Technological University Dublin
ARROW@TU Dublin

Articles | School of Science and Computing

2021

Fedoram: A Federated Oblivious RAM Scheme

Alexandre Pujol

Liam Murphy

Christina Thorpe

Follow this and additional works at: https://arrow.tudublin.ie/ittsciart

 Part of the Computer Engineering Commons

This Article is brought to you for free and open access by the School of Science and Computing at ARROW@TU Dublin. It has been accepted for inclusion in Articles by an authorized administrator of ARROW@TU Dublin. For more information, please contact arrow.admin@tudublin.ie, aisling.coyne@tudublin.ie, gerard.connolly@tudublin.ie.

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 4.0 License

ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/346677565

FedORAM: A Federated Oblivious RAM Scheme

Article *in* IEEE Access · December 2020

DOI: 10.1109/ACCESS.2020.3027516

CITATION
1

READS
43

3 authors, including:

Liam Murphy
University College Dublin
216 PUBLICATIONS **2,707** CITATIONS
SEE PROFILE

Christina Thorpe
Technological University Dublin
40 PUBLICATIONS **271** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

MULE: Monitoring Utilisation of Large Environments View project

Robust Testing Environments View project

All content following this page was uploaded by Christina Thorpe on 07 December 2020.

The user has requested enhancement of the downloaded file.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 5, 2020, accepted September 13, 2020, date of publication September 28, 2020, date of current version October 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3027516

FedORAM: A Federated Oblivious RAM Scheme

ALEXANDRE PUJOL[1], LIAM MURPHY[1], (Member, IEEE), AND CHRISTINA THORPE[2]
[1]School of Computer Science, University College Dublin, Dublin 4, D04 V1W8 Ireland
[2]Technological University Dublin, Dublin 6, D06 F793 Ireland

Corresponding author: Alexandre Pujol (alexandre.pujol@ucdconnect.ie)

This work was supported in part by the Irish Research Council under Grant GOIPG2016479, in part by the Science Foundation Ireland to Lero—the Irish Software Research Centre under Grant 10CEI1855, and in part by the Science Foundation Ireland co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre under Grant 13RC2094.

ABSTRACT Instant messaging (IM) applications, even with end-to-end encryption enabled, pose privacy issues due to metadata and pattern leakage. Our goal is to develop a model for a privacy preserving IM application, by designing an IM application that focuses on hiding metadata and discussion patterns. To solve the issue of privacy preservation through the obfuscation of metadata, cryptographic constructions like Oblivious Random Access Machines (ORAM) have been proposed in recent years. However, although they completely hide the user access patterns, they incur high computational costs, often resulting in excessively slow performance in practice. We propose a new federated model, FedORAM, which is the first ORAM scheme that uses a federation of servers to hide metadata for an IM use case. In order to investigate the trade-off between security and performance, we propose two versions of FedORAM: Weak FedORAM and Strong FedORAM. Strong FedORAM uses a tree-based federation architecture to ensure strong obliviousness, but with an increased overhead cost. Weak FedORAM has a more simple federated architecture that only uses Oblivious Transfer (OT) to increase communication speed, but with security consequences. Our results show that both constructions are faster than a similar client-server ORAM scheme. Furthermore, Weak FedORAM has a response time of less than 2 seconds per message for a middle-sized federation.

INDEX TERMS ORAM, oblivious RAM, metadata, privacy, instant messaging, federation.

I. INTRODUCTION

Privacy is an ever growing concern in society; as more and more of our lives are lived online, protecting the privacy of our data and online activity is of utmost importance, particularly in sensitive contexts. When it comes to maintaining the confidentiality of our digital data, there are many existing cryptographic techniques that can be used very effectively. However, it is not only our data that needs to be protected; metadata, describing our online activity, can be observed on servers hosting the applications we interact with on a daily basis. This metadata can be gathered over time to infer sensitive knowledge about the users on the system.

With free licenses and usage, multi-platform Instant Messaging (IM) applications are becoming the first choice for many people wishing to communicate via text. While some of these popular applications have well-known and attractive security features, they are still vulnerable in terms of privacy attacks. For example, the security of well-known, popular Instant Messaging applications that use end-to-end encryption such as Signal or WhatApp is an incentive for many users. However, even though data are kept safe with the help of encryption when stored or in transit, an attacker or an untrustworthy service provider, with full access to the server, could still have many possibilities for obtaining sensitive information by looking at the metadata discussion patterns.

Indeed, such metadata are sensitive, and have become notorious in the scope of communications surveillance. Former NSA General Counsel, Stewart Baker, said *"Metadata absolutely tells you everything about somebody's life. If you have enough metadata, you don't really need content."*.[1] Our previous work in [37] and [35], defines different types of metadata and demonstrates the potential for metadata leakage in IM applications. Examples of the types of metadata that can commonly be collected on a classic IM server are the

The associate editor coordinating the review of this manuscript and approving it for publication was Ismail Butun.

[1]https://www.nybooks.com/daily/2014/05/10/we-kill-people-based-metadata/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

message *id*, *size*, *senders*, *recipients* and *timestamp*. We also showed that an ORAM construction is a valid solution for metadata leakage as it hides the connections between some of the metatada collected.

Oblivious Random Access Machine (ORAM) is a technique that allows clients to access encrypted data residing on an untrusted storage server, while completely hiding the access patterns to storage. Notably, the sequence of physical addresses accessed is independent of the actual data that the user is accessing. To achieve this, existing ORAM constructions continuously download, re-encrypt, and re-upload data blocks on the storage server. Consequently, ORAM schemes are limited in terms of complexity. In recent years, the goal of most research contributions was to reduce the complexity of the model in order to make it practical in real life [5]. Thus, in order to speed up and simplify the ORAM schemes, the structure of most server models has been changed [40], [43]. Therefore, as of today, ORAM studies focus on scheme complexity and security features like group support [25]. But these schemes remain impractical in real life applications. Moreover, to the best of our knowledge, there is nothing in the literature about ORAM implementations that have been adapted for a given use case such as IM. Most of the schemes proposed are for generic mass data storage or do not target a specific application.

There is a well known trade-off between security and performance: the more secure something is, the less usable it becomes. The overhead introduced by adding layers of security measures negatively impacts on the performance of a system and thus the usability of that system. *Security is a process of choosing between "less safe" and "more safe" and continuing to fork toward safety until you read safe enough.* Edward Snowden.[2] When considering the construction of a secure application this is particularly accurate.

Finding the balance between 'secure enough' and 'practically usable' is critical. Considering the existing work in the literature, the main motivation of this paper is to create an ORAM scheme that is practical in terms of performance for a given use case. Therefore, we need to find the trade-off between privacy and speed in order to design new oblivious algorithms. In our quest to achieve this, we address the following research questions:

- What infrastructure design for an IM application will result in a faster ORAM scheme?
- Can we define a new ORAM construction that only slightly decreases the privacy protection in order to increase the speed?
- Can a multiple server architecture be used to help protect against metadata leakage?

In order to answer these questions, we built a prototype ORAM IM server and implemented two novel federated ORAM algorithms to explore the trade-off between obliviousness and performance. That is, the strong oblivious mode is more secure but slower and the weak oblivious mode is less secure but faster. This work compares the two schemes proposed in this paper together and against two reference schemes: A non-ORAM federated IM application and Square Root ORAM (SqrORAM) [46] a multi server ORAM system. Comparisons are done considering response time and network consumption metrics. Our results show that both FedORAM schemes can support reasonable response times, and can be considered more practical for use in real-world IM applications than classic ORAM schemes.

