TRANSACTIONS ON DATA PRIVACY 12 (2019) 175–206

Spying on Instant Messaging Servers: Potential Privacy Leaks through Metadata

Alexandre Pujol, Damien Magoni, Liam Murphy, Christina Thorpe

Performance Engineering Laboratory, School of Computer Science and Informatics,

University College Dublin, Belfield, Dublin 4, Ireland

LaBRI — University of Bordeaux, Talence, France

Technological University Dublin

E-mail: alexandre.pujol@ucdconnect.ie, damien.magoni@u-bordeaux.fr, liam.murphy@ucd.ie, christina.thorpe@itb.ie

Received 14 March 2018; received in revised form 19 February 2019 and 28 April 2019; accepted 30 April 2019

Abstract. Nowadays, digital communications are pervasive and as such, they carry a huge amount of both professional and private information all around the world. Given the knowledge that can be extracted from such information, its confidentiality is of utmost importance for both companies and individuals. Recent news related to massive breaches of privacy by both external actors such as government agencies, rogue teams; and internal actors such as communication services providers (i.e., Google, Apple, Facebook, Amazon, Microsoft) have exacerbated the need for more secure communication technologies. Although message content can be encrypted end-to-end by so-called off-the-record techniques', message metadata such as sender, recipient, time sent and size can still leak a lot of information about communicating parties. Oblivious RAM (ORAM) systems form a promising new branch of research for hiding metadata from the hosting servers, but they have not yet been deployed in production environments. Due to their complexity and performance penalty, they can currently be used only for very simple client-server applications such as instant messaging (IM). In this context, we show accessing metadata on a messaging server can leak information that could be concealed by ORAM systems. More specifically, we show the differences observed in metadata collection between a classic XMPP server and two ORAM-based servers. In order to assess those systems, we have designed a new attack based on live forensic techniques to retrieve metadata from the RAM of a running IM server. We have used two datasets of instant messages for carrying out this assessment. Our experimental results highlight the leak of metadata from a standard messaging server and can also be used for testing the security of an ORAM-based messaging server.

Keywords. Data Privacy, Metadata, Privacy, Oblivious RAM, ORAM, Spy, Attack, Instant Messaging

1 Introduction

We live in a global digital society that is becoming more and more ubiquitously connected. Therefore, we now rely heavily on digital communications services such as e-mail, instant messaging, teleconferences and forums. In recent years, many of these services have moved to centralised Cloud-based applications, thus enabling their high scalability [8]. Among

these, popularity of Instant Messaging (IM) services have grown significantly. Nowadays IM system such as XMPP, Slack, Mattermost or Gitter[1] are becoming predominant in the workplace. Unfortunately, the main Cloud-based IM applications are controlled by only a few companies: the GAFAM[2], which can raise concerns about the control they can exert on users. Indeed, Edward Snowden's revelations [20] on the NSA's industrial and domestic spying through these companies have broken the trust between users and these service providers. Meanwhile, non-technical users have become accustomed to sending more and more personal information to the various services they use. This has led to an increase in privacy concerns from client companies and populations [37]. These breaches of privacy have proven to be a major threat since the service providers could accidentally or deliberately disclose the data, or use it for unauthorised purposes. Moreover, fear of massive spying attacks from external entities such as governments, companies, or specialised attack teams tends to be a challenge to democracy. Such fear could increase self-censorship and therefore prevent people from developing opinions different from mainstream. To mitigate these issues, many companies have implemented data privacy in their IM services by using end-to-end encryption. It has become a new de facto standard for IM applications such as Signal and WhatsApp, which are using Off-the-Record (OTR) cryptography. This is a first step to securing IM communications, but although data can be kept safe with the help of encryption when stored or in transit, an attacker or an untrustworthy service provider, with full access to an IM server, could still have many more possibilities for obtaining sensitive information. The question for these malicious parties then becomes: knowing that the user's data is encrypted, how can they still retrieve information? One potential solution would be to spy on the metadata while staying as passive on the server as possible.

From a user's point of view, the detailed privacy issues remain valid whether the attacker is inside (service provider) or outside the system (state or otherwise). Considering a given IM application, this paper tries to answer the following questions:

- How much metadata could an attacker collect over time?
- Could an attacker retrieve valuable knowledge from this collection?
- If an attacker could retrieve interesting knowledge from metadata, what are some of the possible solutions to prevent such attacks?

Given the stated research questions, this work looks at methods to obfuscate metadata. Oblivious Random Access Memory (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, see Section 5. 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 [6]. Thus, in order to speed up and simplify the ORAM schemes, the structure of most server models have been changed [44, 46]. Therefore, as of today, ORAM studies focus on scheme complexity and security features like group support [29]. Meanwhile, the security proof of these schemes is limited to the formal proof of obliviousness [46]; no empirical study of metadata leakage has been conducted on either classic or on ORAM systems.

[1] `https://slack.com/`, `https://about.mattermost.com`, `https://gitter.im/`.
[2] Google, Apple, Facebook, Amazon, Microsoft

In order to answer the above research questions, this study has the following objectives: We want to show the leak of metadata on a classic IM server and to compare it against an ORAM IM server. The purpose is to show that an ORAM construction can be seen as a solution to metadata privacy leak. Finally, we want to provide an attack, a dataset and a testbed in order to test the security of future ORAM models empirically. We limit our study to the IM use case in order to have a reasonable scope and because IM applications are perfect candidates for metadata collection. Our research contributions are as follows:

1. We defined a new metadata definition to be valid in the scope of data privacy. This definition is required when we work on data privacy. Indeed, the manner in which data leaks depends on its technical nature.
2. We created a new attack based on live forensic techniques in order to retrieve metadata from the RAM (Random Access Memory) of a running IM server. This attack is not based on a security vulnerability, and its efficacy increases over time, the more time spent passively collecting metadata, the more knowledge can be inferred from it.
3. In order to test this attack over different systems, we designed an experiment and conducted it using two instant message datasets.
4. We created and implemented a simple ORAM model for comparison purposes against a faster model from the literature.
5. We highlight the leak of metadata from non-ORAM IM servers and show that our experiment can be used in order to test the security of an ORAM scheme.

This work compares three different IM systems. All these systems share the same architecture, use case and attacker model. It is necessary for us to select only one application to be able to use the same dataset over our three IM systems and thus be able to compare them to each other. This comparison will look only at metadata leaks; we are not taking the performance of the systems into consideration.

The remainder of the paper is organised as follows: Section 3 proposes the new definition of metadata in the security and privacy general scope and for the use case of an IM server. Then we present, define and explain the attacker model considered in this work. Section 6 explains and justifies how our proposed attack works. Section 5 describes the ORAM systems that have been tested. Section 7 explains the construction of the datasets used, describes and discusses the experimental results. Finally, Section 8 concludes the paper and presents some future work.

2 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 that shows the importance of metadata leaks on instant messaging servers, and that proposes an Oblivious RAM solution in order to prevent this leak. Our previous work in [39] provides a distinction between metadata types, presenting the difference between tangible and intangible metadata. This article extends the previous definition to a more global and formal one.

Landau [26] highlights the importance of metadata in Edward Snowden's revelations on the security world. For instance, this paper shows that NSA is now known to collect domestic and telephony metadata. Furthermore, Mayer [32] evaluates the privacy properties of telephone metadata and shows how applying prediction on collected metadata, we can

retrieve sensitive knowledge such as location, relationship, health state. On another application, but still considering metadata, Dubin [14] shows how video identification can be achieved from an encrypted multimedia stream like YouTube. Finally, Hoang [21] shows 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 [1] or data anonymisation [3]. More generally, a lot of various techniques to ensure data privacy exists [48]. However, since these techniques do not protect against metadata leaks, 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) [7, 18] 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 different use case. Nevertheless, a lot of ORAM scheme include techniques that are similar to PIR.

Preventing the leak of data using an oblivious structure between a trusted processor and an untrusted RAM has been presented for the first time in 1996 by Goldreich and Ostrovsky [19]. They were the first to introduce the notion of Oblivious RAM. The purpose of this technique was to protect the access pattern of the software on the local memory. Later, the system was extended for cloud storage purpose [6, 44]. 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 changed. The structure of most models changed from tree based models, where the server structure is a binary tree [33, 41, 44, 45], to Path ORAM based system [46, 49]. 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 [2, 13, 30, 52]. These schemes use a fully homomorphic encryption function [17] 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. Onion ORAM [13] is a layered encryption approach that aims to reduce the communication overhead incurred in the oblivious storage process using bandwidth-efficient additive homomorphic encryption schemes. Meanwhile, some recent systems support sharing features, e.g., Group ORAM [29] or Obliv P2P [23]. They introduce some additional concerns; new parameters such as access rights management must be considered. Finally, Dog ORAM [38], proposes an ORAM scheme that merges some existing solutions, a Path ORAM based server, with server-side computation and user management.

All of these schemes propose innovations in terms of complexity or security features. They provide proof of security in the form of formal proof of obliviousness of the protocol proposed. No empirical security test is carried out. Moreover, they do not compare their security with other systems or with a non-ORAM server.

Since ORAM solutions present complexity issues, there have been numbers of studies conducted to examine the necessity of this scheme. On searchable encryption, Naveed [34, 35] presents 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 [40] shows ORAM is needed for other type of application.

This 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.

Computer forensics is the branch of computer science that collects, preserves and analyses material found on digital devices. It often occurs in relation to an ongoing investigation regarding computer crime. Any digital devices can be the source of investigation: desktop computer, mobile, server, etc. In order to facilitate more efficient and effective memory forensics investigations, forensics framework like Volatility [50] or *Rekall* [10] have been developed. Rekall [10] is a memory live forensic analysis framework. It is used in our work to easily read data from server RAM. As one of the main memory analysis tools, it is widely used around the world for forensic and anti-forensic analysis. For instance, an attacker can extract the TrueCrypt master key from the RAM of a running machine[3]. It can also be used as an anti-forensic tool to detect and report anomalies on a server using Google Rapid Response (GRR) [9]. GRR is a distributed live forensics and incident response platform. GRR uses rekall for most of the memory analysis tasks and is widely used on Google servers. Although we do not use GRR in this paper, it is an essential platform for our work because it guarantees the scalability of our attack over thousands of servers.

However, until recently, the collection of data from the user space process was complex using rekall. [11] solves this issue on Windows based systems, adding to rekall the ability to enumerate heap allocations and discover internal references. Meanwhile, in 2017, [4] and [5] add this feature to rekall on Linux. For instance, they successfully retrieved passwords, logins and emails from an unlocked KeePass[4] database.

