HABILITATION À DIRIGER DES RECHERCHES

Université de Nice Sophia-Antipolis

Security and Privacy for Emerging Technologies

Written by

Melek Önen

12 January 2017

N. ASOKAN	**Aalto University and University of Helsinki**	**Referee**
Claude CASTELLUCCIA	**INRIA**	**Referee**
Josep DOMINGO-FERRER	**Universitat Rovira i Virgili**	**Referee**
Frederik ARMKNECHT	**Universität Mannheim**	**Examiner**
Bruno MARTIN	**Université Nice Sophia-Antipolis**	**Examiner**

Acknowledgments

This manuscript is the result of more than a decade of research work with many people. I had the honor of collaborating with brilliant researchers ranging from PhD students to postdocs and professors. I would like to express my sincere gratitude to all these people who played a significant role in my academic career. I hope that our collaborations will continue. I also had the chance to make very good friends.

I would like to thank all the members of my committee for their feedback and discussion: N. Asokan, Claude Castelluccia, Josep Domingo-Ferrer, Frederik Armknecht and Bruno Martin.

I would like to thank EURECOM for the very good working conditions and I highly appreciate the unlimited help of Claire Cristofaro, Christine Mangiapan, Sophie Salmon, Audrey Ratier, Christine Roussel and Delphine Tales for their administrative support and the IT team. Without them, I would have had much less time for research.

I am indebted to my parents, my husband, my sisters and my friends for their encouragements and their endless support.

Last but not least I owe special thanks to my niece Eliz and my nephew Alaz who animated my work either with the unexpected drawings in some pages of my papers or with the noise they make while reading some papers.

Contents

Introduction

The purpose of this manuscript is to provide an overview of my research activities during the last ten years in the broad domain of privacy and security. With the significant progress achieved since many years in hardware and software technologies, communication among people became much easier and much cheaper. The ease of communication resulted in an exchange of large amount of information. The rapid changes in information technology which refers to the storage and processing of such information have raised serious security and privacy challenges and led to loss of users' control over their data.

The main goal of my research activities was (and still is) to come up with novel security and privacy solutions dedicated to new technologies. The methodology that has been applied with the upcoming new technology consists of first analyzing the security and privacy issues raised by the new technology and further designing and evaluating new and dedicated security or privacy preserving primitives.

The document first starts with the identification of the security and privacy challenges of opportunistic networks which leverage any "opportunity" to transmit communication packets to the destined nodes. Given the mobility of the nodes and the diversity of information used at the communication layer, such networks are exposed to various security and privacy threats within an environment without any a-prior trust relationship. Chapter 1 overviews various privacy preserving primitives proposed for different opportunistic forwarding strategies. This research work was jointly carried out with Dr. Abdullatif Shikfa, the first Ph.D. student I was co-advising with Prof. Refik Molva.

Chapter 2 gives an overview of the problem of privacy protection in online social networks. It mainly describes a study we conducted together with Dr. Leucio Antonio Cutillo and Prof. Refik Molva on the impact of social graph theory on the privacy of distributed online social networks. The chapter further outlines a dedicated privacy preserving primitive for picture sharing applications.

Finally, in chapters 3 and 4 the document focuses on the cloud computing technology which helps customers to outsource their massive data and computation with the various advantages of higher availability, reliability and flexibility. This new paradigm comes with high security and privacy exposures and the document briefly describes some solutions proposed not only to protect the privacy of the user but also to increase transparency with respect to the services of cloud providers. While chapter 3 mainly studies the problem of privacy preserving word search and describes three different solutions with different user settings, it also reviews two secure deduplication solutions designed together with Dr. Pasquale Puzio and Prof. Refik Molva. Chapter 4 tackles the problem of verifiability and reviews a verifiable storage and a verifiable word search solution jointly proposed with Dr. Kaoutar Elkhiyaoui and Monir Azraoui, the second Ph.D student I supervised with Prof. Refik Molva.

Chapter 1

Security and Privacy in Opportunistic Communications

In this chapter we focus on the problem of security and privacy in opportunistic communications which aim at exploiting all available communication opportunities to eventually deliver a message to its destination. Users' physical mobility is considered as an additional opportunity of bringing messages closer to the destination. This new communication model relying on users' cooperation and context information raises several security and privacy challenges which will be analyzed in this chapter. Section 1.1 introduces the different security and privacy problems for which the next sections briefly describe potential solutions: While section 1.2 tackles the problem of cooperation enforcement and presents a solution dedicated to a specific opportunistic forwarding algorithm, sections 1.3 and 1.4 focus on the problem of privacy and introduce two solutions that address the problem of user privacy while ensuring the opportunistic communication among nodes.

1.1 Problem Statement

1.1.1 Opportunistic forwarding

Opportunistic networks can be seen as an extension or a follow-up to mobile ad-hoc networks (MANETs) [1]. They inherit many of their characteristics such as:

- **the lack of infrastructure**: nodes form a temporary network without the help of any infrastructure and without any a-priori trust relationship. Therefore all nodes participate in the underlying communication operations.
- **the scarcity of resources**: mobile nodes are assumed to have limited resources (such as memory or battery) and therefore constantly need to optimize their usage.

On the other hand, although considered as **mobile**, MANETs very loosely support mobility: packet routing rely on a fixed path between the source and the destination and any modification of the network topology during the communication requires the recomputation of the entire path. On the contrary, opportunistic networks consider node mobility as an advantage since it is a new opportunity for messages to be delivered in an efficient manner. Opportunistic communication follows the *store, carry and forward* principle: when a communicating device receives a message, it stores it, carries it by using physical mobility, and forwards it when a communication opportunity arises. Therefore, opportunistic networks **do not rely on the existence of end-to-end connectivity** and thus are **delay-tolerant**. Furthermore, since messages are disseminated, destinations are

sometimes implicitly defined by their interests or their context rather than by an explicit address. Opportunistic forwarding solutions can be classified under three categories:

- oblivious forwarding protocols: such protocols can be considered as classical routing protocols where destination of messages is defined by an address. The approach is mainly based on epidemic forwarding which basically consists in forwarding a packet to all encountered nodes. The main advantage of epidemic forwarding is that it achieves almost always 100% delivery ratio with very few information required; on the other hand, the disadvantage is either the huge bandwidth waste or a potentially high delivery time. Solutions thus focus on limiting the flooding.

- context-based forwarding: in such protocols, the destination is defined through its context. The context of a message mainly consists of the profile of the destination. The profile of a node consists of information related to the physical node and includes personal information about the user, its location, its activities, social relations, etc. Based on this contextual information, forwarding nodes decide on which next hop matches the context of the message the best and forward the message to this particular node. Messages eventually reach the destination which matches all the context.

- content-based forwarding: in such a communication paradigm, messages do not define a destination at all: content is forwarded based on nodes' interests. Such protocols rely on a collapsed architecture where information concerning the application or the network operation is at the same level.

1.1.2 Security issues

The specific constraints of opportunistic networks, namely, the lack of infrastructure, the lack of end-to-end connectivity and the collapsed architecture, raise new challenges in terms of security, trust and privacy.

- **trust and cooperation**: opportunistic communications (especially, oblivious forwarding protocols) do not assume a routing infrastructure and messages are forwarded by all communicating devices, hence cooperation between devices is crucial. On the other hand, these devices having limited resources, may require some incentives to cooperate and enable the communication.

- **confidentiality and privacy**: applying traditional encryption primitives is not straightforward since while data often needs to be encrypted in an end-to-end way, encryption will hinder basic communication mechanisms such as packet forwarding that need to be able to parse the content of packets.

- **authentication and integrity**: because of the underlying collapsed architecture, the integrity of messages becomes crucial for the correct delivery of the messages. The lack of a security infrastructure increases the problem of authentication even more.

The next sections tackle these problems and propose a secure forwarding solution dedicated to each category of forwarding protocols.