The contributions of this paper can be summarised as follows:

1) To the best of our knowledge, we are the first to propose a federated ORAM scheme.
2) We propose a novel ORAM model for an IM application. More specifically, an ORAM scheme adapted for small message sizes (less than 1KB) with an automatic deletion policy for old messages.
3) We define the notion of 'Weak Obliviousness' that leaks *some* metadata, but does not provide an attacker with enough to infer a full access pattern. Using this definition, we propose a new ORAM scheme, that takes advantage of the trade-off between privacy and performance, resulting in a faster scheme.

The remainder of the paper is organised as follows: Section II discusses the related work in the area. Section III proposes the definition of Weak and Strong Oblivious RAM as well as the attacker model used in this paper. Section IV describes the two ORAM systems proposed. Section V describes and discusses the experimental results. Finally, Section VI concludes the paper and presents some future work.

II. RELATED WORK

This section describes the previous work that has been published in the area. To the best of our knowledge, this is the first work to propose Oblivious RAM for a federation of servers.

Landau [23] highlights the importance of metadata in Edward Snowden's revelations on the security world. For instance, this paper shows that the NSA is now known to collect domestic and telephony metadata. Furthermore, Mayer *et al.* [28] evaluates the privacy properties of telephony metadata and show how applying prediction on collected metadata, we can retrieve sensitive knowledge such as location, relationship status, and health status. On another application, but still considering metadata, Dubin *et al.* [13] show how video identification can be achieved from an encrypted multimedia stream like YouTube. Finally, Hoang *et al.* [20] show how, using a simple trilateral method, it is possible to retrieve user positions on famous LGBT-focused dating applications, which poses enormous privacy issues.

Protection against data privacy threats can be achieved with data encryption techniques, data sanitisation [2] or data anonymisation [4]. More generally, various techniques to ensure data privacy is maintained [44]. However, since these techniques do not protect against metadata leaks

[2]https://twitter.com/Snowden/status/1165391070726950913

(as demonstrated in our previous work [35]), a significant amount of work has been done in order to provide access privacy for outsourced data. This can be achieved by means of Private Information Retrieval (PIR) [9], [17] or by means of Oblivious RAM. PIR and ORAM are similar in the way that a PIR scheme can be seen as a read-only ORAM scheme. PIR is, therefore, suitable for a different use case. Still, many ORAM schemes include techniques that are similar to PIR.

Preventing the leak of data using an oblivious structure between a trusted processor and an untrusted RAM was presented for the first time in 1996 by Goldreich and Ostrovsky [18]. They were the first to introduce the notion of Oblivious RAM. The purpose of this technique was to hide the true access pattern of the software on the local memory. Later, the system was extended for cloud storage purposes [5], [40]. Then, the goal of most research contributions was to decrease the complexity of the models in order to make them practical in real life. Thus, the techniques used were changed. The structure of most models changed from tree based models, where the server structure is a binary tree [30], [39], [40], [42], to Path ORAM based system [43], [45]. Furthermore, given that executing all computation on the client side is not efficient and since the Cloud has the advantage of providing computational resources in addition to storage, ORAM with server side computation was proposed [3], [10], [26], [46]. These schemes use a fully Homomorphic Encryption (HE) function [16] resulting in the computation shrinking from logarithmic in client-side to constant in server-side computation model. However, server-side computation suffered from significant communication overhead. Ring-ORAM [39] reduced the cost of communication in Path-ORAM significantly, by performing XOR computations on the server. Some other alternatives ([3], [11], [14], [27], and [32]) leveraged single-server PIR or fully/partial HE to further reduce the communication cost. For example, Onion ORAM [10] is a layered encryption approach that aims to reduce the communication overhead incurred in the oblivious storage process using bandwidth-efficient additive HE schemes. Meanwhile, some recent systems support sharing features, e.g., Group ORAM [25] or Obliv P2P [22]. They introduce some additional concerns; new parameters such as access rights management must be considered. Finally, Dog ORAM [36], proposes an ORAM scheme that merges some existing solutions, a Path ORAM based server with server-side computation and user management.

All these schemes propose innovations in terms of complexity or security features. They provide proof of security through a formal proof of obliviousness of the protocol proposed. However, they do not compare their security with other systems or with a non-ORAM server.

Since ORAM solutions present a complexity problem, there have been a number of studies conducted to examine the necessity of this scheme. On searchable encryption, Naveed *et al.* [33], [34] present inference attacks on searchable encrypted databases secured with an ORAM model. They show these systems still leak metadata. This is because ORAM models are secure only when the data is accessed through a random access interface. For an application such as searchable encryption, that does not fit this requirement, ORAM schemes are not a solution. However, Pulls *et al.* [38] show ORAM is needed for other types of applications. Their paper introduces and discusses the feasibility of a practical anonymous cloud storage service that includes an anonymous payment method. It especially shows that long term anonymous cloud storage can be achieved using an ORAM system. Moreover, the authors confirm ORAM methods are an open research challenge.

In an attempt to improve the performance and scalability of ORAM to make it more practical, several proposals have been made to eliminate the extremely costly (full/partial) HE operations by implementing multi-server models. Stefanov *et al.* [41] were one of the first to propose a multi-server ORAM scheme that utilises two non-colluding servers, capable of computation, in order to decrease the bandwidth usage between the client and server in Partition ORAM [42]. Moataz *et al.* proposed CHf-ORAM [31], which has constant communication complexity without using HE, achieved by moving from the traditional single-server to 4 non-colluding servers. However, the work in [1] demonstrated how to break CHf-ORAM [31]. Meanwhile, [1] also proposed a new model, consisting of two non-colluding servers, to perform XOR computations for block retrieval over a k-ary ORAM tree structure. In 2018, Chan *et al.* [6] proposed a secure 3-server ORAM scheme based on the original hierarchical ORAM structure proposed by [18]. Gordon *et al.* [19] proposed a simple and efficient 2-server tree-based ORAM, with low bandwidth overhead. A static position map is implemented and references deterministic and static data block paths, which are computed using a pseudo-random function. Hoang and Yavutz [21] proposed the multi-server ORAM model S^3, which leverages Shamir Secret Sharing and a multi-party multiplication protocol on applicable binary tree-ORAM paradigms to avoid the costly complexity of HE or other cryptographic primitives.

Conceptually, in an attempt to increase the speed of ORAM, because classic Path ORAM schemes are limited to a lower bound [24], some researchers looked to a multi-server approach. For example [7] uses S2PC to store part of the data on two non colluding servers. However S2PC comes with an important cryptographic cost. Metal ORAM [7] is the first cryptographic file sharing system that hides both user identities and file access patterns from both the server and from malicious users. Metal uses a tree-based ORAM structure and has logarithmic worst-case complexity and considerably less computation. It avoids the linear worst-case times that is experienced by SqrORAM [46] and FLORAM [12]. This setup is very interesting because it allows the reduction or elimination of the trust requirements present in a single server system. The ORAM literature contains very strong evidence showing that the lack of trust in a server comes with a very high performance cost in terms of cryptographic primitives needed to address it.

Oblivious Transfer (OT) [29] is a cryptographic protocol that allows a sender to transfer one (or more) message to a receiver. The sender remains oblivious as to what piece (if any) has been transferred. It differs from ORAM because in ORAM the sender is trusted while the receiver is not trusted. When used with elliptic curve cryptography, OT schemes are fast and scalable [8], [15].

Our work presented in this article is the first to propose a federation of servers to provide oblivious access for users. Moreover, we consider an IM application, which has specific properties and performance requirements when compared to a generic mass storage use case seen most often in the literature. Finally, we also investigate the potential for leveraging the trade-off between security and performance in order to provide a faster, but less secure ORAM model.

III. OVERVIEW OF OUR CONSTRUCTION

This section describes FedORAM's system architecture and threat model as well as the definitions for some pertinent concepts discussed in the paper.