In order to test our attack on both classic and ORAM IM servers, we need a dataset to test our attack on. As described in Section 7, we use two datasets. The first dataset used is the Enron Email dataset [12, 43]. To the best of our knowledge, it is the only available large-scale dataset where sensitive information can be identified and separated from regular information. As there is no available similar large-scale dataset for instant messaging. Our second dataset is a statistically generated synthetic dataset, using distributions from the literature [22, 27]. The work in [27] presents a study from Microsoft Research on a large instant messaging network. It explores a dataset from 30 billion conversations by 240 million users over one month, but unfortunately, it only provides aggregated information about it.

3 Metadata

In this section, we propose a new definition of metadata in the scope of data privacy. Then we define the concept of obliviousness and ORAM used in this paper.

In 2009, Georges [15] presented the three digital layers that constitute the full digital identity of a person on internet. As shown with a non-exhaustive list of elements in Figure 1, these layers are:

[3] https://volatility-labs.blogspot.com/2014/01/truecrypt-master-key-extraction-and.html
[4] https://keepass.info

1. **Declarative identity:** the data entered by the users (emails, contacts, files).

2. **Acting identity:** composed by the user's activities and events (such as sending emails, reading or writing files and ads seen or blocked).

3. **Computed identity:** calculated by the server using the data from the declarative & acting identities.

The author [15] performed a study on Facebook profiles and showed that digital identity on Facebook is determined less by declarative identity than by acting and calculated identity. Although this paper is ten years old, the concepts remain valid today, as evidenced by the recent blog on this topic[5].

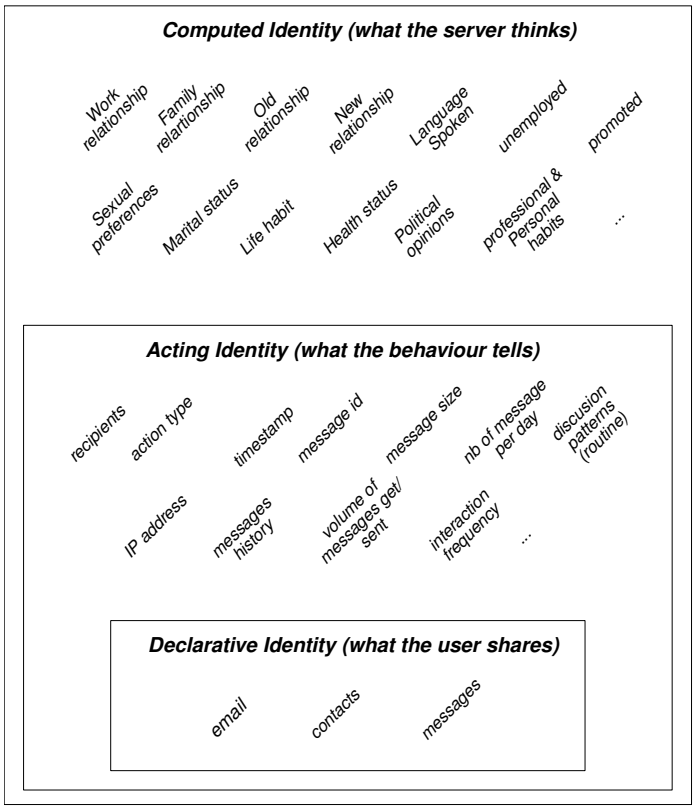


Figure 1: Representation of digital identities.

The definitions presented in this section are inspired by existing work in the literature on digital identity. Section 3.1 is dedicated to the definition of metadata concepts regardless of application use case, while Section 3.2 is dedicated to the definitions of metadata in the scope of an IM application. This differentiation is significant because the exact list and types of metadata vary with the application.

[5] https://qz.com/1525661/your-digital-identity-has-three-layers-and-you-can-only-protect-one-of-them

3.1 General Definitions

In this article, we propose a new formal definition of metadata and show that there are two classes of metadata, which differ from a technical perspective. They need to be handled in different ways, and they require different protection systems. Additionally, we propose a definition of 'metadata type'. These definitions are based on the work we published previously [39]; however, they provide a significant enhancement. Moreover, these definitions are different from the classification that can be found in the literature [47] because our work falls within the scope of data privacy, and therefore, we classify metadata according to its technical nature, not its use.

Definition 1 (**Metadata**). *Metadata is defined as the data that provides information about other data.*

Definition 2 defines the concept of type of metadata. A data has a given format (e.g., file, message), its metadata can be multiple, but they describe the data in their very own way. Each different way to describe a data is called a 'type' of metadata.

Definition 2 (**Type of Metadata**). *When the information collected between different data describes the same property, this information is from the same type of metadata.*

While definition 1 is similar to the common dictionary definition of metadata, it is not sufficient when we want to prevent metadata leaks in order to ensure user privacy. For this purpose, inspired by the declarative and acting identity detailed in the introduction of this section, two classes of metadata must be defined: Tangible Metadata (definition 3) and Intangible Metadata (definition 4).

Definition 3 (**Tangible MetaData (TMD)**). *Tangible metadata is the static metadata of user's data.*

TMD describes the data attributes; they are stored next to the data itself, and they can be found either encrypted or unencrypted on the server drive. For instance, they could be the file name, the file path, the timestamp, the owner id, image or video attributes. Therefore, TMD is part of the user's declarative identity because it is directly associated with the data the user wants to put online.
To limit metadata leakage, the TMD could be protected using a standard encryption mechanism or reduced using a metadata anonymisation toolkit[6]. However, these protection mechanisms are limited because TMD is usually needed in plain text when the user is performing an action on the server, in order to execute it.

Definition 4 (**Intangible MetaData (IMD)**). *Intangible metadata is the metadata created by an action of a client on a server.*

An example of when IMD is generated is when the server is uploading a file from a client to a server. The packets received by the server reveal that an action is being executed. This action by itself leaks intangible knowledge about the file owner. Therefore, IMD is part of the user acting identity.

It is important to note that there is a strong interdependence between TMD and IMD. Indeed, in order to execute an action (to generate IMD), the server usually needs to access TMD to check the validity of the action using, for instance, the owner id, the file name, etc.

[6] https://github.com/jubalh/MAT

In some cases, depending on the software running on the server, IMD can be converted by the server at run-time into TMD. The best example of this is the log files that shows the list of actions the server observed. Once converted and written on disk, they become full TMD and cannot be considered as IMD any longer.

As we explain with definition 4, each event/action generates intangible metadata. However, an action might not reveal anything by itself. Therefore, definition 5 is required and proposes the concept of a metadata pattern that considers time.

Definition 5 (**Metadata Patterns (MP)**). *The Metadata Pattern of a user for a given application is the list of all the metadata collected over time on this application.*

For ease of understanding, Figure 2 sums up the definitions presented here. TMD is written with the data while IMD is linked to an action. TMD and IMD together form the MP over time. For each of these metadata classes, the different type of metadata is always present.

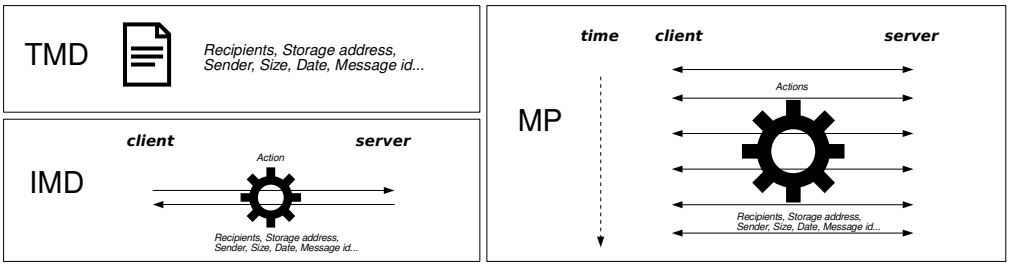


Figure 2: Conceptual overview of the metadata types.

3.2 Scoped Definitions

The scope of this research is defined for an instant messaging application as a use case. Therefore, knowing the exact type of data (messages) and metadata types, we can propose more precise definitions.

As shown in Table 1, we consider an instant messaging server S that stores a database $\mathcal{DB}$ on its memory disk. $\mathbb{U}$ is the list of all the users of the server S. For a given message m_{id} where id is the message identifier, we propose the following definitions.

Table 1: Definitions notations.

Notation	Meaning
S	Server
$\mathcal{DB}$	Server Database
id	Message identifier
m_{id}	Message from the database
$\mathbb{U}$	List of all the users known by S
U	Subset of $\mathbb{U}$
TMD/IMD	List of tangible/intangible metadata
N	Number of message in $\mathcal{DB}$
E	Number of events executed on S
c_U	A conversation from user in U

Definition 6 (**Tangible Metadata**). *Let TMD be the list of all tangible metadata stored in $\mathcal{DB}$. TMD contains an entry for all messages in $\mathcal{DB}$. For an IM server: $TMD = [tmd_1, \ldots, tmd_N]$ where tmd_{id} is the tangible metadata for m_{id}. Such that tmd_{id} contains at least one of the following metadata types:*

- U *the list of the users (such that $U \subseteq \mathbb{U}$) that have access to m_{id}),*
- s *the size of m_{id},*
- t *the access time,*
- id *the message id,*
- a *the message address in the database.*

Definition 7 (**Intangible Metadata**). *Let IMD be the list of all intangible metadata generated by an action on S. IMD contains an entry for all action that held in S. For an IM server: $IM = [imd_1, \ldots, imd_E]$ where imd_{id} is the intangible metadata for an action that send m_{id}. Such that imd_{id} contains at least one of the following metadata types:*

- U *the list of the users (presents in $\mathbb{U}$) that have access to m_{id}),*
- t *the current date of the action,*
- id *the message id,*
- $type$ *the type of action (e.g., pull, push).*

Definition 8 (**Metadata**). *The metadata of m_{id} is defined as M_{id}, the union of its tangible and intangible metadata, as in the following equation:*

$$M_{id} = IMD_{id} \cup TMD_{id} \tag{1}$$

Definition 9 (**Metadata Patterns**). *Lets c_U be an instant messaging conversation between users of U, such that $U \subseteq \mathbb{U}$. c_U contains the list of message id that represents this conversation. The metadata patterns of c_U is defined as MP_{c_U}, the union of all the metadata for the message in c_U, as in the following equation:*

$$MP_{c_U} = \bigcup_{id \in c_U} M_{id} \tag{2}$$

4 Attacker Model

This section describes the attacker model for this work; it is critical for the validity of the proposed solution. Changes in the attacker model might have major use case, security, and performance consequences. Although most of the ORAM models in the literature share an important common attacker model, there are often minor changes that make each model unique. One of the objectives of this paper is to compare a real life IM server with IM servers implementing classic ORAM models presented in the literature. Therefore, the attacker model and architecture considered need to be compatible in order to have comparable results.