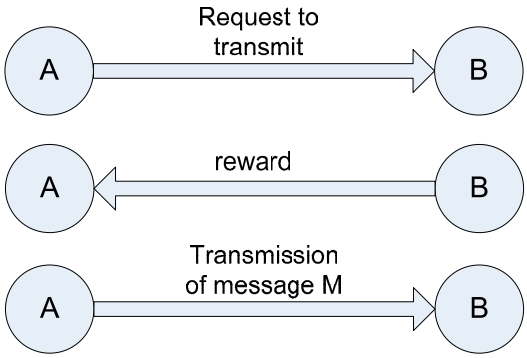

Figure 1.1: The basic hot potato forwarding mechanism

1.2 Cooperation Enforcement

As in MANETs, opportunistic networks involve all available nodes on packet forwarding. Scarcity of resources would inherently foster them to behave selfishly. In order to reduce the effect of selfishness which may have a strong impact on the performance of the network [2], in [3], we propose a new cooperation enforcement scheme which is based on a simple technique called the hot-potato forwarding whereby, to receive a packet, potential recipients must first pay the sender a reward without prior knowledge about the packet. Additionally, the underlying optimistic fair exchange protocol designed for this forwarding solution resolves the fairness problem that is inherent to peer rewarding schemes.

1.2.1 Solution Overview

We propose a new forwarding technique whereby as in existing incentive mechanisms such as [4] or [5], to receive a packet, each node must first deliver an advance reward to the sender without any prior knowledge about the packet. Upon reception of the packet, if the recipient is not the destination, the recipient forwards the packet to the next hop en route to the destination in order to recover its rewarding amount. This new approach is illustrated with a simple protocol in figure 1.1. The motivating idea behind this approach is quite similar to hot potato routing [6] where packets must keep on moving until they reach their destination. The hot potato approach enforces cooperation among nodes since potential recipient nodes are motivated to deliver the reward for a packet based on the fact that this is the only way they can receive traffic destined to themselves and in case packet is destined to another node the rational recipient is also motivated to forward it to recover the reward.

1.2.2 Security Requirements

The simple version of the protocol depicted in figure 1.1 can be a potential target for Denial of Service: an attacker may generate some bogus messages in order to earn some rewards and use the resources of other legitimate nodes; moreover, an intermediate node may also forward the same message to several nodes. On the other hand, once a certain node receives the reward, it may refrain from forwarding the message and end the exchange protocol. Based on these observations, four specific security requirements are defined:

- **cooperation enforcement**: when a node paid for a packet and received it, it must forward it;

- **protection against poisoning attacks**: nodes must be prevented from sending bogus packets;
- **protection against cheating actions**: nodes must be prevented from unduly earning rewards by sending the same packet several times;
- **fairness**: transmission of a packet subsequently to the reception of the corresponding reward must be assured.

1.2.3 The rewarding mechanism

We consider the scenario whereby a source node S wishes to send a packet m to a destination D. We assume that node S as well as all intermediate nodes can find out the next hop en route to the destination node D. The proposed protocol also relies on the existence of a trusted third party (TTP). However, the TTP does not initiate any communication and is only required in case of conflicts; it is also responsible of crediting and debiting nodes with respect to their rewards. All nodes store the TTP's public key as well as an individual pair of public and private keys in their memory. When S wishes to send m to D, the first intermediate node I_1 pays S for a certain amount r. Then, in order to get a compensation, when I_1 contacts the next intermediate node I_2, it asks for a larger amount $r + c$. When the packet reaches D, D recovers its rewards by further contacting the TTP which credits D and debits S for the corresponding amount. Thanks to the proposed rewarding mechanism, intermediate nodes that forward packets are rewarded to compensate the energy they deploy to do so. The reward paid by a node A to another node B for a message M is denoted by $reward(A, B, H(M), r)$ where H denotes a cryptographic hash function. The underlying rewarding mechanism exhibits the following properties:

- **No anonymity**: a reward is tightly bound with the payer's and payee's identities and the hash value of the corresponding message.
- **No double-spending and non forgeability**: rewards resulting from duplication or copying of valid rewards or forging will be detected and funding of such rewards by the TTP will be prevented.
- **No re-usability**: each reward is tightly bound to a single message exchange. Nodes periodically contact the TTP in order to transform earned rewards into usable rewards. Only the TTP can perform such operations.

Thanks to these properties and the existence of the TTP, a node cannot earn rewards by just sending bogus packets to some other nodes and therefore poisoning attacks are inherently prevented.

1.2.4 The exchange protocol between two nodes

The exchange protocol is accomplished in five steps. The protocol is illustrated in Figure 1.2. As an initial step, node A sends the hash value of m with the requested rewarding amount. These two elements are signed with its private key. Upon reception of the request, node B first verifies the signature and checks if this request is not a replay (B keeps a temporary history of previous hash values). If the signature is valid and B has enough rewards and resources to receive the message, then node B agrees to receive m and sends its signed agreement. Once node A successfully verifies B's signature, it selects a random secret key K_A and encrypts m with this key using a symmetric encryption algorithm such as AES [7]. It then encrypts K_A with the public key of the TTP and signs all these data pieces with its private key. At step 3, node B sends to node A the requested

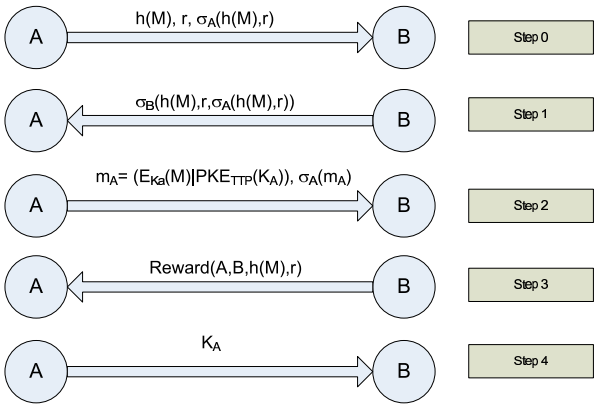

Figure 1.2: The exchange protocol between nodes A and B

reward. When A receives the reward corresponding to m, it verifies the validity of the reward. If the reward is not valid, then node A contacts the TTP to resolve this conflict. Otherwise, node A proceeds to step 4 at which it sends K_A to node B who finally can decrypt and retrieve m; node B verifies the integrity of m with respect to the reward using the digest value $H(M)$ included in the reward. If the message resulting from decryption does not match the reward, then B contacts the TTP to resolve the conflict. Moreover, if B does not receive the key K_A, then it can contact the TTP in order to receive K_A from the TTP. If on the other hand the decrypted message and the reward match, the communication ends with success.

1.2.5 Evaluation

Thanks to the underlying rewarding mechanism, cooperation becomes mandatory: Nodes have no choice but to accept receiving all incoming packets if they want to be sure to receive packets that are intended to them. Once they received packets, they must forward those that are not intended to them to recover rewards spent before the reception. If packets are not forwarded, then they simply lose some rewards and thus are immediately punishing themselves. Additionally, since only the source is charged for sending the packet, a node will not have any incentive to send bogus messages for the purpose of poisoning. Furthermore, duplicate transmissions would also be detected by the TTP due to the strong bounding between each message and the corresponding reward. Finally, the proposed protocol is defined to be fair if at the end of the protocol, node A receives its reward and node B receives the message. Nodes A or B contact the TTP only if a problem occurs during the exchange. They also might cheat by contacting the TTP to reclaim the resolution of an inexistent conflict. In order to evaluate the fairness of the protocol, we consider all possible communications between the TTP and each of the nodes at the end of each step and prove that the protocol is complete. More details can be found in [3].

1.2.6 Discussion

Existing optimistic fair exchange protocols such as those proposed in [8] usually implement a verifiable escrow scheme whereby the public encryption of the exchanged message/signature is binded with the context and enables the receiver to verify that it is indeed an escrow of the to-be-exchanged message. To ensure a timely termination, the protocol also includes three sub-protocols during which the TTP is contacted, namely: an *abort* protocol for the initiator and two *resolve* protocols, one for each party. While the proposed forwarding solution could have used the

fair exchange solution in [8] (especially, the solution for the fair exchange of digital content, see section 8.2), we believe that the fairness of our solution is ensured thanks the assumptions made on the underlying rewarding mechanism and on the role of the TTP:

- The reward is binded to the IDs of both parties and the exchanged message;
- As opposed to existing solutions, the TTP is also in charge of both crediting and debiting nodes. Whenever A is credited with a certain reward from B, B will immediately be debited. Additionally, A is only credited if she provides the message corresponding to the hash value defined within the reward.

1.2.7 Summary

In conclusion, thanks to the newly proposed rewarding mechanism which forces nodes to place an advance payment of rewards to receive traffic destined to themselves, cooperation among nodes becomes mandatory. The reliance of the protocol on a trusted third party is alleviated by the fact that the TTP is only involved in case of conflicts. While the fairness of the protocol is analyzed using a logical chart enumerating all cases, As mentioned in the previous section, an existing fair exchange protocol based on verifiable escrow schemes (such as the one in [8], section 8.2) could have also been used as a black box. Additionally, it would have been interesting to analyze the rewarding of the fair nodes given the recent invention of Bitcoin[1].