A. FedORAM

The concept of a trade-off between security and performance motivated our design of multiple federated ORAM models. We created two different constructions in order to compare them in terms of security and speed:

1) **Strong Federated ORAM (Strong FedORAM):** A federated ORAM scheme where both clients and servers are fully compliant with the classic ORAM definition (see definition 4).
2) **Weak Federated ORAM (Weak FedORAM):** A federated ORAM scheme that by construction is designed to provide a weaker protection against metadata leaks than the strong federated ORAM scheme. However, it aims to be faster (see definition 3).

Outside the context of ORAM, we are not the first to consider such a federation, the most famous example being the email system. Although it has well known drawbacks such as spam detection issues, a federation based scheme has several advantages over a classic client-server architecture:

- It provides fault tolerance as a server can be removed without stopping the rest of the federation.
- It provides a protection by default as only the source and destination server can collect data. Furthermore the source server is often chosen by the user and therefore is more trusted than other servers.
- Multiple service providers can easily add and remove a server on the federation without trusting each other.

As shown in definition 1, FedORAM is a new federated infrastructure, that requires new server types to play different roles when sending messages.

Definition 1 (Server Types):

- **Entry server** S_E: The entry server is the first server the data sent by a user will meet. It is the only interface an end-user has access to within the federation.
- **Destination server** S_D**:** The destination server is the last server the data sent by a user will meet. Data stored on this server will be obliviously collected by the destination user.
- **Third party server:** The other servers in the federation that are neither the transfer server nor a destination server, but that can have a role in the data transaction as an intermediary.

B. FEDERATED OBLIVIOUSNESS

Definition 2 is as a reminder of the security property of obliviousness. However, this paper focuses on oblivious constructions in the scope of a federated architecture. We define a new security property: federated obliviousness. Definitions 3 and 4 are our new proposed definitions for strong and weak federated obliviousness, which also consider the accesses from a user and between servers inside the federation.

Definition 2 (Obliviousness): A scheme is oblivious if the server cannot distinguish between two accesses which contain any kind of authorised operation of the users.

Definition 3: (Weakly Federated Obliviousness): A scheme is weakly federated oblivious if and only if all the following conditions are meet:

- *An entry/destination server knows its type for an access.*
- *A destination & third party server cannot distinguish between two accesses which contains any kind of operation the users have the rights to execute on the server (at the exception of the access type: push/pull).*
- *An entry server cannot distinguish between a destination and a third party server.*
- *An external attacker could distinguish an entry server.*

Definition 4: (Strongly Federated Obliviousness): A scheme is strongly federated oblivious if and only if all the following conditions are meet:

- *An entry/destination server knows its type for an access.*
- *A servers (regardless of type) cannot distinguish between any two accesses which contain any kind of operation the users have the rights to execute on the server (at the exception of the access type: push/pull).*
- *An entry server cannot distinguish between a destination and a third party server.*
- *A destination server cannot distinguish between an entry server and a third party server.*
- *An external attacker cannot distinguish between any servers.*

C. METADATA

Although the notion of metadata is common as per definition 5, in the scope of data privacy this high-level definition is not sufficient. Definition 6 shows that we need to consider the different types of metadata that can be leaked by a server. All data have a specific format (e.g., file, message), but it can have multiple distinct sets of metadata, each describing a different property of the data. Each different way of describing data is called a 'type' of metadata.

TABLE 1. Metadata leakage depending on the scheme.

Scheme	Metadata Type						Server Type
Classic Server	id	size	senders	recipients	time		
Oblivious RAM					time		
Weak FedORAM			senders		time		entry server
				recipients	time		destination
Strong FedORAM					time	leaf	

Definition 5 (Metadata): Metadata is defined as the data that provides information about other data.

Definition 6 (Type of Metadata): When the information collected about different data describes the same property, this information is from the same type of metadata.

As previously described in [35], the types of metadata that can commonly be collected on a classic IM server are the message *id*, *size*, *senders*, *recipients* and *timestamp*. Furthermore, it is the link between some of them (*id*, *senders*, *recipients*) that is the root cause of privacy leaks. As presented in table 1, breaking this connection by only leaking the message timestamp is the main objective of an ORAM solution. By definition, on a classic ORAM scheme (like Path ORAM), only the activity timestamp is leaked. In a less secure scheme, such as our proposed Weak FedORAM, the objective is to leak a little more metadata so as to improve the performance, but not so much that an attacker could use it to breach the users' privacy. For example, we leak the identities of the senders or the recipients to two different non-colluding servers. In Strong FedORAM, we only leak the path taken by the message on the federation tree.

Usually ORAM schemes maintain a symmetry in the system. A read action has a corresponding write action. Meaning they first read a possibly dummy data then they write a (possibly dummy) data. It provides read/write obfuscation and it is extremely valuable in ORAM construction. However, this is usually slow. Even more so considering a path structure, where a write can be fast, but the read is slow. Therefore, to speed up the system, the approach used in FedORAM is to break this symmetry. Thanks to the federated structure, the lack of read/write symmetry does not come with much privacy drawback.

D. ATTACKER MODEL

1) SECURITY ASSUMPTIONS

Our adversarial model draws from common assumptions in real life and in the ORAM literature. They are justified in real-life cloud architectures because service providers need to maintain a good reputation:

- The data owners are always trusted.
- The other clients are not especially trusted. However, they do not collude with the server. This assumption makes sense because it would be against the purpose of the application for a malicious client to voluntarily send metadata to the server.
- The servers are considered "honest but curious"[3] (HbC).
- We assume the third party servers in the federation can be corrupted and therefore we do not trust them. They remain HbC.
- An entry or destination server knows its status. Meaning it knows it is an entry or a destination server for a given message id.
- The servers are not colluding.

Furthermore, the Weak and Strong FedORAM schemes have some differences in terms of attacker model. With Strong FedORAM, the entry server and the destination server are oblivious to each other. While for Weak FedORAM:

- Destination and entry servers are not colluding.
- An external adversary (or some colluding servers) can guess general access pattern of the federation.

2) SECURITY GUARANTEES

Both FedORAM schemes provide the following security guarantees:

- **Secrecy:** A client can only read messages for which they hold read permissions.
- **Integrity:** A client can only write messages for which they hold write permissions.
- **Authentication:** A client can only access their messages after the server has verified their identity.
- **Obliviousness** for the considered scheme as shown in definitions 4 and 3.

We do not consider forward secrecy and data sharing as we really focus on the creation of the first federated ORAM scheme.

IV. FedORAM

This section describes the Weak and Strong FedORAM schemes.

A. COMMON STRUCTURE

1) SERVER STRUCTURE

Each individual server stores data into a tree. The structure is similar to the PathORAM protocol. The data is stored in a binary tree of $L_S + 1$ levels (see figure 1). The leaves are numbered from 0 to $2^{L_S} - 1$. Each block is associated with a random path from the root to a leaf l: $\mathcal{P}(l)$. Notations used throughout the rest of the paper are summarised in table 2.

When a new server enters the federation, it must generate a new key pair and maintain a list of public keys from all the servers in the federation. All the users also have they own key pair.

2) OBLIVIOUS TRANSFER

By definition, an ORAM construction sends data from a trusted client to an untrusted server. However, when employ-

[3]The servers are regarded as passive adversaries following the scheme specifications correctly but seeking to gather additional information about the clients' data and access patterns.

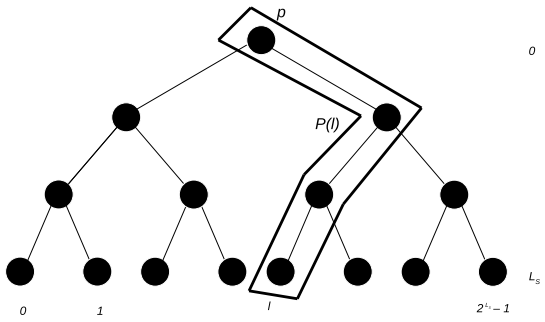

FIGURE 1. Tree data structure on each server of the federation.