The target application for this work is an IM server; Figure 3 shows the simple client-server architecture considered for this research. Table 2 sums up the attacker model. The IM server is considered 'honest but curious' as shown in definition 10. The data owners (discussion members) are trusted and the other users are not trusted, but we assume they do not collude with the server. We remark this paper focuses only on server security assessment, and not on clients security.

Definition 10 (Honest but Curious Server (HbC)). *The server is regarded as a passive adversary following the scheme specifications correctly but seeking to gather additional information about the client's data and access patterns.*

The common security policies considered for the IM servers are followings:

- **Secrecy:** A client can only read entries for which they hold read permissions.
- **Integrity:** A client can only write entries for which they hold write permissions.
- **Authentication:** A client can only access their entries after the server has verified their identity.

This paper focuses on obliviousness and its consequences. Definition 11 introduces the security property of obliviousness and definition 12 presents the definition of commonly used and accepted ORAM system in the literature [13, 49]. This definition is an extension of obliviousness to a client-server system. Intuitively, with an ORAM system, the server cannot determine:

1. What data is being accessed,
2. How old the data is,
3. If the same data has been accessed multiple times,
4. The access pattern (sequential, random, etc.),
5. Whether the access is a read or a write action.

Definition 11 (Obliviousness). *A scheme is oblivious if the server cannot distinguish between two accesses which contain any kind of authorised operation of the users.*

Definition 12 (Oblivious RAM). *Let a be the address to query in $\mathcal{DB}$ such as a denotes an input access sequence. Let $o = ORAM(a)$ be the resulting data request sequence of an ORAM algorithm. The ORAM guarantees that for any two sequences a and a', $ORAM(a)$ and $ORAM(a')$ are computationally indistinguishable if $|a| = |a'|$, and also that for any sequence a the data returned to the client by ORAM is consistent with the access sequence a with high probability.*

There are three different IM servers (also called systems) considered in this paper: a classic IM server using the XMPP protocol and two IM ORAM servers called *Basic ORAM* and *Path ORAM*. Section 5 describes in detail how these ORAM schemes work. Table 3 shows the security policies each system provides. Since this paper mostly focuses on obliviousness, other security properties are secondary and are not always respected by the ORAM scheme. This is not an issue for the scope of this work.

Table 2: Overview of the attacker model.

	Server	Owners	Clients
All Systems	HbC	Trusted	No collusion

Table 3: Security policies of the systems considered.

	Secrecy	Integrity	Authentication	Obliviousness
XMPP	✓	✓	✓	×
Path ORAM	✓	✓	×	✓
Shell ORAM	✓	✓	×	✓

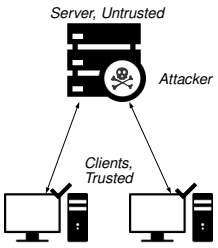


Figure 3: System architecture.

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

- The IM server S is assumed to be *HbC* and does not collude with any users.
- 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 software for a malicious client to voluntary send metadata to the server.
- The network is not trusted and can be malicious.

5 Oblivious RAM Scheme

In this section, we present two ORAM schemes: *Path ORAM* [46] and *Basic ORAM*. *Basic ORAM* is a very simple ORAM scheme we developed, it follows a construction that is often presented as obvious in the literature [38]. Path ORAM is a very well-known ORAM construction in the literature and it is used as a basis for most of the modern ORAM schemes [29, 46, 49]. This section presents these schemes and explains how we use them in the scope of our paper.

The attacker model used for these ORAM schemes is presented in Section 4. ORAM's particularity implies the obliviousness security policy is provided.

5.1 Basic ORAM

Basic ORAM is a new and straightforward ORAM scheme designed and implemented for the purpose of this research. It follows a simple protocol which is derived from an "obvious" algorithm from the literature [13]: download, re-encrypt, and upload the entire server database for each access to the database, in order to prevent the leak of any kind of metadata. This approach is obviously not efficient, however, the scope of this paper is to compare metadata leaks between different kinds of systems, not to improve the performance of an ORAM scheme. Therefore, the scheme is not intended to be practical and used in real-life IM application.

The client-server architecture presented in Section 4 is respected, but due to its simple oblivious structure. This IM ORAM scheme works differently than a classic IM server:

- Users always work on the full server database.
- The database is shared, which means, all the users can download, decrypt it and read it.

Overview Figure 4 presents how *Basic ORAM* works. The server is seen as an external storage system. From the server point of view, the database is encrypted using the clients' keys. All the clients that have access to the server and are able to decrypt the database with their user key. The encrypted database of size N bytes is called a *block*. This size is constant and must not change through time.

All the data the server wants to make available to the clients (all its storage capacity) is written in this block. For each action a client wants to execute on the database, algorithm 2 is used. It can be summarised in 5 simple steps:

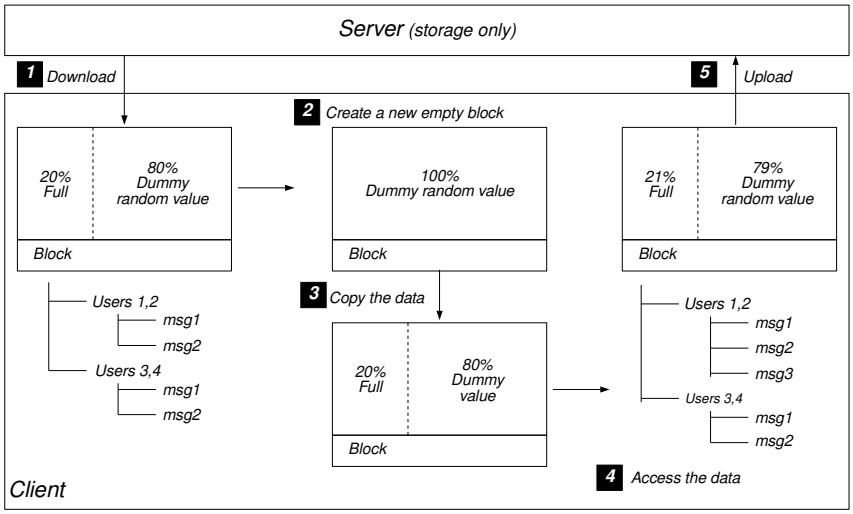


Figure 4: Overview of the Basic ORAM database structure.

1. **Download:** Download the server database and store it locally,
2. **Create:** Create a new empty block filled with random dummy values,
3. **Copy:** Copy all the data present in the database to the new block,
4. **Access:** Access the database (e.g., read, write, etc.),
5. **Upload:** Upload the new database version to the server.

Probabilistic encryption is used in order to encrypt the database, thus the re-encryption of the same data will create an entirely new binary. Moreover, it is vital to fill unused space with random values in order to make it impossible to differentiate between used and free space. The purpose of this algorithm is to show to the server that all data changed after every access, thus not revealing the real action. It is not possible to determine what data has been accessed and in which mode. As a consequence, it requires all the possible users on the server to be able to update the database.

By design, this scheme cannot handle a server-side IM database. Therefore, we mimic the IM action using a simple file system architecture as an IM database. This structure is presented in Figure 4. For each conversation, a folder named by the id of the participants, $(id1, id2)$ is created at the root of the database. All the messages are simple files in this folder and named in chronological order $(1., 2., 3.)$. Therefore, pushing a new message to the database means adding a new file into the conversation folder.

As a consequence, all the users have access to the full list of conversations. Therefore, this system cannot provide secrecy of the conversation. This was a design decision made during the conception of this scheme because of the sole objective of this ORAM scheme in the context of this work: test the obliviousness security property on a very well known ORAM scheme.

Formalisation Basic ORAM follows definitions 11 and 12 of obliviousness and ORAM respectively from Section 3. Definition 13 proposes a formal definition of *Basic ORAM*. Table 4 presents the notation used in the algorithms. Basic ORAM uses a classic probabilistic encryption scheme $\mathcal{E}$, the encryption key is noted k.

Algorithm 1 shows the block generation phase. At server initialisation, the user generates its database key k, then calls the algorithm 1 and uploads the encrypted database to the

Algorithm 1 $\mathcal{B} \leftarrow Gen(N)$

Input: Database size N
Output: $\mathcal{B}$ Block to be uploaded to the server
1: **for** $i \in \{0, \ldots, N-1\}$ **do**
2: $\quad \mathcal{B}[i] \leftarrow UniformRandom()$
3: **end for**
4: Encrypt $\mathcal{B}$ using k
5: **return** $\mathcal{B}$

Table 4: Basic ORAM parameters and notations.

Notation	Meaning
a	Address of the requested data
op	operation (read, write)
N	Database size in bits
$\mathcal{B}$	Local encrypted database
k	Block key, generated separately

server. Algorithm 2 shows the access function. When the user wants to access (read or write) a file on the database, it downloads the block, generates a new block and copies all the data to the new block and accesses the file. *Basic ORAM*'s security is regarded as the proof of obliviousness of the scheme. Theorem 14 shows the proof of obliviousness, therefore, we can say that *Basic ORAM* is a new valid oblivious scheme.

Definition 13 (Basic ORAM). *A Basic ORAM scheme is a tuple of algorithms* $(Gen, Access)$ *such that:*

$\mathcal{B} \leftarrow Gen(N)$ *This algorithm initialises an empty database full of dummy data. It takes as input the size of the database and outputs a newly generated encrypted database to be uploaded to the server.*

$data \leftarrow Access(op, a, data^*)$ *The access algorithm takes as input the action name to be executed on the database (read, write) and the optional new data to be written. It outputs the data requested. A new version of the database is uploaded to the server by this algorithm.*

Algorithm 2 $data \leftarrow Access(op, a, data^*)$

Input: a address, op operation, $data^*$ optional data.
Output: $data$ stored at address a
1: $\mathcal{B} \leftarrow$ Download block from S
2: $\mathcal{B}' \leftarrow Gen(N)$
3: **for** $i \in \{0, \ldots, \#\mathcal{B}-1\}$ **do**
4: $\quad \mathcal{B}'[i] \leftarrow \mathcal{B}[i]$
5: **end for**
6: $data \leftarrow \mathcal{B}'[a]$
7: **if** op **is** 'write' **then**
8: $\quad \mathcal{B}'[a] \leftarrow data^*$
9: **end if**
10: Upload $\mathcal{B}'$ to S
11: **return** $data$

Theorem 14. *Basic ORAM is a valid oblivious RAM scheme as defined in definition 12.*

Proof. Let a be the address of the $data$, a denotes an access sequence to query. Let $BORAM$ be a Basic ORAM scheme such that $BORAM(a) = data$. By construction, $BORAM$ uses a probabilistic encryption scheme $\mathcal{E}$ such that for any encryption of $data$ and $data'$: $\mathcal{E}(data) \neq \mathcal{E}(data')$ even if $data = data'$. Dummy random values fill empty space, free and used space are indistinguishable. Finally, for every access, $\mathcal{B}$ is re-encrypted using $\mathcal{E}$. Therefore, for all addresses (a, a'), $BORAM(a)$ and $BORAM(a')$ are consistent and computationally indistinguishable even if $|a| = |a'|$. Therefore BORAM is oblivious. □

Basic ORAM has been implemented in Bash. It heavily uses well-known cryptographic software like GPG, Tomb[7] and dm-crypt[8]. The source code of this application is open source[9]. We implemented it in shell for ease of simplicity and speed reason.