1.3 Secure Context-based forwarding

In this section, we analyze the security and privacy challenges raised by context based forwarding protocols which exploit any information including users' profiles to transmit messages. We then overview the solution proposed in [9] and further improved in [10].

1.3.1 Security and privacy requirements

As mentioned in section 1.1.1, context based forwarding [11, 12] consists of forwarding messages based on the context (e.g. location, workplace, social information, etc.) instead of explicit addresses. Each message is associated with its context which corresponds to the profile of its destination and nodes take their forwarding decisions by comparing their own profile and the profile of their neighboring nodes with the context of the message: the larger the context shared with a node, the higher the chances to reach the destination. Figure 1.3 illustrates a context based forwarding protocol where message M_2 is destined to nodes with profiles including the attributes {(Name, Alice);(Workplace, INRIA); (Status, Student)} and whose payload is *"I love you"*. The first and foremost security requirement of any communication protocol including a context-based forwarding protocol is confidentiality of the content of the message to be delivered: access to the payload of the message should only be authorized to destined nodes. Therefore the payload of the message should be encrypted. However, since context based forwarding cannot rely on any security infrastructure and on any end-to-end key management mechanisms, source nodes should be able to encrypt the data without sharing any key with the destination in advance. Furthermore, a threat that is specific to context-based forwarding is privacy leakage: a node's profile should not be exposed to other nodes in clear, even if this helps forwarding. Therefore, an intermediate node should be able to detect matches between its profile and the destination of the message

[1] https://bitcoin.org

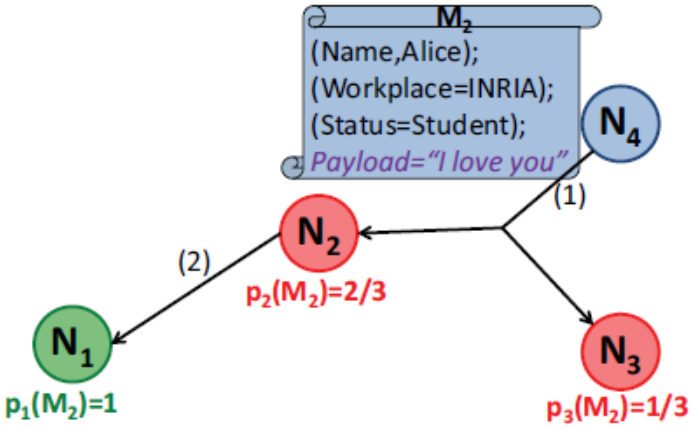

Figure 1.3: Context based forwarding

only, and other non-matching attributes should remain secret. Hence, revealing to another node a shared attribute is acceptable from a privacy perspective. Moreover, as for **payload confidentiality**, encryption of the header cannot rely on an end-to-end key management mechanism because of the opportunistic nature of the communication medium. To summarize, in order to achieve data confidentiality and user privacy, a secure context based forwarding protocol should implement the following primitives:

- ENCRYPT_PAYLOAD which takes as input the payload of the message and some public keying material and outputs an encrypted payload;
- ENCRYPT_HEADER which takes as input the header of the message and some public keying material and outputs and encrypted header; this primitive should enable forwarding nodes to compare their profile with the encrypted context in order to correctly forward packets;
- MATCH_HEADER which is executed by any forwarding node to determine matching attributes without revealing any other information;
- DECRYPT_PAYLOAD which allows the decryption of the message only by nodes which actually have the required attributes, hence the corresponding keying material for decryption.

1.3.2 Background

As previously mentioned, the ENCRYPT_PAYLOAD primitive should assure end-to-end confidentiality without any interaction between the source and the destination. Attribute based encryption [13] nicely fits such requirements since it is an asymmetric mechanism where the encryption key is public. We propose a simple attribute based encryption solution based on refinements of identity based cryptography [14] to allow any node to compute an encrypted version of the message. Note that any attribute based encryption solution can be used to assure payload confidentiality.

1.3.2.1 Identity based encryption (IBE)

As previously introduced, Identity based encryption (IBE) [14] is an asymmetric cryptographic encryption solution where the encryption public key is the node's identity which is known and represented by a string. Hence, IBE consists of the following algorithms:

- Setup_{ibe}: this randomized algorithm takes as input a security parameter ζ and returns a master key MK_{ibe} that is only known by the party who generates private keys using the subsequent $\mathsf{Extract}_{ibe}$ and some public parameters $param_{ibe}$ which basically define the message spaces.
- $\mathsf{Extract}_{ibe}$: given public parameters $param_{ibe}$, master key MK_{ibe} and an arbitrary identity $id \in \{0,1\}^*$, this algorithm returns a private key SK_{id} corresponding to id which is defined as the public key.
- Enc_{ibe}: this randomized algorithm encrypts a given message m with public key id and returns an encrypted message c.
- Dec_{ibe}: this deterministic algorithm takes as input ciphertext c, a private key SK_{id} and $param_{ibe}$ and returns a cleartext m which is the decryption of c which was early encrypted with public key id.

1.3.2.2 Public Key Encryption with Keyword Search (PEKS)

Introduced by Boneh et al. in [15], PEKS allows a node to search if a keyword exists in an encrypted data without disclosing any additional information than the result of the search. The node can perform this test only if it possesses a correct trapdoor computed from its private key and the word. PEKS consists of the following four algorithms:

- KeyGen_{PEKS}: this first algorithm generates a pair of public and private keys (A_{pub}, A_{priv}) given a security parameter ζ.
- PEKS: this randomized algorithm takes as inputs public key A_{pub} and a keyword ω and returns ω' a PEKS encryption of ω.
- $\mathsf{Trapdoor}$: this randomized algorithm takes as inputs private key A_{priv} and a keyword ω and returns trapdoor T_ω.
- Test_{PEKS}: this algorithm takes as input public key A_{pub}, an encrypted keyword ω' and the corresponding trapdoor T_ω and outputs 1 if $T_\omega = \mathsf{Trapdoor}(A_{priv}, \omega)$ and $\omega' = \mathsf{PEKS}(A_{pub}, \omega)$. Otherwise, the output is 0.

1.3.3 Privacy preserving context based forwarding

The proposed privacy preserving context based forwarding protocol is subdivided into three phases:

- **Setup phase**: during this phase each node N_i holding a certain number of attributes $\{Att_{N_i,j}\}$ receives the corresponding secret keys $\{SK_{i,j}\}$ from a trusted third party (TTP).
- **Forwarding phase**: whenever node S wants to send a message m to destination D whose profile is defined with k attributes that are $\{Att_{D,j}\}_{1 \leq j \leq k}$, it encrypts the payload $\mathcal{P}$ with ENCRYPT_PAYLOAD and the header with ENCRYPT_HEADER described in the next sections. To make the forwarding decision, all candidate forwarding nodes receive the

header $\mathcal{H}(m)$ and execute the MATCH_HEADER primitive in order to securely compare their profile with the attributes in the header of the message: they compute the matching ratio and send the result to the previous node together with some proof. The previous node makes the forwarding decision accordingly.

- **Message decryption phase**: whenever m reaches its final destination D, ie. whenever the matching ratio is 1, D decrypts the payload $\mathcal{P}(m)$ with DECRYPT_PAYLOAD.

1.3.3.1 Setup phase

Using Setup_{ibe}, TTP generates the master key and a pair of public/private keys (TTP_{priv}, TTP_{pub}) for itself. For each node N_i holding attribute Att_j, TTP first generates the private key $SK_{Att_{i,j}}$ using $\mathsf{Extract}_{ibe}$. TTP also generates all corresponding trapdoors $\{T_{i,j}\}$ by calling $\mathsf{Trapdoor}(SK_{TTP}, Att_{i,j})$ and sends it to the corresponding user. TTP can now go offline and the forwarding phase can start.