TABLE 2. FedORAM parameters and notations.

Notation	Meaning
m	Message
M	Message block sent by a user
c	Number of clients
s	Number of servers
id	Virtual message id (dummy or real)
L	Depth of the Federation tree
L_S	Server internal data tree depth
l	Federation path from the root server to a leaf
$\mathcal{P}(l)$	Path from the root to the leaf l

ing a federated architecture, there are a minimum of two servers to consider:

- The destination server, from where the recipient user will pull new messages.
- The entry server, to which the sending user pushes the message.

The new challenge of a federated oblivious scheme is to be able to obliviously transfer data from the entry server to the destination server.

A way to achieve this is to use Oblivious Transfer (OT) [15], [29]. Despite its name, OT is very different to ORAM because OT and ORAM do not have a similar attacker model. As shown in figure 2 using OT, the receiver knows what message id is sent, not the sender. However, OT schemes cannot be directly applied to our new FedORAM construction. Therefore we needed to adapt a $1-2$ OT to use in FedORAM.

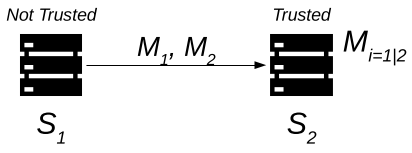

FIGURE 2. Oblivious transfer.

Our remote oblivious transfer works as follows: a client generates a message block M. It contains the real message the client wants to send but also extra parameters that will be needed by the entry server to run the OT protocol. Let a client C_1 send a message m to a user C_2. Let S_E be the entry server for C_1, S_D the entry server for C_2 and thus the destination server for C_1. S_d denotes a dummy third party server in the federation. Then, in both Weak and Strong FedORAM, C_1 generates M as follows:

$$M = \begin{cases} (M_1, S_D) \\ (M_2, S_d) \end{cases} \tag{1}$$

We construct M_1 and M_2 as follows:

$$M_1 = \begin{cases} (\mathcal{E}(m), E_{S_D}(S_D)) \\ (\mathcal{E}(d), E_{S_D}(S_D)) \end{cases} \tag{2}$$

$$M_2 = \begin{cases} (\mathcal{E}(d), E_{S_d}(S'_d)) \\ (\mathcal{E}(d), E_{S_d}(S'_d)) \end{cases} \tag{3}$$

where d are dummy random messages of the same size as m. E and $\mathcal{E}$ are encryption functions. To simplify the notation, $\mathcal{E}$ uses a client public key and therefore only a client can decrypt it while E uses a server public key such that E_{S_D} denotes an encryption function using the public key of S_D. Finally, S'_d is another dummy third party server different to S_d.

Using this structure, when a client sends M to an entry server, M_1 and M_2 are encrypted and the entry server cannot decrypt them. Nevertheless, it knows S_D and S_d where it can send the message using OT.

3) EVICTION PROCESS

Each server stores data in a classic oblivious tree structure as used in PathORAM. When adding a new message to the server, the data is always written on the root node of this server. Therefore, every time the root node of a server is full, the server must run an eviction algorithm in order to evict the messages from the root node into the children nodes to ensure it does not overflow. This process is usually long and contributes significantly to the overall complexity of an ORAM system.

As in PathORAM, the eviction process is executed by a client if needed when pulling new messages from the server. Due to some design features, in FedORAM, this process is faster:

- As it is a scheme for an IM use case, we included an additional feature called *automatic data deletion*, which simply deletes old messages. This deletion is done by the eviction algorithm that ignores the last stage of the path and considers them as having become dummy data. Therefore, less data needs to be evicted.
- The oblivious tree of each server is independent of the federation.

On each server, the eviction algorithm ensures that the root node is not full. This is needed as new messages are always written on the root node. On a pull request, a client runs the eviction if it is required. That is to say, if the root node of its entry server is full.

The user selects a random leaf l_{evict} on which the eviction will be run. For all considered nodes on the path, the messages are moved into its two children nodes. It is the same eviction algorithm as PathORAM. Therefore, by construction, old messages will be on the last levels of the tree.

To speed up the eviction process, we do not evict the messages on the last e nodes as the messages in these nodes are not shuffled. e, the number of levels from the leaf to not evict, is determined randomly between L_l, the local tree depth and $D < L_l$, a configuration parameter that determines how fast old messages are deleted.

B. WEAK FedORAM

Figure 3 shows the general architecture of the Weak FedORAM model when a client C_1 sends a message to another client in the server S_D. Weak FedORAM can be summarised as follows:

- A set of s independent servers S_i, $1 \leq i \leq s$.
- A root server S_R.
- Each server has its own set of clients C_i, $1 \leq i \leq c$. Each client as a unique ID in format $C_i@S_i$
- A set of algorithms as shown in definition 7.

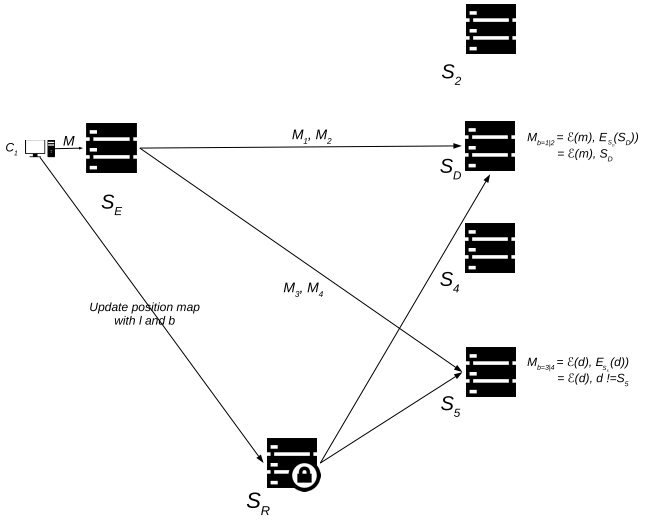

FIGURE 3. Overview of the weak FedORAM scheme.

Definition 7 (Weak FedORAM Scheme): *A Weak FedORAM Scheme is a set of the following algorithms*:

$M \leftarrow Push(id, m, user)$ Send a message to a user on a server. Generate and send the message block to be transferred to the destination server using OT from the entry server

$m \leftarrow Pull(id)$ Pull new message from the client's entry server.

$Serve(id)$ Transfer/Receive a client message to the federation using OT.

$Eviction(leaf, forget)$ Start the eviction algorithm on the entry server.

1) OVERVIEW

As shown in figure 3, when a client C_1 sends a message to another client in S_D, the client generates M as shown in equation 1. Then C_1 sends this message block to its entry server S_E. S_E knows the destination servers it has to send data to (S_D and S_d). Thus S_E runs two OT transactions to S_D and S_d using M_1 and M_2. The real and dummy destination servers both receive their message block sub-part, M_1 and M_2 respectively. Both servers decrypt $M_{i=1|2}[1]$ and can compare their own server id against it. If they are equal, the server is a real destination server and it writes $\mathcal{E}(m)$ on its internal root node. If they are not equal, the server ignores the transaction.

$$\begin{aligned} M_1 &= (\mathcal{E}(m), E_{S_D}(S_D)) \\ &= (\mathcal{E}(m), S_D) \\ M_2 &= (\mathcal{E}(d'), E_{S_d}(S_{d'})) \\ &= (\mathcal{E}(d'), S_{d'}) \end{aligned}$$

2) PositionMap

Internal metadata management is a critical feature of all ORAM models; metadata allows the operations on the remote messages. In a similar way to other ORAM schemes, we use a *PositionMap* matrix that we store on the root server. This matrix links the virtual id id of a (real or dummy) message to its assigned leaf on the server destination server. $PositionMap[id]$ gives a leaf l indicating that the message with the id id can be found in the destination server on the path $\mathcal{P}(l)$.