5.2 Path ORAM

Path ORAM [46] is one of the main ORAM schemes as it inspired most of the schemes in the literature and shares concept and data structure. Unlike Basic ORAM, Path ORAM intends to be as practical as possible. Indeed, instead of working with the full database for every access, this scheme works on a subset of data as shown in Figure 5. In order to ensure obliviousness, like Basic ORAM, these blocks are re-encrypted and re-written after every access. It makes Path ORAM an ideal scheme to conduct security tests and to compare against Basic ORAM. In this paper, the client-server architecture is respected, but all the users have access to the full database.

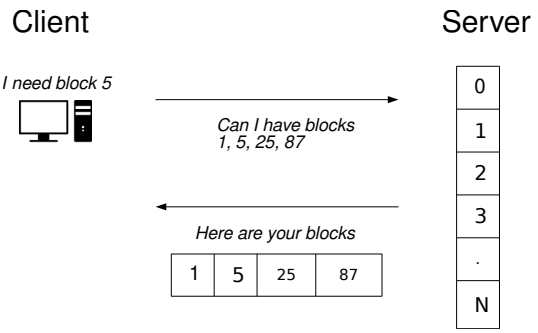


Figure 5: Path ORAM base system.

Table 5: Path ORAM notations.

Notation	Meaning
N	Number of data blocks
c	Number of clients
L	Depth of the binary tree
B	Block size in bits
Z	Capacity of each bucket (in blocks)
$\mathcal{S}$	Client local stash
$PositionMap$	clients local position map
$\mathcal{P}(l)$	Path from the root to the leaf l
$\mathcal{P}(l,i)$	The i-th bucket (from the root) on $\mathcal{P}(l)$

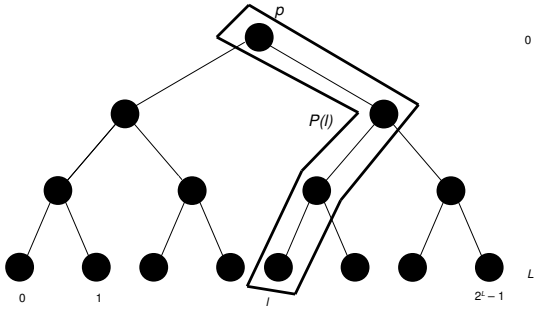


Figure 6: Illustration of the Path ORAM database structure (from Dog ORAM [38]).

Overview This scheme is called Path ORAM because data on the server is always accessed in the form of tree path.

Table 5 presents the notation used in the algorithms. The client stores a small amount of local data in a stash $\mathcal{S}$. The data on the server is stored in a binary tree of $L+1$ levels (see Figure 6). Each node is called a bucket. Each bucket contains Z blocks of data (real or dummy). The leaves are numbered from 0 to 2^L-1. Each block is associated with a random path from the root to a leaf l: $P(l)$.

A matrix of metadata called $PositionMap$ is needed to remember on which leaves the virtual blocks are assigned. It links the virtual address a of a block to its assigned leaf on

[7] `https://gnupg.org/`, `https://www.dyne.org/software/tomb/`
[8] The Linux standard tool for block storage encryption.
[9] `https://github.com/roddhjav/boram`

the server. $PositionMap[a]$ gives a leaf l. The block at the address a can be found in the server in the path $P(l)$.

Each time, each block is mapped to a random leaf in the tree. On every action, the entire path $P(l)$ is read into the stash $\mathcal{S}$, the target block is reassigned to a random leaf, and the path that was read is re-written to the server.

Formalisation Path ORAM follows the definitions 11 and 12 of obliviousness and ORAM respectively from Section 3. Because this is not a new model, we do not propose a security proof of the scheme.

Algorithm 3 (taken from Path ORAM [46]) shows the Access algorithm, similar to Basic ORAM, this is the only algorithm needed in order to conduct any kind of operation on the database. This algorithm can be summarised in 4 steps:

1. Remap the block to a new random leaf l.
2. Read the path $\mathcal{P}(l)$. It contains the block a.
3. Access (read or write) the block.
4. Write the path back to the server.

Algorithm 3 $data \leftarrow Access(op, a, data^*)$

Input: a block address, op operation (read, write), $data^*$ optional data.
Output: $data$ stored at address a

$l \leftarrow PositionMap[a]$
$PositionMap[a] \leftarrow UniformRandom(0, 2^L - 1)$
for $i \in \{0, 1, \ldots, L\}$ **do**
    $\mathcal{S} \leftarrow \mathcal{S} \cup ReadBucket(\mathcal{P}(l, i))$
end for
$data \leftarrow$ Read block a from $\mathcal{S}$
if op **is** 'write' **then**
    $S \leftarrow (S - \{(a, data)\} \cup \{(a, data^*)\}$
end if
for $i \in \{L, L-1, \ldots, 0\}$ **do**
    $\mathcal{S}'' \leftarrow (a', data') \in \mathcal{S} : \mathcal{P}(l, i) = \mathcal{P}(PositionMap[a'], i)$
    $\mathcal{S}'' \leftarrow$ Select min($|\mathcal{S}'|, Z$) blocks from $\mathcal{S}''$
    $\mathcal{S} \leftarrow \mathcal{S} - \mathcal{S}'$
    WriteBucket($\mathcal{P}(l, i)$)
end for
return $data$

Implementations of *Path ORAM* for research purposes already exist and are available for download and use. We used PyORAM[10], a python library implementing Path ORAM. Our implementation creates an oblivious IM server in python based on PyORAM. The IM structure is similar to Basic ORAM's structure.

[10] https://github.com/ghackebeil/PyORAM

6 Proposed Attack

This section propose a proof of concept attack that would allow an attacker to spy on the metadata leaked by an IM server. The attack proposed here has been tested against classic IM server and the ORAM schemes presented in Section 5.

In Section 4, we explained that the server is not trusted. Therefore, it is assumed for our attack that the attacker has root access to the server. This assumption is common in the literature [13]. Figure 7 shows an overview of the attack. The principle is that even if the metadata are encrypted alongside the data when written on the disk while they will always be available in plain text on the server's memory. Therefore, our attack analyses the state of the RAM of a running server in order to collect metadata. Our method is based on a live forensic technique. In order to exploit this attack, an attacker would have to follow these steps:

1. Gain root privilege on the server.
2. Spy on the server's RAM in order to collect live data from the IM server.
3. Select the metadata from the data present in RAM
4. Exfiltrate the metadata collected.

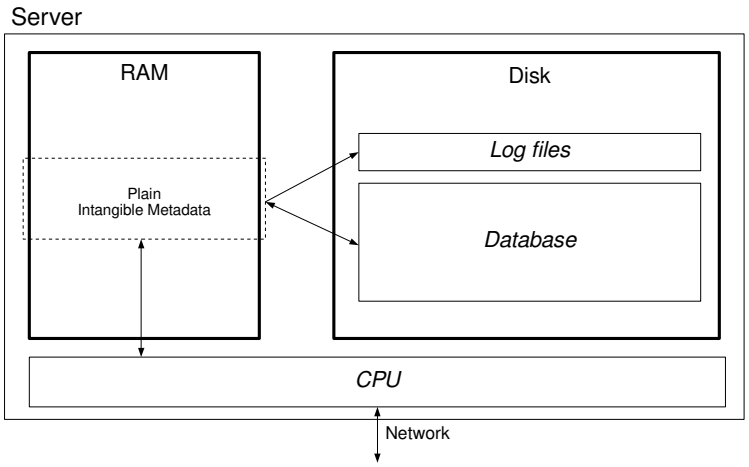


Figure 7: Architecture of the proposed attack.

6.1 Technical Justification

This section is dedicated to the justification of our proof of concept for the attack we proposed.

The collection of metadata could not have been done on the network because it is not trusted and therefore the packets traversing the network are encrypted. Moreover, not enough information can be collected from the network for the attack we propose in this work. Finally, many networks have dedicated devices (Network Intrusion Detection/Protection System) that are able to detect and prevent network spying.

Since metadata collection from the network is not possible in the scope of our paper, we need to collect data from the server itself. We considered the two following possibilities of spying on metadata from a server. Each time we explain the advantages and disadvantages of the technique envisaged.

Collection from the server hard drive It is a simple solution that only requires an attacker to be able to read files from the server disk. Depending on the system used, these files could be either log files or data files. Because IMD can be converted to TMD, it seems to be useful to spy on specific files on the server. However, this solution has several major issues:

- Not all software writes metadata to the disk. Therefore, this technique might not work on many systems.
- Applications could protect metadata using a simple encryption scheme.
- Outsourcing application's logs to a dedicated system is common[11].

Collecting from the process RAM It is a complex solution, it requires an inspection of the process memory state of the RAM to perform an analysis of it, and to select and retrieve the interesting metadata. Like the previous solution, collecting metadata from the process state requires that each system creates a specific program that will look for the target data. Moreover, in order to be useful, it has to be done in real time. Nevertheless, this technique has the following advantages:

- It collects all metadata before it is written on the server drive.
- It is impossible to prevent on an untrusted server because it is not based on a security issue as explained in the Attack Protection paragraph below.

Attack Protection Our attack is not based on a security vulnerability present in the operating system; it can be executed on any operating system. Indeed, the attacker has root access on the server; therefore they can do everything including spying at the RAM state. Moreover, in our security model we define that the server is not trusted, therefore, it could even be the server owner and IM provider that is spying on the users.

A solution that would prevent this attack on a trusted processor (and not-trusted RAM) would be to use a local ORAM system between the processor and the RAM. As explained in our related work, Section 2, modern cloud ORAM (presented in Section 5) are all based on this local ORAM [19]. However, such systems are intended to be used on the client computer; it is useless on a server because we do not trust the server, meaning we do not trust both the RAM and the processor.

Not limited to centralised architecture This attack focuses on simple client-server architecture. However, similar results could be obtained on a peer to peer architecture as a decentralised ORAM system has already been proposed [23].

[11] `https://logentries.com/`

6.2 Data Collection

A basic example of a memory forensic investigation on a server could be summarised as follows:

1. Create an image of the server memory.
2. On a local system, analyse the image generated and look for irregularities in the kernel and the process executed.
3. Write a report on your findings.

