10^4$, $k = \log(n) = 4$ and the stash has a size of $\sqrt(n) = 100$, without considering fake accesses, we need 25 accesses before evicting the stash.

Considering fake access.

When a user wants to access a record which already is in the stash, a random database access is performed to hide potential frequency attacks: an adversary having observed the record access frequency beforehand and knowing the average access per day would be able to guess which records are in the stash by not observing accesses which should happen. These fake accesses can be considered as free permutations as no fetched records need to remain in the stash, thus we can arbitrarily set $k_{in} = k_{out}$.

Security Theorem 4. *In the case of a uniform distribution access we have an average number 0 of fake access for $s << n$.*

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:

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

□

However it is well known that the frequency access of records is not uniform.

Security Theorem 5. *Let A be the access distribution. Let's call v_0 and v_1 the volume above the line $y = 0$ and under it respectively. The average number of fake access is $\frac{v0}{v1}$.*

Proof. The volume under such line represents the number of real accesses a user does while the volume above it represents accesses in the stash which leads to fake accesses. When the number of accesses is great, the access frequency approaches the average number of accesses. □

5.3 Delaying the eviction

The client could replace elements of the stash when querying items to delay the eviction process, i.e. setting $k_in = k_out$.

6 Acknowledgement

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

7 Conclusion

- New ORAM with mix-net
- amortizable eviction

References

[1] Miklós Ajtai. Oblivious RAMs without cryptographic assumptions. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 181–190. ACM, 2010.

[2] Miklós Ajtai, János Komlós, and Endre Szemerédi. An 0(n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1983.

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

[4] Michael Backes, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Anonymous RAM.

[5] Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314. ACM, 1968.

[6] Dan Boneh, David Mazieres, and Raluca Ada Popa. Remote oblivious storage: Making oblivious RAM practical. 2011.

[7] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious RAM without random oracles. In *Theory of Cryptography Conference*, pages 144–163. Springer, 2011.

[8] Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. Three-server swapping for access confidentiality.

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[10] Martin Franz, Peter Williams, Bogdan Carbunar, Stefan Katzenbeisser, Andreas Peter, Radu Sion, and Miroslava Sotakova. Oblivious outsourced storage with delegation. In *International Conference on Financial Cryptography and Data Security*, pages 127–140. Springer, 2011.

[11] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 182–194. ACM, 1987.

[12] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[13] Michael T Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1262–1277. Society for Industrial and Applied Mathematics, 2010.

[14] Michael T Goodrich. 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*, pages 684–693. ACM, 2014.

[15] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious ram simulation. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 157–167. SIAM, 2012.

[16] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195. IEEE, 2005.

[17] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. The melbourne shuffle: Improving oblivious storage in the cloud. In *International Colloquium on Automata, Languages, and Programming*, pages 556–567. Springer, 2014.

[18] Rafail Ostrovsky. Efficient computation on oblivious RAMs. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 514–523. ACM, 1990.

[19] Michael S Paterson. Improved sorting networks with O(log N) depth. *Algorithmica*, 5(1-4):75–92, 1990.

[20] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Annual Cryptology Conference*, pages 502–519. Springer, 2010.

[21] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Ring oram: Closing the gap between small and large client storage oblivious RAM. *IACR Cryptology ePrint Archive*, 2014:997, 2014.

[22] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 247–258. ACM, 2013.

[23] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652*, 2011.

[24] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[25] Raphael R Toledo, George Danezis, and Ian Goldberg. Lower-Cost e-Private Information Retrieval. *Proceedings on Privacy Enhancing Technologies*, 2016(4):184–201, 2016.

[26] Sameer Wagh, Paul Cuff, and Prateek Mittal. Root oram: A tunable differentially private oblivious RAM. *arXiv preprint arXiv:1601.03378*, 2016.

[27] Peter Williams, Radu Sion, and Bogdan Carbunar. 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*, pages 139–148. ACM, 2008.