1.3.3.2 Forwarding phase

The forwarding protocol first starts by encrypting the payload of the message with ENCRYPT_PAYLOAD and the header regrouping attributes $\{Att_{D,j}\}$ with ENCRYPT_HEADER as follows:

- ENCRYPT_PAYLOAD: this algorithm takes as inputs the payload $\mathcal{P}(m)$ to be encrypted together with a set of attributes $\{Att_i\}$ and simply calls Enc_{ibe} with payload $\mathcal{P}(m)$ and the sum of attributes, namely $\sum H_1(Att_i)$.
- ENCRYPT_HEADER: this algorithm takes as input the header $\mathcal{H}(m)$ of message m and calls $\mathsf{PEKS}(TTP_{pub}, Att_{D,j})$ for each attribute $Att_{D,j}$ in the header. All encrypted attributes are concatenated and included in the new header $\mathcal{H}(m')$.

As the first forwarding node, S sends the encrypted header to neighboring nodes. Each neighboring node computes the matching ratio between their profile and the encrypted header message using MATCH_HEADER as follows:

- MATCH_HEADER: this algorithm executed by each candidate node N_i takes as input the encrypted header $\mathcal{H}(m')$ regrouping all encrypted attributes and all trapdoors $\{T_{i,j}\}$ received from TTP and calls $\mathsf{Test}_{PEKS}(T_{i,j}, Att'_{D,j})$ for all combinations of pairs of trapdoors and attributes in the message. Whenever Test_{PEKS} returns 1, MATCH_HEADER increments a counter r, initially set to 0, by 1. Finally, r is divided by the number of attributes in the header. For example if two attributes out of 3 match with the header's, then MATCH_HEADER returns $2/3$.

S further forwards the encrypted payload $\mathcal{P}(m')$ to the node with the highest matching ratio. Each node on the path to the destination enters this same phase which ends whenever the computed matching ratio equals 1.

1.3.3.3 Decryption phase

When destination node D receives the encrypted payload, ie. the matching ratio equals 1, D calls DECRYPT_PAYLOAD as follows:

- DECRYPT_PAYLOAD: this algorithm takes as inputs the ciphertext $\mathcal{P}(m')$ and a set of private keys $SK_{D,i}$ corresponding to attributes Att_i and calls Dec_{ibe} with $\mathcal{P}(m')$ and the addition of private keys, namely $\sum(SK_{D,i})$.

1.3.4 Summary

Thanks to the use of a simple attribute based encryption which is an instantiation of identity based cryptography in a multi-user setting, the proposed solution ensures data confidentiality defined as **payload confidentiality** in this section. Concerning user privacy, the solution enables intermediate nodes to discover matching attributes between the context of a message and their own context, while preserving the secrecy of non-matching attributes of the message. The solution is based on a searchable encryption scheme. More details on the solution can be found in [9]. Additionally in [10], we propose a mechanism to guarantee the computation of the matching ratios. The proof of correctness is based on the preimage security properties of cryptographically secure hash functions and on the use of counting Bloom filters [16] that efficiently prevent a malicious node from tampering with the ratio computed over encrypted data.

1.4 Privacy preserving content based Publish/Subscribe networks

1.4.1 Privacy requirements

In this section, we focus on the problem of privacy and security in content based forwarding protocols. Messages in content-based communications are simply sent to all nodes interested in the content. These interests are independent from the context (ie. nodes' profiles) and are subject to frequent changes. The **Publish/Subscribe paradigm** [17, 18] whereby communicating parties are loosely coupled and where messages are forwarded based on the content solely, can be considered as a concrete example of content-based communication. Because the privacy preserving context-based protocol described in the previous section, falls short in addressing the scalability and dynamicity requirements, we define the privacy and security requirements dedicated to this new paradigm and further describe a solution initially proposed in [19] and improved in [20].

1.4.2 Network model

In a typical Publish/Subscribe system a node can take three different roles:

- as a **Publisher** P_i, it publishes content through *event notifications*;
- as a **Subscriber** $S_{i,j}$ it declares its interests in a certain content through *subscription filters*;
- and finally as a **Broker** B_k it basically disseminates messages between publishers and subscribers.

While publishers and subscribers can be considered as the end-users of the system, brokers are intermediate nodes which actually deliver publishers' content to the interested subscribers efficiently. Instead of using recipients' addresses, brokers forward data segments based on their content and on the interest of recipients. For the sake of simplicity, we consider a network with one publisher and several subscribers and the network is modeled as a tree whose root node corresponds to publisher P_i: each leaf node is assigned to a subscriber S_j (whether interested in P_i's content or not) and intermediate nodes correspond to brokers. Note that a publisher P_i can be considered as subscriber or broker in another tree. Brokers construct their forwarding tables FT_k through the Build_FT primitive based on subscribers' subscription filters. Whenever broker B_k receives an event notification e composed of an attribute defining the content and the actual payload, it uses the Lookup_FT primitive to check whether e's attribute is equal to one of FT_k's rows corresponding to subscription filters. If $e's$ attribute is equal to one filter, then B_k forwards

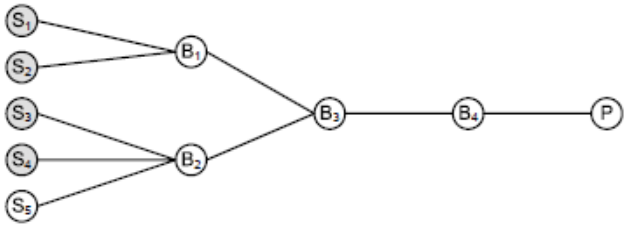

Figure 1.4: Content based Publish Subscribe Networks

the segment to the corresponding child node (defined in this row) in the path to the interested subscriber. Figure 1.4 illustrates a simple publish subscribe network with a single publisher and five subscribers. All subscribers except S_5 are interested in P's content and brokers B_1, B_2, B_3 and B_4 basically make sure that publisher P's notifications are delivered to subscribers S_1, S_2, S_3 and S_4.

1.4.3 Privacy requirements

We assume that nodes do not trust each other and are considered as being honest-but-curious. While nodes do not deviate from the designed protocol, they are interested to capture as much information as possible on the content of the data and/or the interest of subscribers. Hence, privacy of the nodes becomes an important concern in such a new communication paradigm. For example, subscribers do not want to expose their interest against any other node (including publishers); on the other hand, publishers do not want to reveal the content of their events to curious brokers or other publishers. The main requirements to ensure nodes' privacy also raised by [21] are the following:

- **Information confidentiality**: Publishers' event notifications should remain confidential with respect to brokers and unregistered subscribers. Nevertheless, brokers should still be able to evaluate these encrypted notification against subscription filters without leaking information on the corresponding content.
- **Subscription confidentiality**: Subscribers sending a subscription filter do not want to disclose their interests to any other node including publishers. Similarly to the previously described requirement, brokers should nevertheless be able to match a content with an encrypted subscription without disclosing the subscription filter.

1.4.4 Security primitives for content based Publish/Subscribe networks

In order to assure these two requirements, a content based Publish/Subscribe protocol should define the following algorithms:

- Enc_f: this algorithm executed by a subscriber takes as input some keying material and a subscription filter f and returns an encrypted subscription filer f'.
- Enc_n: this algorithm used by a publisher outputs an encrypted version of a notification n' given the cleartext notification n and some keying material.
- Build_FT: this algorithm allows a broker to build a forwarding table. Given a forwarding table FT and an encrypted subscription filter f', this algorithm should return the resulting updated routing table FT'.

- Lookup_FT: upon reception of an encrypted notification n', a broker calls this algorithm which takes the notification and the forwarding table FT on inputs and returns the id of the child node to forward n' to.

1.4.5 Background: Commutative encryption

The basic idea behind our solution is to use a new encryption primitive called multiple layer commutative encryption (MLCE) to allow broker nodes in charge of routing protected messages to perform secure transformations and transfer the message correctly without having access to it. A multiple layer commutative encryption allows the encryption and decryption of data with multiple keys at any order thanks to its commutative property. We define a multiple layer commutative encryption (MLCE) with the following algorithms:

- KeyGen_{MLCE}: This randomized algorithm takes as input a security parameter ζ and returns a pair of encryption and decryption keys (EK, DK).