This method is useful in the case of an investigation of a known attack or security breach. However, it is limited because it is a manual investigation that takes place only after an attack has been noted. In order to fix this limitation, live forensic analysis tools and methodologies have been developed. Typically, they are automated in order to detect in real time a specific set of actions on the system being monitored.

Rekall Forensic Framework We use *Rekall* as a framework to easily create a passive attack on a server. We selected *Rekall* because it has the ability to work on the live RAM of the system, not only on an image of the memory. However, this is not a simple task, it requires:

- To compile and add a module to the Linux kernel in order to bypass kernel's security preventing even a root user from reading more than 1 MB at once from the RAM.
- A *Rekall* kernel profile that describes kernel allocation. It needs to be generated for each different Linux distribution and kernel version we target.

The *Rekall* framework was used because it provides high-level access to complex memory structures. It is allowing us to quickly set up an attack while not spending to many time on it. However, it is not mandatory to use *Rekall*. The same attack could have been done from scratch to reduce payload size, thus maximising the chance for the spy to stay as passive as possible on the target server.

Memory Heap The purpose of the attack presented here is to retrieve the metadata when they appear in a plain text form in server memory. Therefore, it is necessary to understand how volatile memory works and how the operating system handles memory allocation of a user process. Linux process memory is organised into a heap. The Linux heap is a global data structure that provides dynamically allocated memory storage. It allows an application to allocate space for variables at run time that can exist outside the scope of the currently executing function. A significant amount of research has been carried out on the behaviour of the heap with Linux processes. In 2007, Ferguson [16] showed how the heap on a Linux process works and presented a possible approach to exploit it. However, not enough information was released to use it in a real-life usage. In 2015, Cohen et al. [11] solved this issue on Windows based system, adding to *Rekall* the ability to enumerate heap allocations and discover internal references. Meanwhile, in 2017, [4] and [5] add this feature to *Rekall* on Linux. This later work is used without significant change in this paper to collect data from process memory. The description provided here is simplified by design for ease of understanding.

Data Collection Figure 8 shows the process memory layout. A chunk contains the actual process data which has been allocated either explicitly via a malloc call or implicitly

via a new class instantiation. Those chunks are located in memory regions called "Mmap region". Each chunk contains the actual data structure stored in memory alongside the chunk's internal metadata, such as size, state (free, used, etc.), pointers. The "Heap" region contains metadata describing its size and the different Mmap regions used by the process. Indeed, a process has as many Mmap regions as it has threads.

The purpose of our data collection system is to read message objects from the IM server process. Therefore, knowing the message object structure, we need to look for all the data chunks present in the server process that contains this structure. Then, *Rekall* allows us to repeat this action through time.

This functionality has been implemented in *Rekall* [5]. It takes the form of Rekall modules. The implementation of our attack also takes place as a python *Rekall* module by using the heap functionality on the live memory. The exact structure collected depends on the target process.

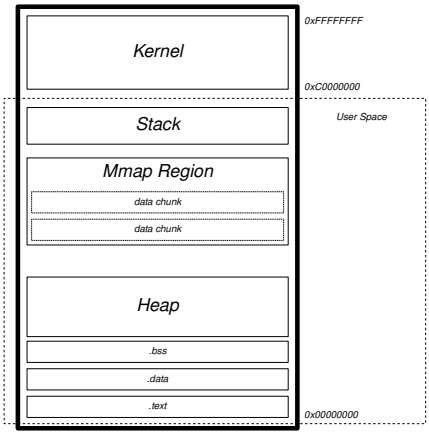


Figure 8: Linux memory process layout.

7 Experiments

In this section, we present the experiments we designed in the scope of this paper. We test the classic XMPP server and the ORAM schemes presented in Section 5, using the attack from Section 6, and using the datasets we present in Section 7.1. The purposes of these tests are to:

- Compare metadata leaks between classic and ORAM servers.
- Show XMPP servers leak metadata and metadata leaks can leak to privacy issues.
- Validate our proposal to use an ORAM scheme as a solution to prevent this leak.

Table 6 summarises the systems we tested on our metadata collection attack proposed in Section 6. We use two datasets in order to feed the IM server with comparable data. We use an XMPP server as a reference IM server while we employ Basic ORAM and Path ORAM as oblivious IM server.

Table 6: Overview of the experiments.

	Reference IM Server	**ORAM IM Servers**	
Dataset 1: Sensitive **Dataset 2**: Synthetic	XMPP IM Server	*Basic ORAM*	*Path ORAM*

7.1 Datasets

In order to test our experimental ORAM servers, we need a relevant IM dataset. Then, sending all messages from our dataset, we can collect the metadata available from the server and compare the metadata collected between the IM systems considered. Dataset selection is important; it should come from a real application with real users, the number of messages should be high enough in order to be representative, and the messages should show real interactions between users. However, to the best of our knowledge, no such IM dataset currently exists. Consequently, we do not use one but two datasets that allow us to respect our specifications for a good dataset:

1. **Sensitive Dataset:** Based on the Enron email dataset [12], selecting only sensitive conversations (as defined in Section 7.1.1).
2. **Synthetic Dataset:** Statistically generated from literature description of IM dataset.

7.1.1 Sensitive Dataset

The first dataset considered is based on the Enron email dataset [12]. It contains 0.5 million public emails and is well known in the literature [24, 43]. The Enron Corporation was an American gas company that went bankrupt in 2001 because they were involved in market manipulation, corporate fraud, and corruption. The email dataset was released a few years after and is the collection of 0.5M emails from approximately 150 users, mostly senior management of Enron.

The Enron dataset takes the form of hundreds of mailboxes from Enron employees. Therefore, not all the emails show a real discussion between users. Some emails are duplicated, spam, deleted chunks or unfinished drafts. Before being able to use the dataset, a preliminary scan was needed in order only to select real discussions. The usable Enron dataset used in this work has approximately 0.4M messages.

Objectives The most interesting part of this dataset is that they are real emails from a real company. Therefore, we can use this dataset for our attack by converting the email files present in the dataset in instant message object.

However, this dataset has a few disadvantages: The Enron bankrupted 17 years ago, IM usage in a company almost did not exist at this time whereas, nowadays IM system such as XMPP, Slack and Mattermost are becoming predominant in the workplace. Furthermore, users do not use emails the same way they use IM systems (e.g., frequency of access, time between messages, etc.).

Nevertheless, this limitation is not a big issue because we can use the Enron dataset as a real-life dataset from a company that had sensitive discussions to protect. The leak of these discussions would show a privacy leak. We can read special discussions regarding market manipulation, and we know the company structure. Therefore, we can identify important users. Besides, we know the list of people who were at the origin of the Enron scandal; we also know that the message content at the origin of the Enron scandal are sensitive in the scope of this scandal. Consequently, we can select the sensitive messages and compare the metadata of these messages with the metadata of the full dataset. Then we can run our experiment on these sensitive messages and check how much metadata is collected depending on the system under test. The purpose of this dataset is to show the following:

- Show there are differences in the metadata generated from regular discussions and sensitive discussions, we show these differences on the local dataset.
- Show the reference IM server has the same leak as the local dataset.
- Show ORAM systems do not leak metadata in the same way.

In order to achieve these objectives, we select all the discussions that are specifically related to the Enron scandal; it forms a sub-dataset of the Enron's dataset that we call "Sensitive dataset". Then, we show differences in terms of MP pattern between the Enron dataset and the sensitive dataset. It provides experimental proof, depending on the kind of discussion people have, their IM use change. This is a first demonstration of the existence of privacy leak through the means of metadata. Then, we show using the experiment described in Section 7, we continue to see MP leaks on the reference server. Finally, we compare metadata leaks on reference and ORAM servers.

Kolmogorov-Smirnov Test The collected data forms different distributions of metadata. For a given dataset, our experiments require to show whether the empirical distribution formed by the collection of metadata on two different systems are the same. To achieve this objective, we used the Kolmogorov-Smirnov (KS) test for goodness of fit [31]. It is a common test used for distribution comparison.

The KS statistic $D_{n,n'}$ quantifies a distance between two empirical distributions of size n and n'. X and Y are the Empirical Cumulative Distribution Function (ECDF) of these distributions. An ECDF shows the probability that the secret key will take a value less or equal than x (3).

$$D_{n,n'} = \sup_x |X_n(x) - Y_{n'}(x)| \tag{3}$$

The null hypothesis H_0 we want to prove is that the two samples are drawn from the same distribution. If the KS distance is smaller than a critical distance then we cannot invalidate our null hypothesis H_0.

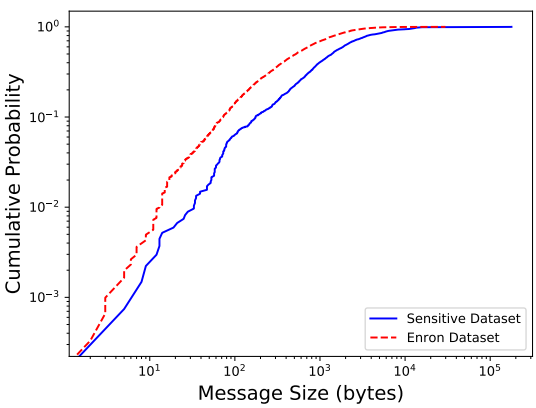
In order to test H_0, we proceed with a comparison of $D_{n,n'}$ against a critical distance δ_α. δ_α is function of a critical value $c(\alpha)$, where α is the significance level, and of the number of samples. $\alpha = 0.05$ is typically used for most applications [51] and $c(\alpha)$ is given by equation (4):

$$c(\alpha) = \sqrt{-\frac{1}{2}\ln\left(\frac{\alpha}{2}\right)} \tag{4}$$

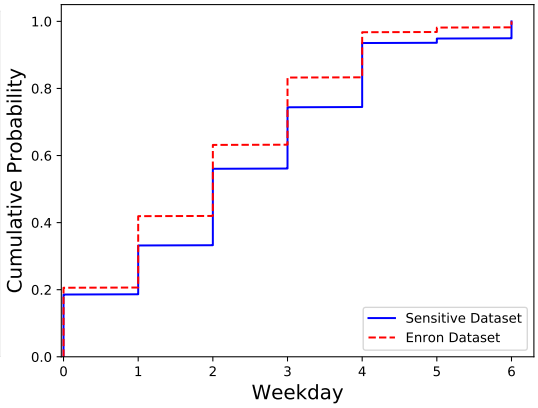
The null hypothesis H_0 is rejected for a given α if (5):

$$D_{n,n'} > \delta_\alpha = c(\alpha)\sqrt{\frac{n+n'}{nn'}} \tag{5}$$

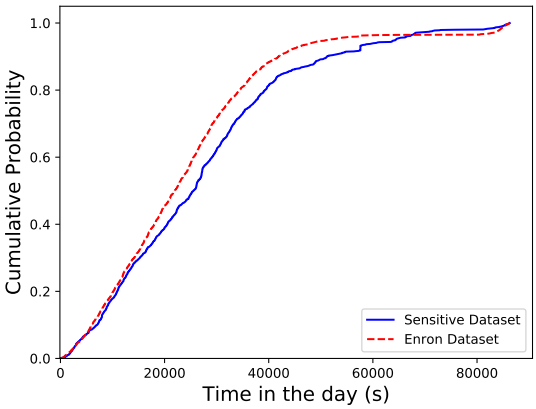
Sensitive vs Enron Dataset The sensitive dataset was sorted from Enron dataset selecting both the discussions from the main actors of the Enron scandal and the discussions containing sensitive keywords relative to the scandal. Figure 9 shows the differences in the metadata pattern (MP) between the sensitive dataset and Enron dataset. This figure shows