In FedORAM, it is the sender of the message that generates the random leaf l and appends it to *PositionMap* on the root server. There is no indication of the destination server in the matrix.

3) OTMap

Similarly to the *PositionMap*, FedORAM needs to store an oblivious transfer matrix on the root server. It links the virtual id id of a message to the oblivious transfer identifier b that tells which (if any) message id of the real message that has been transferred. b is encrypted such that only the destination server can read it. $OTMAP[id]$ gives an identifier b.

4) ALGORITHMS DESCRIPTION

In this paragraph we describe the main Strong FedORAM algorithms. Both *Push* and *Pull* algorithms run on clients. The OT transactions that run on the servers have already been described in previous paragraphs.

Push In algorithm 1, the client generates the message block M and encrypts it using the public key of the recipient. It also generates the leaf of the destination server as well as the OT selection identifier for S_d and S_D. Then, M is sent to the entry server S_E that will proxy it.

Pull In algorithm 2, the client reads and decrypts the message using the same algorithm as in PathORAM. If needed, the eviction is started as shown in section IV-A.

5) LIMITATION & CONCLUSION

This scheme is quite different from a classic ORAM scheme as it uses mostly direct connections to servers in the federation and does not require extra costly computational resources. However this efficiency comes at a security cost. While it

Algorithm 1 $M \leftarrow Push(id, m, user)$

Input: virtual message id, m plain message, $user$ federation user.
Output: message block M

$M \leftarrow GenereateMessageBlock(m)$ ▷ As shown in (1)
for all server in (S_d, S_D) **do**
  for all message id in M_i **do**
    $OTMap[id] \leftarrow b$ ▷ Set OTMap according to M
    $PositionMap[id] \leftarrow GenLeaf()$ ▷ Local leaf for id
  end for
end for
return M ▷ Message block to be sent to S_E

Algorithm 2 $m \leftarrow Pull(id)$

Input: virtual message id
Output: plain message m

m $\leftarrow PathORAMAccess(id)$
if Eviction **is** needed **then**
  $l_{evict} \leftarrow Random(0, 2_l^L - 1)$
  $forget \leftarrow Random(D, L - 1)$
  $Eviction(l_{evict}, forget)$
end if
return m

is able to protect against some metadata leakage, it remains sensitive to some access pattern attacks that could leak the entry and destination server, but not the sender and receiver. The solution to this privacy issue is to ensure the messages are always transferred through the same server regardless of the originating server. As a solution we propose Strong FedORAM, that takes inspiration from the well-known tree based path ORAM oblivious data structure on a single server and adapts it to a federation of servers.

C. STRONG FedORAM

Figure 4 shows the general architecture of the Strong FedORAM model when a client C_1 sends a message to another client in the server S_D. Strong FedORAM can be summarised as follow:

- A tree based federation of servers.
- A set of s independent servers S_i, $1 \leq i \leq s$, such that $s = 2^L - 1$.
- Each server has its own set of clients C_i, $1 \leq i \leq c$. Each client as a unique ID in format $C_i@S_i$
- A set of algorithms as shown in definition 8.

Definition 8 (Strong FedORAM Scheme): A Strong FedORAM Scheme is a set of the following algorithms:

$M \leftarrow Push(id, m, user)$ *Send a message to a user on a server. Generate and send the message block to be transferred to the destination server using OT from the entry server*
$m \leftarrow Pull(id)$ *Pull new messages from the entry server.*

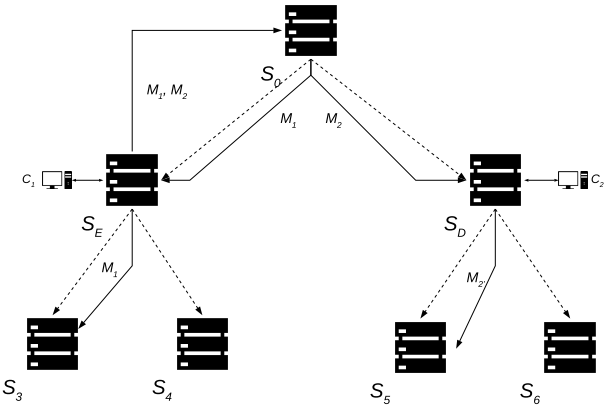

FIGURE 4. Overview of the strong FedORAM scheme.

Proxy$(id, leaf)$ *Proxy message from a federation server towards a leaf.*
Eviction$(leaf, forget)$ *Start the eviction algorithm on the entry server.*

1) FEDERATION STRUCTURE

The federation structure we propose follows the structure of a binary tree. Each node is a server in the federation. Each server can store up to n messages in their own ORAM data structure, similar to Path ORAM. All these data structures are independent of each other. In a similar way to Path ORAM, each message sent from a client to an entry server is associated with a path from the root server to a leaf l: $\mathcal{P}(l)$. The client selects l such that the destination server is on $\mathcal{P}(l)$.

2) ACTIVITY

We keep the OT structure presented in Weak FedORAM, however, we adapt it to a tree based federation. As shown in figure 4, when a client C_1 sends a message to another client C_2 in S_D. Let l_d be a random dummy leaf in the federation tree. The client generates M as shown in equation 4.

$$M = \begin{cases} (M_1, l) \\ (M_2, l_d) \end{cases} \tag{4}$$

Then, in order to successively transfer the message to the paths, M_1 and M_2 are generated in the following layered form:

$$M_1 = E_{S_R}(\ldots E_{S_{D-1}}(E_{S_D}(\mathcal{E}(m), S_D), S_d), \ldots, S_d) \tag{5}$$
$$M_2 = E_{S_R}(\ldots E_{S_{d-1}}(E_{S_d}(\mathcal{E}(d), S_{d'}), S_{d'}), \ldots, S_{d'}) \tag{6}$$

This layered construction ensures that:

- Only the destination user can decrypt it: $\mathcal{E}(m)$
- Only S_D can decrypt it: $E_{S_D}(\mathcal{E}(m), S_D)$

Then C_1 sends this message block to its entry server S_E and updates the *PositionMap* matrix and the *OTmap* matrix in the root server S_R. S_E the entry server runs two OT transactions with the first element of M_1 and M_2 to the root server. Then the root server runs two other OTs into the federation tree to the next server in the federation that belongs to the leaf l_d and l respectively for M_1 and M_2.

Then, for any third party server, the server checks if it is the destination server using the same method as in Weak FedORAM. If it is the destination server, it stores the message in its internal root node and replaces the message with a random dummy one before forwarding it to the next server in the path. If it is not the destination server, it simply forwards the message to the next server in the path.

With this construction, the users only connect through their entry server. Regardless of the destination server, the entry server always sends the messages to the root server. The root server itself sends the message back into the federation leaf. This structure takes inspiration from the eviction process on a single server Path ORAM scheme where the data are evicted to two paths, a real one and a dummy one.

Strong FedORAM uses the same *PositionMap* and *OTMap* matrices structure as Weak FedORAM. The only difference is that these matrices are stored on the root server, not a server outside of the federation.

3) ALGORITHMS DESCRIPTION

In this paragraph we describe the main Strong FedORAM algorithms. The *Proxy* algorithm is the only algorithm that needs to be run on a server of the federation. As *Push* and *Pull* are client algorithms. The *Pull* algorithm is the same as in WeakORAM.