- Enc_{MLCE}: This algorithm takes as input an encryption key EK and a message m and returns a ciphertext m'. Thanks to its commutativity, given two encryption keys EK_1 and EK_2 we have:

$$\mathsf{Enc}_{MLCE}(EK_1, \mathsf{Enc}_{MLCE}(EK_2, m)) = \mathsf{Enc}_{MLCE}(EK_2, \mathsf{Enc}_{MLCE}(EK_1, m))$$

- Dec_{MLCE}: This algorithm takes as input a decryption key DK corresponding to an encryption key EK and a ciphertext m' and returns a decrypted message m such that:

$$\mathsf{Dec}_{MLCE}(DK, \mathsf{Enc}_{MLCE}(EK, m)) = m$$

  Similarly to Enc Dec is also commutative; hence:

$$\mathsf{Dec}_{MLCE}(DK_1, \mathsf{Dec}_{MLCE}(DK_2, d)) = \mathsf{Dec}_{MLCE}(DK_2, \mathsf{Dec}_{MLCE}(DK_1, m'))$$

For our privacy preserving publish/subscribe solution, as a concrete example, we use the Pohlig-Hellman (PH) cryptosystem [22] which by definition is commutative:

- KeyGen_{PH} takes as input a large prime number q and returns the pair (EK, DK) such that $EK.DK \equiv 1 \mod (q-1)$.

- Given message m and encryption key EK, Enc_{PH} returns encrypted message $m' = m^{EK} \mod q$.

- Similarly, given ciphertext m' and decryption key DK, Dec_{PH} returns $m = m'^{DK} \mod q$.

1.4.6 Description of the solution

For the sake of simplicity, we propose to implement a privacy preserving Publish/Subscribe protocol with a two-layer commutative encryption scheme. As a first phase, every node N_i runs a key agreement protocols with some other nodes as follows:

- If N_i is a leaf node, N_i shares a pair of encryption/decryption keys $(EK_{i,j}, DK_{i,j})$ with its parent node N_j and another pair of keys $(EK_{i,k}, DK_{i,k})$ with its grand-parent node;

- if N_i is an intermediate node, in addition to the keys shared with leaf nodes if these are its children or grand-children, N_i also shares a pair of keys with this grand-parent only.

- if N_i is the root of the tree, ie. the publisher, it shares a different pair of keys with all its children and its grand-children.

Subscribers S_i located at leaf nodes first need to declare their interest f using Enc_f and broker nodes construct their forwarding tables using Build_{FT} accordingly.

- Enc_f: executed by subscriber S_i, this algorithm takes as input the two keys $\{EK_{i,j}\}$ that S_i shares with its parent and grand-parent nodes and filter f_i and recursively calls Enc_{MLCE}. Therefore, Enc_f returns a two-layer encryption of f_i.

- Build_{FT}: on reception of an encrypted filter from N_i, N_j removes one encryption layer using Dec_{MLCE} with the key it shares either with its child node (if this is a leaf node) or its grand-child node and checks if this value already exists in its forwarding table RT_j; if it is not the case it adds an additional row $f_i' \rightarrow N_i$; otherwise it adds the id of N_i to the already existing row. Build_{FT} further re-encrypts the filter with the encryption key shared with its grand-parent node (with its parent node if this one is the root of the tree) and returns the new table RT_j' and the partially encrypted filter f_i'. This filter is further forwarded to N_i's parent node.

Publisher P_i further encrypts its event notification n_i using Enc_n and the content distribution phase can start. Whenever broker nodes receive encrypted notifications, they run the Lookup_FT in order to forward them to the destined subscribers.

- Enc_n: executed by publisher P_i for each of its grand-children nodes, this algorithm calls $\mathsf{Enc}_{MLCE}(EK_j, \mathsf{Enc}_{MLCE}(EK_k, n_i))$ where n_i is P_i's event notification and $EK_{i,j}$ and $EK_{i,k}$ are the encryption keys shared with the grand-child node and its parent node (ie., one of its children node), respectively.

- Lookup_FT: this algorithm is called by an intermediate node N_j and takes as input its forwarding table FT_j, its keying material and some encrypted notification n_i' received from its parent node. It first calls Dec_{MLCE} with the encrypted notification n_i' and decryption key shared with its grand-parent node (or with its parent node if this one is the root of the tree). Once one encryption layer is removed, Lookup_FT checks if the resulting partially encrypted notification matches (is equal to) some partially encrypted subscription filter f_l'; if there is a match, the algorithm encrypts the notification with the key shared with the corresponding grand child node.

1.4.7 Summary

Figure 1.5 illustrates the overall protocol. The security of the scheme relies on the number of encryption layers and on the network topology. The MLCE scheme requires a topology-dependent key management solution. More details on the solutions can be found in [20]. In [23], we describe a solution to bootstrap security associations along with a secure neighborhood discovery. This new key management solution enhances the security of the previously described solution.

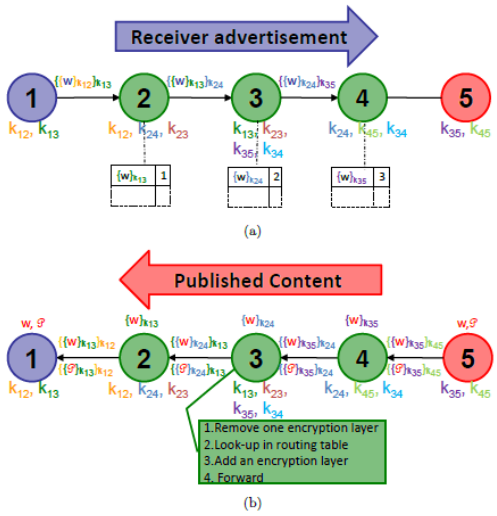

Figure 1.5: MLCE based content-based forwarding

1.5 Conclusion

This chapter analyzed the main security and privacy issues raised by different categories of opportunistic forwarding protocols. As forwarding of messages is performed by all nodes and not by a dedicated infrastructure of routers, cooperation among nodes become mandatory. We summarized the cooperation enforcement solution based on the hot-potato principle. We then focused on the protection of the information used to correctly and efficiently forward the message to its destination. The first solution allows intermediate nodes to forward packets based on the context by securely computing a matching ratio between the attributes of the profile of the destination of the packet and their profile. Such a computation is performed while preserving the confidentiality of non matching attributes. The second solution aims at protecting content based forwarding algorithms for Publish/Subscribe networks whereby packets are routed following subscription filters. Thanks to the newly proposed multi-layer commutative encryption, brokers (intermediate nodes) can perform secure transformations without having access to the data that is being transferred. Intermediate nodes can indeed remove or add an encryption layer without destroying the others and hence perform aggregation, routing tables building or look-up on private data protected by the other layers.

Chapter 2

Security and Privacy in P2P based social networks

Online Social Networks (OSNs) are nowadays extremely popular and some of them such as Facebook[1] or LinkedIn[2] are even inescapable with respect to the interpersonal communication. Unfortunately such networks severely suffer from the centralized control on users' data and its potential misuse. As an answer, several distributed OSNs propose to design new applications based on a distributed peer-to-peer (P2P) architecture. Some of them leverage real life social links to construct a network with trusted peers where the correct execution of any network/application operation depends on users' behavior. In this chapter, we analyze a particular P2P based online social network called *Safebook* (section 2.1) and present a study (see section 2.2) that shows that the security and privacy degree of such solutions strongly depends on the theoretical properties of the social graph representing friendship relations between users. We further outline a dedicated usage control solution for picture sharing applications in online social networks (section 2.3).

2.1 Safebook

In [24], authors porpose a P2P online social network named Safebook. The main aim of Safebook is to avoid any centralized control over user data by service providers. Safebook relies on the cooperation among its users. The correct execution of different services depends on the trust relationships among nodes which are by definition deduced from real-life social links. In order to protect users' privacy, Safebook defines a particular structure named as **Matryoshka** ensuring end-to-end confidentiality and providing distributed storage while preserving privacy. As illustrated in figure 2.1, a user V is assigned to a Matryoshka which is composed of several nodes organized in concentric shells. Nodes in the first shell are the real life friends of V and store her profile data to guarantee its availability. For this reason, nodes in the innermost shell are also called **mirrors** and serve requests if V is offline. Since this new storage solution inherently reveals who U's friends are and therefore discloses some privacy sensitive information, several multihop paths, **chains** of trusted friends, are built to protect such information: every user's friend selects among her own friends one or more next hops that are not yet part of the Matryoshka. The same process is executed several times until reaching the depth of the tree and the nodes which lie in the outermost shell are called the **entrypoints**. Regarding data retrieval, whenever a user U looks for V's data, her request is served by the entrypoints of V's Matryoshka and forwarded to the mirrors along