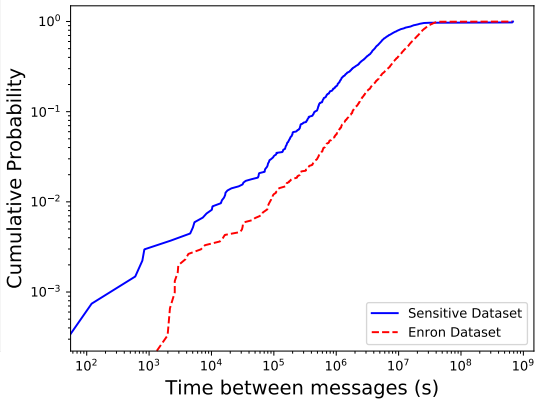
(a) Sensitive dataset size of messages.



(b) Sensitive dataset weekday.



(c) Sensitive dataset hours.



(d) Sensitive dataset time between messages.

Figure 9: Sensitive vs Enron Dataset, the reference test: Figure 9a messages size, Figure 9b weekday (0 Monday, 6 Sunday), Figure 9c time in the day and Figure 9d time between messages.

the cumulative probability in terms of MP regarding the classes of comparison considered in our work. The y-axis is the Empirical Cumulative Distribution Function (ECDF), that is to say, the probability of the class of metadata (message size, time in the day, time between messages) will take a value less than or equal to the value in the x-axis. The KS statistic is the biggest distance between the two ECDF curves. This figure is useful to see the differences between the sensitive dataset and the Enron dataset (the reference test). Table 7 shows the exact KS distance calculated. These results show for all the classes of metadata considered, there is a difference in terms of MP between the sensitive and Enron dataset and this difference is statistically significant.

Table 7: Enron Dataset vs Sensitive dataset.

	size	**weekday**	**hours**	**timesstamp**
Critical distance δ_α	4.202×10^{-2}			
KS Distance $D_{n,n'}$	3.021×10^{-1}	8.863×10^{-2}	1.030×10^{-1}	3.986×10^{-2}
Validity	Invalid	Invalid	Invalid	Invalid

7.1.2 Synthetic Dataset

The second dataset is a synthetic IM dataset. Indeed, the Enron dataset is an email dataset, not an IM dataset. Although it stays very interesting because of its structure from a real company, the messages are from email, not IM. We do not use an IM application in the same way we use an email system [36]. Therefore, we also need to run our experiment on an IM dataset. However, since no such dataset is freely available, we created one based on statistical information found in the literature. Microsoft present in a paper from Eric Horvitz [27] a study on a large IM network. It presents the statistical characteristics of a real dataset from 30 billion conversations by 240 million users over one month. We use this paper in order to generate our own synthetic IM dataset.

The synthetic dataset is mostly useful in order to validate the use of the Enron e-mail dataset. Indeed, in Section 7, we show metadata collection works the same way on both datasets. However, its use is limited because it does not hold information on important metadata types such as message size, pull, and push.

We generated a synthetic IM dataset using the distribution presented by Microsoft [27]. Although this paper provides a full study on the IM dataset, it does not provide the exact distribution of all the properties needed to generate a dataset. Therefore, our dataset is generated with four different values. It is enough for the scope of this paper because it is only used to retrieve metadata from IM dataset. Moreover, a dataset from a real-life scenario is already used in the experiments. The values considered are:

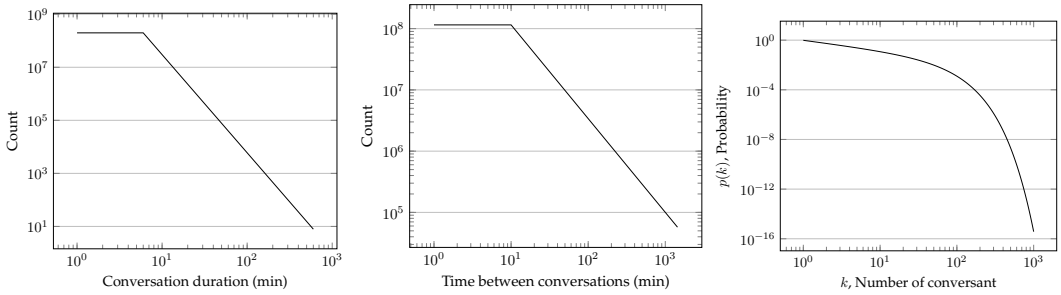
- Average conversation duration per user.
- Time between conversations for a given user.
- The degree of distribution of the communication network, it represents the communication network.
- The number of exchanged messages per minute of conversation.

The generated dataset contains only conversations between two participants because it counts for 99% of all discussions of the source dataset. Figure 10 provides two important distributions in order to generate a relevant IM dataset. These distributions are a simplified form of the distribution proposed [27]. The x-axis shows the value considered and the y-axis indicates the number of occurrences of the same value in the dataset. Figure 10a shows the distribution of average conversation duration in minutes and Figure 9b presents the distribution of time between conversation in minutes. Figure 10c shows the degree of distribution of communication among the users. It provides a probability that a user would have a given number of active friends. It is used in order to link users together in our generated dataset. Finally, the number of exchanged messages per minute of conversation is given in the table 1d [27] and is set to 1.42 message/minutes.

7.2 Test Setup

Reference server As a reference non-ORAM server, we use an XMPP [42] server. We selected the XMPP protocol because it is very well-known and used in a lot of various systems. Moreover, it supports message encryption with Off-the-Record (OTR) messaging. Nevertheless, any open source IM server could be used as a reference server. In the test we conducted, the XMPP server used is Prosody[12] version $0.10.0$.

[12] `https://prosody.im`



(a) Average conversation duration per user.

(b) Time between conversations of user.

(c) Degree distribution of communication network.

Figure 10: Synthetic dataset, the temporal characteristic of conversations. 10a Average conversation duration per user, 10b time between conversations of user and 10c degree distribution of communication network, that is to say number of people with whom a person communicates.

ORAM servers The used ORAM models in the experiments are the two ORAM models presented in Section 5 and adapted for an IM application. These models are by construction very different. Basic ORAM is extremely slow while Path ORAM has been designed in order to be practical. However, they should theoretically provide the same security. Therefore, they are two good candidates for these experiments. These two oblivious servers were selected for our experiments because we would like the answer the following questions:

- Do ORAM servers leak as much data as a classic server?
- Is data leakage a privacy issue?
- Do these models provide the same security (in terms of obliviousness)?
- Could we use the empirical experiment presented here to check the obliviousness of ORAM schemes?

Test bed All the tests presented here share the same test-bed:

- **Dataset:** The different datasets have the same structure. Each dataset is constituted by a list of discussions from different users. Each discussion contains the list of messages that have been sent by the users with the metadata linked to these messages.
- **Servers:** The server takes the form of a VM with four virtual CPU and 2 GB of RAM, it runs ArchLinux. The IM server tested is installed on the server alongside with *Rekall* and the attacker script. Depending on the IM server, the attacker script changes in order to fit the structure of metadata we wish to collect. Then, our *Rekall* script writes down the IMD collected.
- **Clients:** The host system used has the IM client; it mimics all the users in the dataset. It reads the dataset, orders it by date and sends the IM messages to the server by using the good IM interface depending on the IM server. In order to speed up experiments executions, we change the timescale. Otherwise, we would need up to 4 years to send the messages from the Enron dataset. Please note we do not modify the relative time between messages and discussion.

Evaluation Criteria The attack script retrieves IMD from multiple type of metadata. Table 8 presents the collected classes of metadata on the different IM servers considered in this work. On the XMPP reference test, the metadata collected are from different types. We

gathered for all the messages, the message id, the user ids of the sender and receiver, the exact timestamp and, we were able to compute the size of the messages.

Due to the structure of the ORAM server, it is not possible to retrieve most of these classes. The message *id*, *size* and *recipients* are not available. Therefore, our comparison method is based on the only metadata type that is common to all servers, the time. Distribution comparison between servers is done by using the KS-Test as described in Section 7.1. We do not want to compare the time of message reception on the server, but access patterns, the time-based distribution considered is the time between messages.

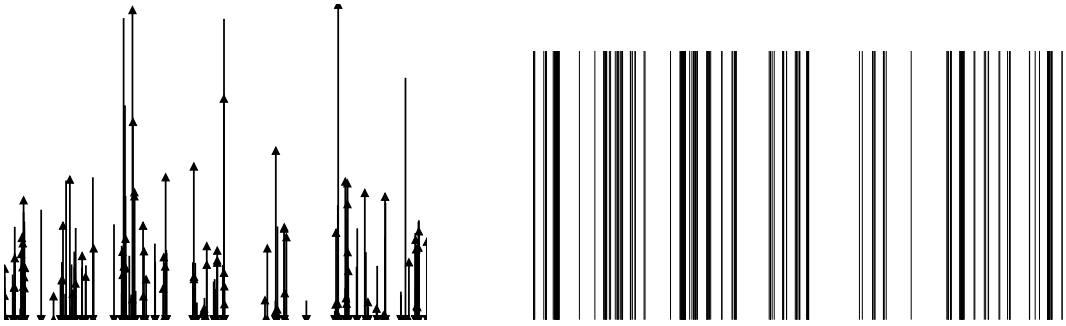
Table 8: Metadata to consider depending on the IM system.

XMPP	*id*	*size*	*from*	*to*	*time*
Basic ORAM					*time*
Path ORAM					*time*

7.3 Results

This section shows the results of our experiments. In all of these experiments, the reference test is the classic XMPP server while the tests are the ORAM servers using sensitive and synthetic dataset.