Algorithm 3 $M \leftarrow Push(id, m, user)$

Input: virtual message id, m plain message, $user$ federation user.

Output: message block M

1: $(l, l_d \leftarrow GenLeafs(user)$ ▷ Generate random federation leafs
2: $M \leftarrow GenereateMessageBlock(m)$ ▷ As shown in (4)
3: **for all** *leaf* in (l, l_d) **do**
4: **for all** message id in M_i **do**
5: $OTMap[id] \leftarrow b$ ▷ Set OTMap according to M
6: $PositionMap[id] \leftarrow leaf$
7: **end for**
8: **end for**
9: **return** M ▷ Message block to be sent to S_E

Push Algorithm 3 is similar to algorithm 1. The client generates the message block M, it also generates the federation and local leaves as well as the OT selection identifier for all servers in the leaves. Then, M is sent to the entry server S_E that will proxy it.

Proxy The proxy algorithm 4 allows a server to obliviously transfer a message into the federation. S denotes the server id on which the algorithm is run. First, regardless of the server type, the algorithm retrieves the internal storage leaf from the metadata matrix. It is only needed for the destination server, however, querying the network only after the server determines its state would leak it to the federation. Then, the message block is locally decrypted into a tuple (M_l, S_l) where M_l is the next message block (or m for the destination server) and S_l is a (possibly dummy) server id. *OTSend* and *OTReceive*() are the OT send and receive functions.

Algorithm 4 $Proxy(id, l)$

Input: Virtual message id (possibly dummy), l federation leaf.

1: $l_S \leftarrow PositionMap[id]$ ▷ Get the internal storage leaf.
2: $b \leftarrow OTMap[id]$
3: $M \leftarrow OTReceive(b)$ ▷ Retrieve message block from parent node
4: $M_l, S_l \leftarrow \mathcal{D}(M)$
5: **if** $S_l == S$ **then** ▷ If destination server
6: $\mathcal{P}(l_S) \leftarrow M_l$ ▷ Write M_l on S.
7: $M_l \leftarrow GenDummy()$ ▷ Dummy block for other servers in l
8: **end if**
9: $S_n \leftarrow GetNextServerOnLeaf(l)$
10: $OTSend(M_l, S_n)$ ▷ Proxy to next server

D. SECURITY ANALYSIS

This section proposes an analysis of the metadata leakage in the federation. It focus on the obliviousness of the scheme.

1) CREATING A MESSAGE

For both schemes, when a client creates a new message, the metadata types present on the client are the *id*, *size*, *senders*, *recipient* and the *timestamp*. M, contains the encrypted message, but no other information. Particularly, the destination server identifier is encrypted. Both *PositionMap* and *OTMap* store metadata for both real and dummy message ids, moreover, OTMap entries are encrypted. The server that stores these matrices is trusted for this usage as it can see which user is active. This is standard behaviour in the ORAM literature.

2) PUSHING A MESSAGE

When M is transferred to the root server. The entry server knows its server type and knows the client is active at a given time. However, as it can only see M and run an OT transaction with it, it does not have access to: the chain of discussion, the message *size* and the *recipient*.

3) FEDERATION

Weak FedORAM presents limitations already described in section IV-B5. Concerning Strong FedORAM, we can state:

- Entry server: Regardless of the destination server (even if it is the same server), M is transferred to the root server. Therefore, no other metadata other than a timestamp is leaked to the server. Only the root server knows the role of the entry server.
- Root & Third Party servers: They only have access to the leaf number. As two message streams are sent to two leaves, no real information can be inferred from this.

- Destination server: knows its state, and therefore could potentially deduce the *recipient*.
- External attacker: as the destination server still proxies a dummy message to deeper nodes in the path, an external attacker observing activity in the federation cannot distinguish between the dummy and the real leaf. Therefore, only the destination server knows its role.

4) PULLING A MESSAGE

When a client wants to pull a message from its entry server, it first reads the *PositionMap* matrix to get the local path l_L and uses it to download data from the server. As in PathORAM, no metadata is leaked in this transaction.

V. EXPERIMENTS

In this section, we present the experiments we designed in the scope of this paper. We tested Weak and Strong FedORAM against each other but also against a classic federated IM server as reference base. We also compared it against a well known ORAM construction in the literature.

A. EXPERIMENTAL SETUP

We used a bare metal server to run the federation. The server had two *Intel Xeon E5-2630 v4 @ 2.20GHz* CPU with 64 GB of RAM. Each server ran in a docker container. Both the host server and the docker containers ran on Debian 10. We implemented Weak & Strong FedORAM in python. The network communication is handled by aiohttp[4] while cryptographic functions were provided by *cryptography*.[5] As a reference non-ORAM server, we implemented a simple IM federated system to compare against Weak & Strong FedORAM. For all schemes, the sent messages was always a 1KB random message. Due to the requirement of Strong FedORAM to have the number of servers as a function of L, the depth of the federation tree, all the different sizes of federations considered always followed the function ($s = 2^L - 1$).

B. FedORAM PERFORMANCE

1) RESPONSE TIME

Figure 5 plots the results for average response time in seconds of at least 100 messages over an increasing number of federation servers. There are 5 users per server and the number of servers ranges from 3 to 31. Although, we only consider relatively small sized federations, it remains the first oblivious federation in the literature. The scalability of ORAM schemes is a hot topic in the literature; one of the goals of the literature is to make an ORAM construction less impractical. Not to provide similar speed as a non-ORAM system. Therefore, with the response time from Figure 5, we show that oblivious federated architectures are good candidates to build an ORAM scheme. However, FedORAM does not aim to be as scalable as a classic IM system. Indeed,

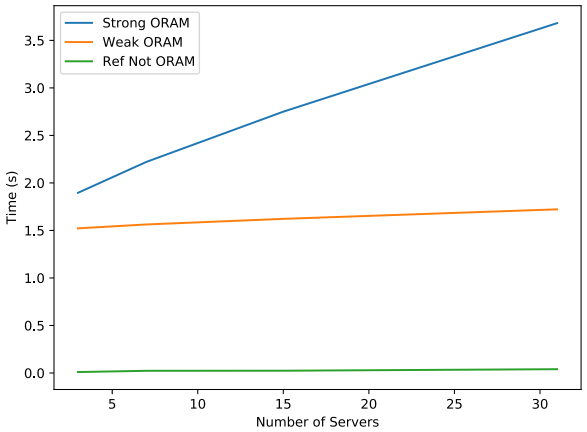

FIGURE 5. Average response time.

due to the overhead required to ensure obliviousness, that is not currently possible.

The traditional IM application without ORAM, depicted by the green line, has an almost constant response time of approximately 0.023 second. Our Weak FedORAM scheme, depicted by the orange line, has a response time of 1.5 seconds with a 3 server federation, which marginally increases as the number of servers increase. The response time for Weak FedORAM is higher than the non-ORAM system, which is to be expected, but it remains at a practical level while affording greater privacy preservation for the users.

The response time for our Strong FedORAM system is depicted by the blue line; it is evident that this approach has a much greater overhead than the other two models, which increases significantly as the number of servers in the federation increases. The response time for a 3-server federation is 1.8s and increases linearly.

To summarise, the response time for both Weak and Strong ORAM is dependant on the number of servers in the federation. However, due to the construction of Strong FedORAM, we see a much larger impact when additional servers are added. The response times of between 1.5 and 3.5 seconds, while higher than traditional non-ORAM applications, can be considered reasonable in the context of highly sensitive communications and the additional privacy preservation afforded by the schemes.

2) COMMUNICATION COST

Figure 6 plots the average network usage in MB for one message sent from one sender to one receiver over an increasing federation size. An average was taken from 100 messages.