[1] http://www.facebook.com
[2] http://www.linkedin.com

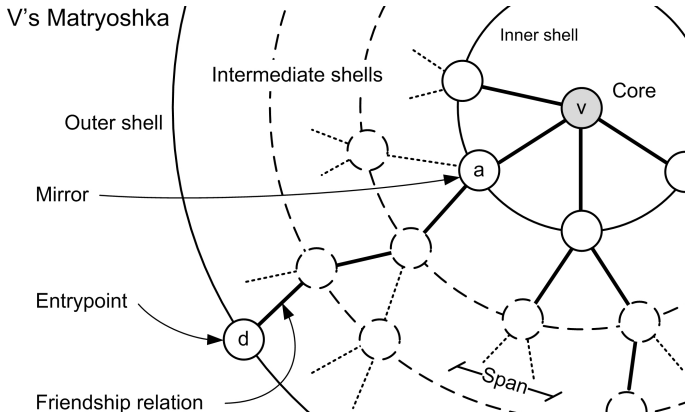

Figure 2.1: The Matryoshka graph of a user $\mathcal{V}$

these predefined path. The answer follows the same path in the opposite direction. To protect V's and other users' privacy expressed by the links in the Matryoshka, not even V (defined as the **core** of the Matryoshka) knows its entire composition. Apart from the list of the entrypoints which is publicly available, every node in the Matryoshka only knows the previous and next hops who by definition, are their friends, anyway. Every user in Safebook is assigned to a unique Matryoshka and can play different roles in different Matryoshkas. As previously mentioned, the list of the entrypoints is public and is stored by the second component of Safebook, the **P2P system**. By looking up for a hash value of a property of V, such as her full name, the P2P system provides the list of the entrypoints of V's Matryoshka. Figure2.2 depicts the data lookup process in Safebook.

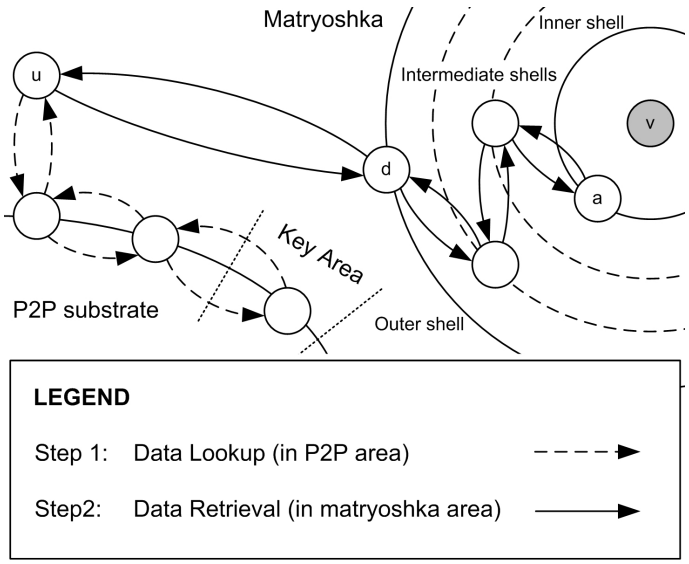

Figure 2.2: Data lookup in Safebook

2.2 Privacy analysis from the graph theory perspective

This section investigates the relationship between the topological properties of a social network graph and the achievable users' privacy in centralized or decentralized OSN. Social graph analysis

has already proved its importance for studies such as sociology [25] and network performance[26]. A similar analysis is driven with respect to the impact of social network topology on privacy. In [27], we show that there exists a strong relationship between a set of metrics and privacy properties. More specifically, three graph metrics, namely the node degree, the clustering coefficient and the mixing time, give fundamental insights on the privacy degree of the resulting OSN.

2.2.1 Graph theory vs. privacy

An Online Social Network can be represented as an undirected **social graph** $G(V, E)$ comprising a set V of users and a set E of edges representing social ties such as friendship. In this section we define the three metrics, namely, the ***node degree***, the ***clustering coefficient***, and the ***mixing time*** that will be analyzed with respect to privacy.

2.2.1.1 Node degree vs. user privacy

In graph theory, the degree of a vertex ν, denoted by $deg(\nu)$, is defined as the number of edges incident to the vertex. Since in a social graph $G(V, E)$, a vertex represents a user and the edges represent friendship links, a user's degree defines the number of friends a user has. Different studies have shown that OSN users are vulnerable to a series of social engineering attacks [28] often caused by accepting contact requests. When a user ν establishes a relationship with a new friend, with the increase of the degree, the probability of connecting to a misbehaving user increases. Assume p_{mal} denotes the probability a new friend η of ν is a malicious user, and assume the events of befriending a malicious user are independent. The number of malicious friends $F_{mal}(\nu)$ of ν then follows a binomial distribution:

$$\mathcal{F}_{mal}(\nu) \sim B(p_{mal}, deg(\nu))$$

In particular, the probability p_ν of having at least one misbehaving friend is:

$$p_\nu = 1 - p_{mal}^{deg(\nu)} \tag{2.1}$$

The more a node has friends, the higher the probability of having a malicious friend which can disclose sensitive personal data. Furthermore, if the correct execution of OSN applications such as in [24] depends on the structure of the social graph, the probability of discovering nodes' friends increases with the malicious behavior as well.

2.2.1.2 Clustering coefficient vs. user privacy

In an undirected graph, the clustering coefficient $c(\nu)$ of a node ν having $deg(\nu)$ edges is defined as the number of existing links between these nodes, denoted as $e_{deg(\nu)}$, divided by the number of possible links that could exist ($\frac{deg(\nu)(deg(\nu)-1)}{2}$). We therefore have:

$$c(\nu) = \frac{2e_{deg(\nu)}}{deg(\nu)(deg(\nu)-1)} \tag{2.2}$$

Knowing or estimating the clustering coefficient of a graph can give an idea on the impact of a malicious friendship whenever it has information on nodes friendship and can further disclose it. We can evaluate in a first approximation the average ratio $\mathcal{Q}_\nu$ of ν's friends that can obtain sensitive information disclosed by a malicious friend η as follows:

$$\mathcal{Q}_\nu = p_\nu c(\nu) \tag{2.3}$$

Therefore, similarly to the degree, the clustering coefficient has a direct effect on the disclosure of users' data. The tighter the friendset, the broader the disclosure of sensitive data.

2.2.1.3 Mixing time vs. profile integrity

Random walks [29] in a graph have an important property: when the random walk approximates its steady state distribution after a sufficient number of hops, the startpoint and endpoint of the walk are uncorrelated. This number of hops is called **mixing time**, and the smaller it is, the faster the above mentioned property is met.
We will introduce the mixing time starting from the steady state distribution.
The **steady state distribution** for a node θ represents the probability this random walk reaches θ after a sufficient number of hops, and does not depend on the node where this random walk originated from:

$$ssd\left(\theta\right) = \frac{deg\left(\theta\right)}{2\|E\|} \tag{2.4}$$

The mixing time [29] $\tau_x\left(\epsilon\right)$ is then computed as follows:

$$\tau_x\left(\epsilon\right) = min\left\{h : \Delta_x\left(h\right) \leq \epsilon\right\} \tag{2.5}$$

where $\Delta_x\left(h\right)$ is the variation distance between the random walk distribution $R^h\left(x\right)$ after h hops, and the steady state distribution $ssd\left(x\right)$:

$$\Delta_x\left(h\right) = \|R^h - ssd\| = \frac{1}{2}\sum_{x \in V} \|R^h\left(x\right) - ssd\left(x\right)\| \tag{2.6}$$

In [30], authors found that mixing time is much higher in social networks where links represent face-to-face interactions. The mixing time of a social network graph is directly related with both profile integrity and communication untraceability as some OSNs rely on random walks to increase users' privacy and security. Indeed OSNs with a small mixing time will be considered as more secure.

2.2.2 Conclusion: Safebook's feasibility

Peer-to-peer based OSN applications implemented over social networks with high clustering coefficient and slow mixing time unfortunately show a lower privacy degree with respect to fast mixing networks without strong local clustering. Additionally, the highest privacy degree that can be reached given a social network is the one where $h = h_\infty$: increasing h after this optimal value does not have an impact on the privacy level anymore. Privacy preserving OSN architectures, including Safebook, should address this problem by discouraging the indiscriminate action of adding friends. Moreover, the OSN should guarantee the fast mixing property to the network. This can be done by ensuring the small world property of the social network graph, and encouraging 'long links' connecting different clusters together, otherwise most of the random walks would be confined to the originating cluster.