Proof of existence of metadata leak Figure 11 shows the observed metadata pattern on a given discussion in the sensitive dataset on both the XMPP reference server (Figure 11a) and Basic ORAM (Figure 11b). These are one dimensional figures with the time on the x-axis. Each vertical line shows the presence of a message from the given discussion received by the server at a time t. On Figure 11a, the height of the line represents the size of the message while the arrows up and down show a push and down action respectively. These figures present the activity of a discussion collected by the server over time. It forms the metadata pattern as presented in Definition 9. They show that even if ORAM servers still leak access time, they are not linked anymore with the other metadata types, and therefore they hold far less information to collect.



(a) XMPP Metadata Pattern. **(b)** ORAM Metadata Pattern.

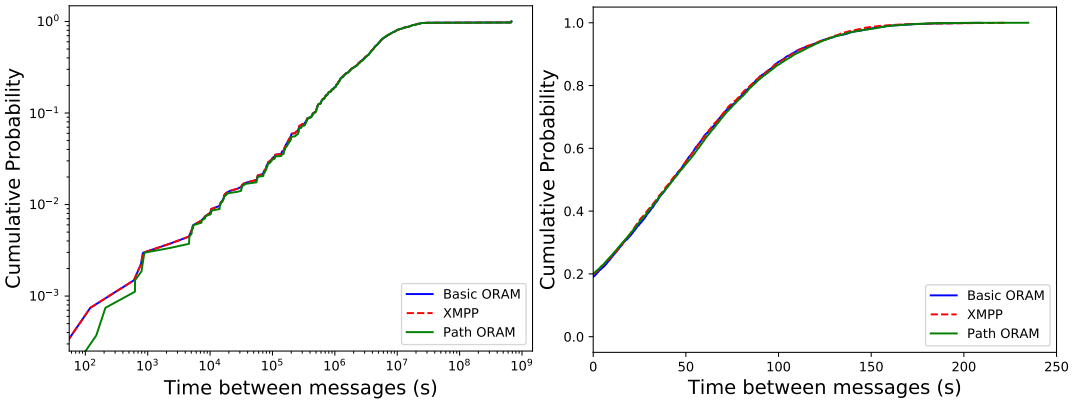
Figure 11: Metadata pattern collected on a given discussion on XMPP and ORAM server.

These figures provide a proof of the existence of metadata leak. The conceptual idea behind this result is that the more metadata you are able to link together, the more knowledge

on the user you are able to infer. In the next set of results, we generalise this concept to the full dataset.

Generalisation on metadata leak As we explained in the previous section, the only metadata available on ORAM server is the timestamp of collected messages. Moreover, because we work on the collection of metadata pattern we consider the distribution of the time between messages for our results.

On the contrary to what we could have expected, the distribution of the time between messages is the same on a given dataset among the different server. Figure 12 and Table 9 show this distribution for both ORAM and XMPP servers on the sensitive and synthetic dataset. On this figure (and on all the remaining figures in this section), the y-axis is the Empirical Cumulative Distribution Function (ECDF), that is to say, the probability the time between messages will take a value less than or equal to the value in the x-axis. The x-axis is in seconds. We note the XMPP curve on Figure 12 shows the ECDF of a normal distribution with a mean of 42 seconds. This result is expected because this is how was generated the messages from the synthetic dataset.



(a) Sensitive Dataset. **(b)** Synthetic Dataset.

Figure 12: Time between messages for ORAM based servers and XMPP server on the sensitive dataset (12a) and the synthetic dataset (12b).

Table 9: XMPP Server vs ORAM Servers.

	Sensitive Dataset		**Synthetic Dataset**	
Server	**Basic ORAM**	**Path ORAM**	**Basic ORAM**	**Path ORAM**
Critical distance δ_α	5.935×10^{-2}		2.444×10^{-2}	
KS Distance $D_{n,n'}$	8.816×10^{-3}	6.040×10^{-3}	1.500×10^{-2}	1.400×10^{-2}
Validity	Valid	Valid	Valid	Valid

Indeed, the purpose of an ORAM server is to hide the full metadata pattern, not the fact a given user is using the server. Furthermore, XMPP and Basic share approximately the same distribution whereas while statistically identical, the distribution from Path ORAM is a little bit different. It is explained by the design. XMPP and Basic ORAM share approximately the same number of transactions by access while Path ORAM has a more significant transaction complexity (see Figure 5). This bigger number of transactions is a requirement in

the Path ORAM protocol to ensure obliviousness while decreasing computation and bandwidth overhead. However, as we show in Figure 12 it does not change the distribution. Furthermore, Figure 12 shows synthetic and sensitive datasets work the same way. Therefore, for the following results, we consider only the sensitive dataset because the synthetic dataset does not hold the data regarding the complex metadata aggregation.

The previous test considers inter-message times pattern over time. When we consider the metadata pattern (MP) of the user's access (i.e., the pattern leaked by all the metadata available on a given server over the time), we show, for the full dataset, important differences in terms of distribution. Indeed, for Basic and Path ORAM servers, the MP leaked by the access over the dataset always remains the one presented in Figure 12. However, Figures 13 and Figure 14 show the differences with the XMPP pattern when we take into consideration the additional metadata types linked to the time. Since the distribution drawn from Basic and Path ORAM are statistically identical, we did not include graphs for both these figures only represent the distribution of Basic ORAM.

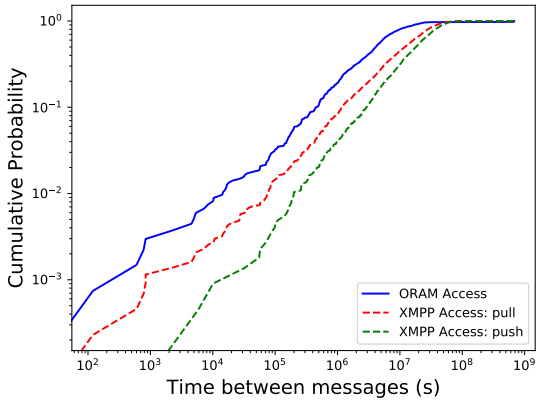


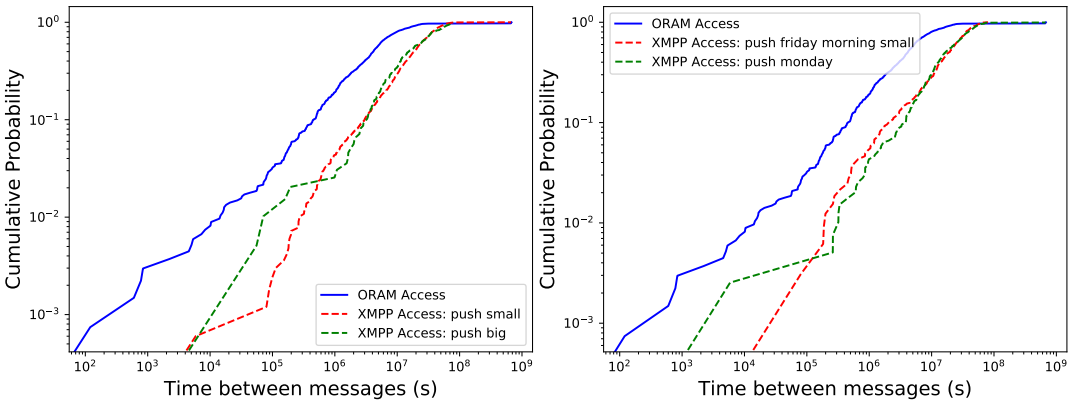
Figure 13: ORAM access vs XMPP access on sensitive dataset considering push and pull access.

Figure 13 presents the distribution of time between messages for push and pull access. Since, on the complete dataset, a given user is always the receiver of their correspondent, we needed to split the users into two parts: the users we target and the users we do not. Push and pull distributions are for the targeted users only and these results only show the distribution from users we target. This method does not represent a flaw in our demonstration because:

- Targeted users are random in the dataset and in a number big enough to maintain the statistical validity of the results.
- Regardless of the user targeted, the MP collected from ORAM servers remains the same.

Furthermore, when we consider a second and third layer of metadata type and for each different combination of these layers, the distribution changes every time as we show in Figure 14. This figure shows the access distribution for the XMPP server changes while the distribution from the ORAM servers remains the same.

Figure 14a considers push action for big and small messages. We define as "big message" the top 25% biggest messages and the "small message" the bottom 25% smallest messages in the dataset. Figure 14b shows the push action on Monday and the push action on Friday



(a) Push on small & big messages.

(b) Complex access on Friday & Friday.

Figure 14: ORAM access vs XMPP access on sensitive dataset considering complex aggregation of different metadata type: Push on big messages and small messages (14a), Push on Monday, Push on Friday morning with small messages (14b)).

morning (before 12 am) with small messages only. Every-time, the distributions from the XMPP server changes and are significantly different from each other while the distribution from ORAM servers remains identical.

These are significant results and show how aggregated metadata from different types on an XMPP server can change the access pattern of a user. Conversely, regardless of the type of access considered, on an ORAM server, all the access patterns remain indistinguishable from each other.

What these figures do not show is that if we consider a very precise access, the messages selected are not present in big enough number to allow the statistical test to remain valid. For instance, even considering the full dataset, we cannot compare the distribution of messages from the push action on Sunday before 8 am with small messages because it does not result in enough messages. Therefore, every metadata type you are able to add allows you to narrow your search down to the activity of your targeted users. As a conclusion, with these results we show:

- XMPP servers leak metadata.
- Leak from XMPP IM server can lead to privacy issue for the targeted user when considering complex aggregation of metadata types.
- ORAM based IM servers prevent the leak of metadata and are, therefore, a valid solution to privacy leaks through metadata.

7.4 Discussion

This section provides a general discussion of the work proposed in this paper.

Basic ORAM vs Path ORAM By construction, Basic ORAM and Path ORAM leak the same amount and type of metadata. The metadata leaked is the access time of the messages (see Figure 12). The access time distribution for a given dataset is thus the same for both systems. While being two fundamentally different constructs, Basic and Path ORAM provide the same obliviousness and are both able to prevent the leak of metadata.

Anonymity This paper does not consider anonymity as a possible solution to these privacy leaks. Although it can be easy to add it to these systems, it is not the purpose of an oblivious system to provide user anonymity. When we provide anonymity to the users of a server, we hide the real identity of the user, but we do not hide their actions. It is a significant limitation because using inference attacks make it possible to retrieve the real identity of the user and thus break user's anonymity. Therefore, users need to take care of compartmentalising their lives as much as possible. However, compartmentalising digital life requires a lot of security skills and is not always possible in real life. Furthermore, many services only make sense when the users are not anonymous or when they keep the same id between platforms such as health system, financial system, social network, and so on. On the contrary, by using an ORAM scheme, users are not anonymous, but the server does not know the actions they have been conducting. Moreover, the non-anonymous part is usually not an issue because a user's interaction with a specific server can be public. For instance, a user can have legitimate communication with the server that manages its medical data, its email or its enterprise IM server. As a conclusion, for these use cases, it is becoming more interesting to provide obliviousness than anonymity to assure user privacy.