The green line depicts the number of MBs that are sent per message using the traditional non-ORAM IM scheme. Each message is very small and consumes about 6KB of network capacity, which remains constant when the federation size increases.

Weak and Strong FedORAM have a similar average network consumption per message. They are depicted by the orange and blue lines respectively. Both schemes incur a large

[4]https://github.com/aio-libs/aiohttp
[5]https://cryptography.io

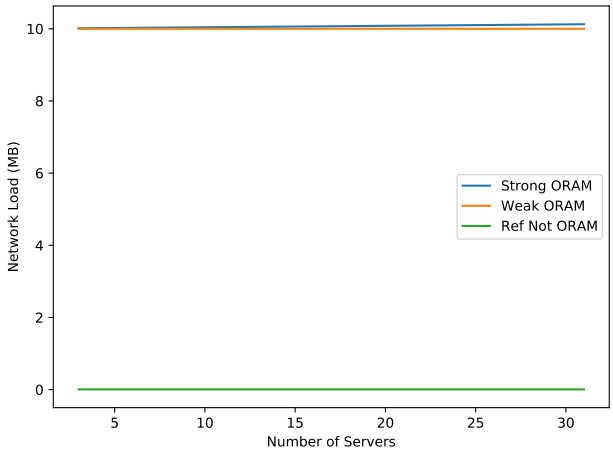

FIGURE 6. Communication cost.

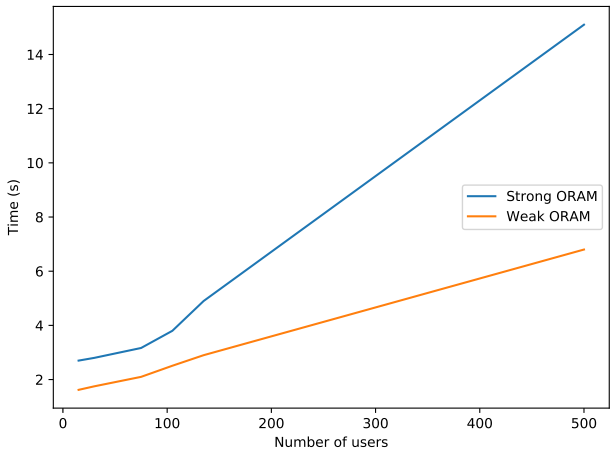

FIGURE 7. Federation at scale: response time.

network overhead per message due to the classic ORAM read operation on the destination server. Moreover, Strong FedORAM has a slightly higher usage due to its tree-based structure, where the number of messages sent depends on the size of the ORAM tree.

3) SCALING UP

Figure 7 plots the average response time per message in a 15-server federation as the number of users increase from 2 to 500.

The average response time for Weak FedORAM is depicted by the orange line, which shows that the response time per message increases as the number of concurrent users in the federation increases. The response time is approximately 1.6 seconds for 20 users and increases linearly.

The average response time per message for Strong FedORAM is depicted by the blue line, which also shows an increase in response time as more users are added to the federation. The average response time is 2.75 for 20 users and is logarithmic. This is mostly due to the bottleneck that is created by the root server and the tree structure.

These results are symptomatic of the inherent scalability issues that exist in ORAM schemes, which remain a limitation and a barrier to their widespread use.

TABLE 3. FedORAM vs Square Root ORAM.

	Strong ORAM	Weak FedORAM	SqrORAM
Response Time (s)	2.22	1.56	3.10
Bandwidth Cost (MB)	10	10	50

4) COMPARISON WITH EXISTING ORAM

Table 3 presents the average response time when a message is sent between 2 users in the different ORAM IM applications: our two schemes Strong and Weak FedORAM and a reference scheme from the literature. Note, Strong and Weak FedORAM are the first federated schemes to be proposed in the literature and therefore it is difficult to perform a completely fair comparison. The results presented are the average response times calculated based on a 7-server federation for FedORAM and the 2-server construction of SqrORAM, where 100 messages were sent from one user to another user. Both Weak and Strong FedORAM out perform SqrORAM, with faster response times. This is due to the costly secure multi-party computation used in the SqrORAM scheme.

5) SCHEME COMPLEXITY

Table 4 summarises our contributions and compares our scheme with some of the state-of-the-art ORAM constructions. Where n is the number of message stored on a given server while N is the number of message stored on the full oblivious scheme. Because FedORAM is based on PathORAM for its client to server connection, they share the same complexity in terms of client to server bandwidth and client storage. The communications inside of the federation are linear as well as the server computation.

TABLE 4. ORAM scheme complexity comparison.

Scheme	Bandwidth Cost C to S	Bandwidth Cost S to S	Client Storage	Server Computation
Path ORAM	$O(M \log N)$		$O(M)$	N/A
Group ORAM	$O((M+c)\log N)$		$O(M \log N)$	N/A
Onion ORAM	$O(M)$		$O(M)$	$O(M \log N)$
Weak FedORAM	$O(M \log n)$	$O(M)$	$O(M)$	$O(M)$
Strong FedORAM	$O(M \log n)$	$O(LM)$	$O(M)$	$O(M)$

VI. CONCLUSION

The objective of this research was to design a practical privacy preserving IM application, and in doing so to investigate the trade-off between security and performance in this applied ORAM context. That is, to assess whether it was possible to create a faster model than a classic ORAM system at the expense of revealing more (but not an exploitable amount) metadata than a classic ORAM system. The application selected for this work was an IM system, where we considered only one-to-one user communication.

We designed and implemented two new federated ORAM models: a Weak FedORAM that was less secure but faster and Strong FedORAM that was more secure but slower. Rigorous experiments were conducted to compare the performance and security of both new schemes against each other and against several reference models from the literature. Our results show that both FedORAM schemes can support reasonable response times, and can be considered more practical for use in real-world IM applications than classic ORAM schemes. Weak FedORAM leaks slightly more metadata than Strong FedORAM, but it has a faster average response time of approx. 1.5 seconds. Strong FedORAM has an average response time of 1.8 seconds in a small federation, but it is more sensitive to the federation size and suffers from delay as the number of servers in the federation increases.

One limitation of our proposal is that we only consider one-to-one conversations in our experiments. While it would be interesting to look at group chats, it is worth noting that two party chats are the most common use case within an IM context. While our federated schemes offer significant benefits in terms of performance, ORAM suffers from scalability issues and our system's response time grows as the number of users on the system increases, Strong FedORAM is affected more severely due to its construction but it provides full obliviousness on the federation.

Improving the scalibility of our federated schemes is an item for future work. Finally, we would also like to consider a scenario where there is an active attack between colluding servers in the federation.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions. They would also like to thank T. Laurent for the valuable discussions and reviews.

REFERENCES

[1] I. Abraham, C. W. Fletcher, K. Nayak, B. Pinkas, and L. Ren, "Asymptotically tight bounds for composing ORAM with PIR," in *Proc. IACR Int. Workshop Public Key Cryptogr.* Springer, 2017, pp. 91–120.

[2] B. Anandan, C. Clifton, W. Jiang, M. Murugesan, P. Pastrana-Camacho, and L. Si, "T-plausibility: Generalizing words to desensitize text," *Trans. Data Privacy*, vol. 5, no. 3, pp. 505–534, 2012.

[3] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam, "Verifiable oblivious storage," in *Public-Key Cryptography–(PKC)*. Springer, 2014, pp. 131–148.

[4] V. Ayala-Rivera, P. McDonagh, T. Cerqueus, L. Murphy, and C. Thorpe, "Enhancing the Utility of Anonymized Data by Improving the Quality of Generalization Hierarchies," *Transactions on Data Privacy*, vol. 10, no. 1, pp. 27–59, 2017.