2.3 Usage control for privacy preserving picture sharing

In this section, we overview a usage control solution dedicated to picture sharing applications within Safebook [31]. We consider a content management scenario where users would like to efficiently store and share their content while still having the entire control on the access or the usage of the corresponding content. Even though a generic usage control solution fitting all possible settings seems infeasible, in a confined environment with a well defined set of subjects, resources and operations, usage control can be achieved. The impact of leaving the system to violate some of the

rules would be negligible. Following this specific approach, we propose a usage control enforcement mechanism where the confined environment is defined Safebook. The proposed mechanism targets a specific content management application which is picture sharing. Picture sharing tools in online social networks allow users to upload any picture. Whenever a user uploads a picture, she defines the corresponding access rules and optionally tags her friends in the picture. We assume that each person whose face appears in any picture should decide whether her face in that picture should be disclosed or not. The proposed usage control mechanism is enforced thanks to the cooperation among multiple social network users: the idea of the proposed mechanism is to exploit the distributed nature of Safebook and to leverage real-life social links to control the access to pictures. The enforcement on the control of the pictures is assured thanks to a dedicated multi-hop data forwarding protocol. Before reaching its final destination, content has to follow a dedicated path of a sufficient number of intermediate nodes which automatically obfuscate the picture and follow the rules defined in the corresponding usage control policy. The length of this path depends on the ratio of malicious nodes in the system: we assume that the cooperation of at least t nodes would guarantee the correct execution of any operation. Thanks to the underlying privacy preserving multi-hop forwarding protocol, cleartext pictures will be accessible to authorized users only.

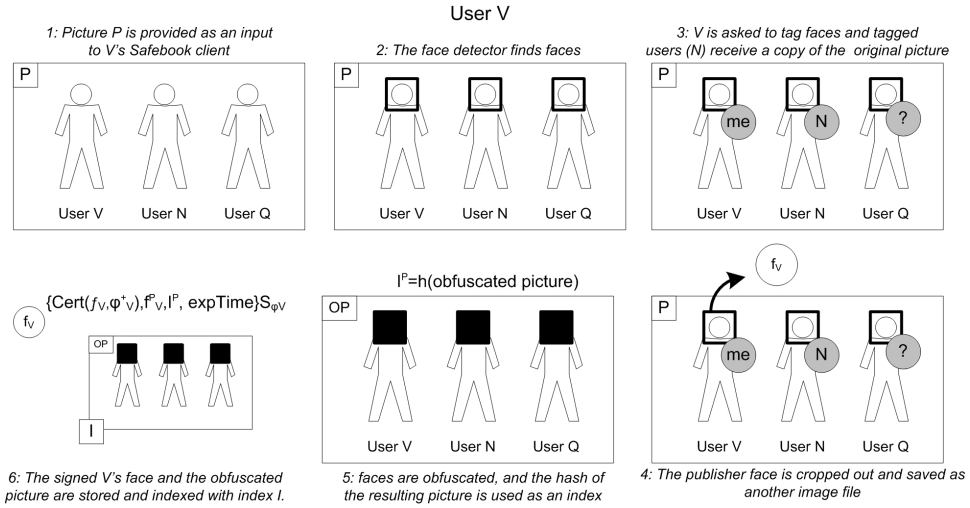

Figure 2.3: Picture publication steps for $\mathcal{V}$, with $\mathcal{V}$'s face $f_{\mathcal{V}}$ made publicly available: 1- picture input; 2- face detection; 3- face tagging; 4- face extraction; 5- face obfuscation; 6- picture and publisher face publication.

2.4 Conclusion

Thanks to this collaborative multi-hop enforcement scheme where it is assumed that there is at least one legitimate node which will execute the underlying algorithms correctly, an unauthorized user will not discover protected faces in the picture. [31] analyzes the impact of collusion against this disclosure. This analysis shows that the protection of the picture and the enforcement of this control is only efficient in the confined environment of the distributed OSN and when pictures are not encrypted by malicious nodes; on the other hand, it is also shown that the impact of attacks launched outside this environment remains very limited if the majority of the users are within the confined environment, the distributed OSN.

Chapter 3

Data privacy in Cloud Computing

The cloud computing paradigm offers clients the ease of outsourcing the storage of their massive data with the advantage of reducing cost. The advantages of cloud computing unfortunately come with a high cost in terms of new security and privacy exposures. In this chapter, we mainly focus on the problem of data privacy while considering the cloud server itself as being potentially malicious. We consider a scenario whereby a client outsources her privacy sensitive data to the cloud and delegates some computations over the data. As the first privacy requirement is data encryption, the client should encrypt her data before the outsourcing operation. As traditional encryption solutions fall short in addressing the requirement of processing over encrypted data, we focus on the widely used **word search** primitive and describe three newly proposed privacy preserving word search primitives designed under different settings. While the first solution is one of the early privacy preserving word search solutions adapted to the MapReduce technology, the second one allows the customer to delegate the search capabilities to authorized parties. Finally the recently proposed third solution is targeting the multi-user setting.

3.1 Introduction

The cloud computing paradigm allows businesses and individuals to migrate their data and computations to the cloud to reduce their costs. Clients almost get unlimited capacity of storage with high availability and reliability guarantees. The process of backup and recovery is relatively much easier. Customers can quickly scale up or scale down their usage of services on the cloud as per market demands. They only pay for the applications and data storage they need. The cloud provides users three different service models:

- Infrastructure as a Service (IaaS) clouds provide access to collections of virtualized computer hardware resources including machines, network and storage;
- Platform as a Service (PaaS) clouds provide access to a programming environment in which users develop and execute their own applications (Microsoft Azure[1], Google App Engine[2]);
- Software as a Service (SaaS) clouds provide software application programs.

The Cloud Security Alliance [3] provided a report in 2013 on the top 9 threats against cloud computing [4] to inform cloud users and providers about the main risks. Among these threats, data breach

[1] https://azure.microsoft.com/
[2] https://cloud.google.com/appengine/
[3] https://cloudsecurityalliance.org/
[4] http://www.cloudsecurityalliance.org/topthreats/

where unauthorized parties have access to privacy sensitive data, is considered to be the top threat. We therefore focus on this problem and assume the cloud being an **honest-but-curious** adversary who runs the protocol or operations correctly but try to extract as much information as possible about outsourced data or computations. The goal is therefore to come up with privacy preserving solutions that scale with large amounts of data, take advantage of the massive parallelism offered by cloud servers and do not request computationally intensive operations at the client side. As previously mentioned, we focus on a scenario where a data owner outsources its large data to the cloud and to preserve her privacy, she does not want any unauthorized party, including the cloud, to discover the content of her data. In order not to cancel out the advantages of the cloud computing technology, the challenge is to design a word search primitive over encrypted data without revealing any additional information to unauthorized parties and without the need for the data owner to download the entire data. We address the problem in an incremental way with respect to the roles of the data owner and the data querier. In section 3.3, we describe an early privacy preserving word search solution, PRISM, [32] that enables a data owner to lookup some words without letting the cloud discover the data and the queries. The main novelty of the proposed solution consists in its compatibility with the Map Reduce paradigm and hence its efficiency. PRISM takes advantage of the inherent parallelization akin to cloud computing and partitions the word search problem into several parallel instances in small datasets. In section 3.4, we further present a second solution described in [33] where in addition to the data owner, some authorized parties can query the database under the same adversarial model. In addition to the main security and privacy operations that classical search solutions include under a semi-honest (ie., honest-but-curious) security model, such a privacy preserving delegated word search solution also defines the delegation and revocation operations: the data owner should be able to efficiently authorize and revoke the previously authorized party at any point in time. Finally, in section 3.5 we consider the most general scenario where multiple users both store and query each others' data following their authorizations. The newly proposed solution [34] introduces a third party, called a proxy, that transforms a user's query into multiple ones each of them targeting one different data owner's document. We show that neither the proxy nor the cloud are trusted and the only assumption is that these two parties cannot collude.

3.2 Privacy preserving word search

A privacy preserving word search involves the following parties:

- **Data Owner** O_i who outsources a file F_i with n_i words to cloud server S in a secure way.
- **Cloud Server** S which stores an encrypted version of outsourced files $\{F_i\}_{i\in\{1,..,n\}}$ where n is the number of users in the system. We assume that each user outsources one file only.
- **Querier** Q_j who performs word search on some outsourced files if authorized. We note that Q_j corresponds to O_i if $j = i$ and hence O_i wants to query her own data.