Spectre & Meltdown While writing this paper, the Spectre [25] and Meltdown [28] attacks have been publicly disclosed. These attacks allow a user process to read any kind of data in the memory. It works with any kind of user-space program, including browsers' JavaScript engine. The attacks are based on hardware CPU vulnerabilities and affect every CPU vendor regardless of the operating system used. Our proposed attack makes the assumption that the attacker has root access on the server. Spectre and Meltdown show that an attacker might conduct this attack just with a regular user's rights. Therefore, it makes metadata collection easier than expected and a much bigger issue than we could have thought.

8 Conclusion

Privacy concerns about Cloud-based applications have been a growing issue in the last few years. In addition to content, metadata can also be used for gathering information about communicating entities. In this paper, we have introduced and formalised the notions of tangible and intangible metadata as well as the notion of metadata type. In the context of instant messaging, we have then proposed a new live forensic-based attack that allows an attacker with access to an IM server to retrieve intangible metadata by spying on the RAM process. The purpose of this attack is to be active as long as possible in order to accumulate information permitting meaningful inferences. We have developed a new ORAM scheme, Basic ORAM for the purpose testing our attack. By using this attack on both a classic IM server and two types of ORAM servers, the results show that the classic system does leak information while ORAM systems do not. Therefore, ORAM is an effective solution for providing privacy to the users of Cloud-based services. Furthermore, this results can easily be extended to similar application and dataset than share the same structure and approach. To the best of our knowledge, this work is the first to experimentally show the leak of information by metadata and to evaluate an ORAM server as a potential solution, notwithstanding performance issues.

Future work will consider other applications such as file storage, and audio/video calls where a single communication usually involves several servers or proxies. We are also on the lookout for obtaining real-life, large-scale complete datasets of messages/calls with

both content data and metadata to be able to analyse potential leaks in more detail. Finally, we plan to make a performance evaluation of state of the art ORAM systems for assessing their possible use in production environments.

Acknowledgement

This work was supported, in part by Irish Research Council grant GOIPG/2016/479 (research.ie), in part by Science Foundation Ireland grant 10/CE/I1855 to Lero — the Irish Software Research Centre (www.lero.ie) and in part by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero — the Irish Software Research Centre (www.lero.ie). The authors thank the anonymous reviewers for their helpful comments and suggestions. We also thank Thomas Laurent and Hilmi Egemen Ciritoglu for the valuable discussions and reviews.

References

[1] Balamurugan Anandan, Chris Clifton, Wei Jiang, Mummoorthy Murugesan, Pedro Pastrana-Camacho, and Luo Si. t-Plausibility: Generalizing Words to Desensitize Text. *Trans. Data Privacy*, 5(3):505–534, 2012.

[2] Daniel Apon, Jonathan Katz, Elaine Shi, and Aishwarya Thiruvengadam. Verifiable Oblivious Storage. In *Public-Key Cryptography–PKC 2014*, pages 131–148. Springer, 2014.

[3] Vanessa Ayala-Rivera, Patrick McDonagh, Thomas Cerqueus, Liam Murphy, and Christina Thorpe. Enhancing the Utility of Anonymized Data by Improving the Quality of Generalization Hierarchies. *Transactions on Data Privacy*, 10(1):27–59, 2017.

[4] Frank Block and Andreas Dewald. Linux Memory Forensics: Dissecting the User Space Process Heap. *Digital Investigation*, 22:S66–S75, 2017.

[5] Frank Block and Andreas Dewald. Linux Memory Forensics: Dissecting the User Space Process Heap. Technical report, CS-2017-02, 2017.

[6] Dan Boneh, David Mazieres, and Raluca Ada Popa. Remote Olivious Storage: Making Oblivious RAM Practical. Technical report, MIT, 2011.

[7] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[8] Hilmi Egemen Ciritoglu, Takfarinas Saber, Teodora Sandra Buda, John Murphy, and Christina Thorpe. Towards a Better Replica Management for Hadoop Distributed File System. In *BigData Congress*, pages 104–111, 2018.

[9] M. I. Cohen, D. Bilby, and G. Caronni. Distributed Forensics and Incident Response in the Enterprise. *Digital Investigation*, 8(SUPPL.), 2011.

[10] Michael Cohen. Rekall Memory Forensics Framework. *DFIR Prague*, 2014.

[11] Michael Cohen. Forensic Analysis of Windows User Space Applications Through Heap Allocations. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 237–244. IEEE, 2015.

[12] William W Cohen. Enron Email Dataset, 2009, 2009.

[13] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, and Ling Ren. Onion ORAM: A Constant Bandwidth and Constant Client Storage ORAM (without FHE or SWHE). *IACR Cryptology ePrint Archive*, 2015:5, 2015.

[14] Ran Dubin, Amit Dvir, Ofer Hadar, and Ofir Pele. I Know What You Saw Last MinuteThe Chrome Browser Case. *Black Hat Europe*, 2016.

[15] Georges Fanny. Self-representation and Digital Identity: A Semiotic and Quali-Quantitative Approach to the Cultural Empowerment of the Web 2.0. *Reseaux*, 154:165–193, 2009.

[16] Justin N Ferguson. Understanding the Heap by Breaking It. *black Hat USA*, pages 1–39, 2007.

[17] Craig Gentry et al. Fully Homomorphic Encryption using ideal Lattices. In *Stoc*, volume 9, pages 169–178, 2009.

[18] Ian Goldberg. Improving the Robustness of Private Information Retrieval. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 131–148. IEEE, 2007.

[19] Oded Goldreich and Rafail Ostrovsky. Software Protection And Simulation On Oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[20] Glenn Greenwald. *No place to Hide: Edward Snowden, the NSA, and the US Surveillance State*. Macmillan, 2014.

[21] Nguyen Phong HOANG, Yasuhito ASANO, and Masatoshi YOSHIKAWA. Your Neighbors Are My Spies: Location and other Privacy Concerns in GLBT-focused Location-based Dating Applications. *ICACT Transactions on Advanced Communications Technology*, 5(3):851–860, 2016.

[22] Fei Hong, Rui Liu, Liting Hu, and Yu Bai. Analysis and Characteristic at the Chat Session Level in Instant Message Traffic. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 1666–1669. IEEE, 2009.

[23] Yaoqi Jia, Tarik Moataz, Shruti Tople, and Prateek Saxena. Oblivp2p: An Oblivious Peer-to-Peer Content Sharing System. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 945–962, 2016.

[24] Parambir S Keila and David B Skillicorn. Structure in the Enron Email Dataset. *Computational & Mathematical Organization Theory*, 11(3):183–199, 2005.

[25] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. *arXiv preprint arXiv:1801.01203*, 2018.

[26] Susan Landau. Making sense from Snowden: What's significant in the NSA surveillance revelations. *IEEE Security & Privacy*, 11(4):54–63, 2013.

[27] Jure Leskovec and Eric Horvitz. Planetary-Scale Views on a Large Instant-Messaging Network. In *Proceedings of the 17th international conference on World Wide Web*, pages 915–924. ACM, 2008.

[28] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading Kernel Memory from User Space. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 973–990, 2018.

[29] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Privacy And Access Control For Outsourced Personal Records. In *36th IEEE Symposium on Security and Privacy*, 2015.

[30] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Maliciously Secure Multi-Client ORAM. In *International Conference on Applied Cryptography and Network Security*, pages 645–664. Springer, 2017.

[31] Frank J Massey Jr. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[32] Jonathan Mayer, Patrick Mutchler, and John C Mitchell. Evaluating the privacy properties of telephone metadata. *Proceedings of the National Academy of Sciences*, 113(20):5536–5541, 2016.

[33] Daniele Micciancio. Oblivious Data Structures: Applications to Cryptography. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 456–464, 1997.

[34] Muhammad Naveed. The Fallacy of Composition of Oblivious RAM and Searchable Encryp-

tion. *IACR Cryptology ePrint Archive*, 2015:668, 2015.

[35] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[36] Carol XJ Ou, Robert M Davison, Xuepan Zhong, and Yi Liang. Empowering Employees Through Instant Messaging. *Information Technology & People*, 23(2):193–211, 2010.

[37] Andrea Peterson. Why a Staggering Number of Americans have Stopped Using the Internet the Way they Used To. *Washington Post. May*, 13, 2016.

[38] Alexandre Pujol and Christina Thorpe. Dog ORAM: A Distributed and Shared Oblivious RAM Model with Server Side Computation. In *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*, pages 624–629. IEEE, 2015.

[39] Alexandre Pujol, Christina Thorpe, and Liam Murphy. Collecting Metadata on an Instant Messaging Server. In *European Conference on Cyber Warfare and Security*, pages 741–744. Academic Conferences International Limited, 2017.

[40] Tobias Pulls and Daniel Slamanig. On the Feasibility of (Practical) Commercial Anonymous Cloud Storage. *Trans. Data Privacy*, 8(2):89–111, 2015.

[41] 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.

[42] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. Technical report, IETF, 2011.

[43] Jitesh Shetty and Jafar Adibi. The Enron Email Dataset Database Schema and Brief Statistical Report. *Information sciences institute technical report, University of Southern California*, 4(1):120–128, 2004.

[44] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM With O((logN) 3) Worst-Case Cost. In *Advances in Cryptology–ASIACRYPT 2011*, pages 197–214. Springer, 2011.

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

[46] 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.

[47] Elaine Svenonius. Metadata and the World Wide Web. *SRELS Journal of Information Management*, 25(4):215–227, 1988.

[48] Vicenç Torra and Guillermo Navarro-Arribas. Data privacy: A survey of results. In *Advanced Research in Data Privacy*, pages 27–37. Springer, 2015.

[49] Sameer Wagh, Paul Cuff, and Prateek Mittal. Differentially Private Oblivious RAM. *Proceedings on Privacy Enhancing Technologies*, 2018(4):64–84, 2018.

[50] Aaron Walters. The Volatility Framework: Volatile Memory Artifact Extraction Utility Framework, 2007.

[51] Ian T Young. Proof Without Prejudice: Use of the Kolmogorov-Smirnov Test for the Analysis of Histograms from Flow Systems and other Sources. *Journal of Histochemistry & Cytochemistry*, 25(7):935–941, 1977.

[52] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting Square-Root ORAM: Efficient Random Access in Multi-Party Computation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 218–234. IEEE, 2016.