[5] D. Boneh, D. Mazieres, and R. A. Popa, "Remote oblivious storage: Making oblivious RAM practical," MIT, Cambridge, MA, USA, Tech. Rep. MIT-CSAIL-TR-2011-018, 2011.

[6] T.-H. H. Chan, J. Katz, K. Nayak, A. Polychroniadou, and E. Shi, "More is less: Perfectly secure oblivious algorithms in the multi-server setting," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Springer, 2018, pp. 158–188.

[7] W. Chen and R. A. Popa, "Metal: A metadata-hiding file-sharing system," Berkeley, Tech. Rep., 2020.

[8] C. Cho, N. Döttling, S. Garg, D. Gupta, P. Miao, and A. Polychroniadou, "Laconic oblivious transfer and its applications," in *Proc. Annu. Int. Cryptol. Conf.* Springer, 2017, pp. 33–65.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proc. IEEE 36th Annu. Found. Comput. Sci.*, Oct. 1995, pp. 41–50.

[10] S. Devadas, M. van Dijk, C. W. Fletcher, and L. Ren, "Onion ORAM: A constant bandwidth and constant client storage ORAM (without FHE or SWHE)," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 5, 2015.

[11] S. Devadas, M. V. Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion ORAM: A constant bandwidth blowup oblivious RAM," in *Proc. Theory Cryptogr. Conf.* Springer, 2016, pp. 145–174.

[12] J. Doerner and A. Shelat, "Scaling ORAM for secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 523–535.

[13] R. Dubin, A. Dvir, O. Hadar, and O. Pele, "I know what you saw last minute-the chrome browser case," *Black Hat Eur.*, 2016.

[14] C. W. Fletcher, M. Naveed, L. Ren, E. Shi, and E. Stefanov, "Bucket ORAM: Single online roundtrip, constant bandwidth oblivious ram," IACR Cryptol. ePrint Arch., Tech. Rep., 2015:1065, 2015. [Online]. Available: https://www.eprint.iacr.org

[15] Z. A. Genç, V. Iovino, and A. Rial, "'The simplest protocol for oblivious transfer' revisited," *Information Process. Lett.*, vol. 21, Sep. 2020, Art. no. 105975.

[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC*, vol. 9, 2009, pp. 169–178.

[17] I. Goldberg, "Improving the robustness of private information retrieval," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 131–148.

[18] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, May 1996.

[19] S. D. Gordon, J. Katz, and X. Wang, "Simple and efficient two-server ORAM," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Springer, 2018, pp. 141–157.

[20] N. P. Hoang, Y. Asano, and M. Yoshikawa, "Your neighbors are my spies: Location and other privacy concerns in GLBT-focused location-based dating applications," in *Proc. 19th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2016, vol. 5, no. 3, pp. 851–860.

[21] T. Hoang, A. A. Yavuz, and J. Guajardo, "A multi-server ORAM framework with constant client bandwidth blowup," *ACM Trans. Privacy Secur.*, vol. 23, no. 1, pp. 1–35, Feb. 2020.

[22] Y. Jia, T. Moataz, S. Tople, and P. Saxena, "Oblivp2p: An oblivious peer-to-peer content sharing system," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 945–962.

[23] S. Landau, "Making sense from snowden: What's significant in the NSA surveillance revelations," *IEEE Secur. Privacy*, vol. 11, no. 4, pp. 54–63, Jul. 2013.

[24] K. G. Larsen and J. B. Nielsen, "Yes, there is an oblivious ram lower bound!" in *Proc. Annu. Int. Cryptol. Conf.* Springer, 2018, pp. 523–542.

[25] M. Maffei, G. Malavolta, M. Reinert, and D. Schroder, "Privacy and access control for outsourced personal records," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 341–358.

[26] M. Maffei, G. Malavolta, M. Reinert, and D. Schröder, "Maliciously secure multi-client ORAM," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Springer, 2017, pp. 645–664.

[27] T. Mayberry, E.-O. Blass, and A. H. Chan, "Efficient private file retrieval by combining ORAM and PIR," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–18.

[28] J. Mayer, P. Mutchler, and J. C. Mitchell, "Evaluating the privacy properties of telephone metadata," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 20, pp. 5536–5541, May 2016.

[29] B. Mi, D. Huang, S. Wan, L. Mi, and J. Cao, "Oblivious transfer based on NTRUEncrypt," *IEEE Access*, vol. 6, pp. 35283–35291, 2018.

[30] D. Micciancio, "Oblivious data structures: Applications to cryptography," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 456–464.

[31] T. Moataz, E.-O. Blass, and T. Mayberry, "CHF-ORAM: A constant communication ORAM without homomorphic encryption," Cryptol. ePrint Arch., Tech. Rep. 2015/1116, 2015.

[32] T. Moataz, T. Mayberry, and E.-O. Blass, "Constant communication ORAM with small blocksize," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 862–873.

[33] M. Naveed, "The Fallacy of Composition of Oblivious RAM and Searchable Encryption," IACR Cryptol. ePrint Arch., Tech. Rep. 2015:668, 2015.

[34] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 644–655.

[35] A. Pujol, D. Magoni, L. Murphy, and C. Thorpe, "Spying on instant messaging servers: Potential privacy leaks through metadata," *Trans. Data Privacy*, vol. 12, pp. 175–206, Jun. 2019.
[36] A. Pujol and C. Thorpe, "Dog ORAM: A distributed and shared oblivious RAM model with server side computation," in *Proc. IEEE/ACM 8th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2015, pp. 624–629.
[37] A. Pujol, C. Thorpe, and L. Murphy, "Collecting metadata on an instant messaging server," in *Proc. Eur. Conf. Cyber Warfare Secur.*, 2017, pp. 741–744.
[38] T. Pulls and D. Slamanig, "On the feasibility of (Practical) commercial anonymous cloud storage," *Trans. Data Privacy*, vol. 8, no. 2, pp. 89–111, 2015.
[39] L. Ren, C. W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. V. Dijk, and S. Devadas, "Ring ORAM: Closing the gap between small and large client storage oblivious ram," IACR Cryptol. ePrint Arch., Tech. Rep. 2014:997, 2014.
[40] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with $O((\log N)^3)$ worst-case cost," in *Advances in Cryptology–(ASIACRYPT)*. Springer, 2011, pp. 197–214.
[41] E. Stefanov and E. Shi, "Multi-cloud oblivious storage," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 247–258.
[42] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious RAM," 2011, *arXiv:1106.3652*. [Online]. Available: http://arxiv.org/abs/1106.3652
[43] E. Stefanov, M. V. Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 299–310.
[44] V. Torra and G. Navarro-Arribas, "Data privacy: A survey of results," in *Advanced Research in Data Privacy*. Springer, 2015, pp. 27–37.
[45] S. Wagh, P. Cuff, and P. Mittal, "Differentially private oblivious RAM," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 4, pp. 64–84, Oct. 2018.
[46] S. Zahur, X. Wang, M. Raykova, A. Gascon, J. Doerner, D. Evans, and J. Katz, "Revisiting square-root ORAM: Efficient random access in multi-party computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 218–234.

ALEXANDRE PUJOL received the master's degree in computer science from Polytech Marseille, France. He is currently pursuing the Ph.D. degree in computer science with the Performance Engineering Laboratory, University College Dublin (UCD).

LIAM MURPHY (Member, IEEE) received the B.E. degree in electrical engineering from University College Dublin, in 1985, and the M.Sc. and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1988 and 1992, respectively. He is currently a Professor in computer science and informatics with University College Dublin, where he is also the Director of the Performance Engineering Laboratory.

CHRISTINA THORPE received the B.Sc. and Ph.D. degrees in computer science from University College Dublin, in 2005 and 2011, respectively. She is currently a Lecturer with Technological University Dublin.

View publication stats