3.2.1 Main algorithms

A basic privacy preserving (key)word search solution consists of three phases:

- Upload phase:
  During this phase, data owner O_i first prepares F_i with respect to the privacy requirements (eg. authorized parties to search for words) and further uploads the encrypted file together

with additional material to S. The uploaded data should not disclose any information on the data. The main algorithms defined under this phase are the following:

  - Setup: this algorithm is run by data owner O_i to generate a master key MK which will be used to mainly generate the data encryption key, some token to authorized queriers and a set of public parameters $param$ that will be used by subsequent algorithms.
  - Encrypt: this algorithm takes as input master key MK_i and file F_i and generates for O_i an encryption F_i' of F_i.
  - BuildIndex (optional): if needed, on input of MK_i and F_i, O_i runs this algorithm in order to construct an index I_i of distinct words present in file F_i.

- Lookup phase:
  During this phase, querier Q_j generates a query $q_{i,j,l}$ for word ω_l in file F_i and sends the query to S which processes it in an oblivious manner and sends the corresponding result $r_{i,j,l}$ back to the querier. Note that S may even not discover the actual response to the query (ie. whether ω_l is in F_i or not). The main algorithms defined under this phase are the following:

  - Query: given word ω_l and the appropriate keying material $QK_{i,j}$, querier Q_j generates a corresponding query $q_{i,j,l}$.
  - Response: this algorithm executed by cloud server S processes the query Q_j on file F_i or index I_i following the nature of the solution and returns a response $r_{i,j,l}$ for querier Q_j.

- Result analysis phase:
  Once $r_{i,j,l}$ is received by Q_j, this latter can decide on the actual response. Querier Q_j runs the following algorithm:

  - Verify: on input of response $r_{i,j,l}$ and the keying material $QK_{i,j}$, this algorithm outputs 1 if ω_l is in F_i or 0 otherwise.

3.2.2 Privacy requirements

The main security requirements of a privacy preserving (key)word search solution are:

- storage privacy: given the stored encrypted files $\{F_i\}$, cloud server S should not discover the content of files $\{F_i\}$; additionally, S should neither be able to mount statistical attacks on the outsourced files (number of occurrences, similarity among files).

- query privacy: during the lookup phase, S should not derive any useful information from the queries $\{q_{i,j,l}\}$ received from different queriers and their corresponding responses $r_{i,j,l}$. Namely, S should not be able to either tell whether two queries target the same word or discover the response to a given query.

3.3 PRISM: Privacy preserving word search in MapReduce

In [32], we propose a solution in the single-user setting that allows a data owner O_i to perform word search queries $q_{i,i,l}$ over her own data outsourced to the cloud server S. The proposed word search operation builds on a Private Information Retrieval (PIR) technique [35] which is extended

in order to generate intermediate search results (that are still encrypted). The database is mapped into a binary matrix whereby each word corresponds to a unique position within the matrix and the search operation consists in the retrieval of the bit corresponding to the word. The cloud, assumed to be honest-but-curious, cannot learn any information about either the content it hosts or the search queries performed. This implies both the encryption of data by the owner before outsourcing and the oblivious processing of queries on encrypted data.

3.3.1 Preliminaries

Private information retrieval As previously mentioned, to ensure query privacy, PRISM uses **Private Information Retrieval** (PIR). PIR allows a user to retrieve data from a server's database without disclosing any information about this particular data. Traditional PIR mechanisms require the user to know the position of the data to be retrieved in the database and therefore are not suitable to privacy preserving word search protocols as the user does not even know whether the word exists or not. Therefore, the owner will first construct a bitmap using the words in the file and then retrieve the bit corresponding to the particular word. To retrieve a certain bit from a database a PIR protocol first represents the database by a (s, t) binary matrix $\mathcal{M}$ and the user would just query the position (x, y) of the bit to be retrieved. Then the protocol by definition consists of three algorithms:

- PIRQuery which on input of the position (x, y) of the to-be-retrieved bit, computes the relevant PIR query pir_q for the user;
- PIRResponse which is run by the server over the matrix and returns a PIR response pir_r to the user;
- PIRAnalysis which based on the server's response retrieves bit b at position (x, y).

3.3.2 PRISM description

Figure 3.1 illustrates the proposed PRISM solution. During the upload phase of PRISM, data owner O_i runs the Setup and Encrypt algorithms instantiated as follows:

- $Setup(\phi)$: the algorithm returns MK_i which consists of the encryption key K_i. The public parameters $param$ comprise a symmetric encryption algorithm $E : \{0,1\}^{\phi} \times \{0,1\}^* \rightarrow \{0,1\}^n$ such as AES [36] and a cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{n_H}$.
- $Encrypt(K_i, F_i)$: for each unique word $w_{i,l}$ contained in file F_i, O_i first initializes a corresponding counter $\gamma_{i,l} = 0$; O_i further starts to parse file F_i. For each word $w_{i,l}$, O_i increments $\gamma_{i,l}$ by 1 and encrypts the pair $(\omega_{i,l}, \gamma_{i,l})$ using E. Thanks to the use of the counter, the underlying encryption solution becomes stateful and thus S cannot discover whether a given word appears several times in the file.

Once the encrypted version of file F_i is outsourced to the cloud, the lookup phase will begin as follows:

- querier Q_i who actually is O_i will prepare query for word w_l by calling the Query with inputs K_i and w_l. This algorithm will first compute the position corresponding to this word by simply computing $E(w_l, 1)$ and computing the position (x_l, y_l) such that $x_l||y_l$ corresponds to the bitmap of $E(w_l, 1)$. Further Q_i computes q PIR queries $\{pir\_q_j\}$ using the PIRQuery algorithm on position (x_l, y_l).

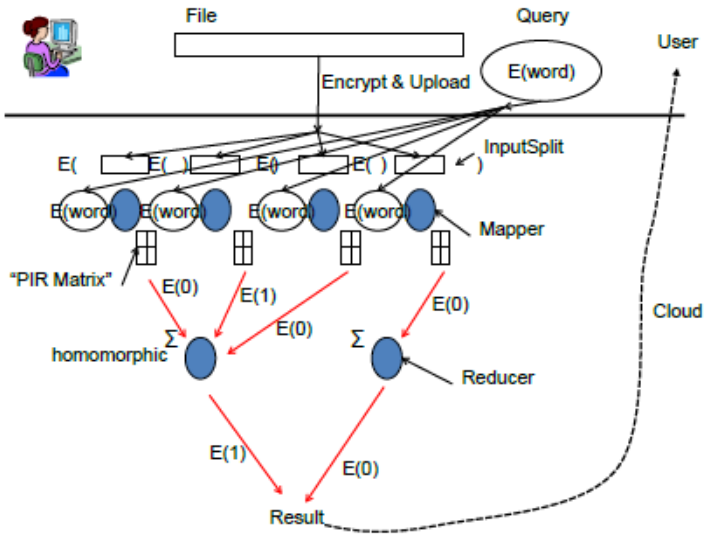

Figure 3.1: PRISM: Privacy preserving word search with Map Reduce

- upon reception of query $q_{i,l}$ cloud server S will first construct q matrices M_j initialized with "0"s in each cell; for each word C_k whose position is (x_k, y_k) in the matrix, $M_j(x_l, y_l)$ takes the value of b_j that corresponds to the jth bit of $H(C_k)$. S further launches PIRResponse to compute the responses to the pir queries at position (x_l, y_l) corresponding to the queried word w_l in all q matrices.

Finally, querier Q_i launches algorithm $Verify$ which basically calls PIRAnalysis q times for the q different bits. It further computes the first q bits of $H(C_l)$ and compares the results with response received from S.

3.3.3 Summary

Thanks to the use of the stateful cipher which consists in encrypting the word with an incremental counter, PRISM offers storage privacy where cloud server S can neither discover the content of file F_i nor the number of occurrences of a given word. Query privacy is inherently assured thanks to the use of PIR. Regarding the performance of the solution, PRISM easily parallelizes the $Response$ algorithm by splitting F_i into several *InputSplits* and executing $Response$ over each *InputSplit* in parallel. The solution is perfectly suitable to a MapReduce cloud where the **Map** phase consists of executing $Response$ to each *InputSplit* and during the **Reduce** phase, all intermediate responses received from each mapper are simply summed up: thanks to the homomorphic nature of the underlying PIR solution, the reduce phase does not have any impact on the correctness of the overall response. Details on the security proofs and experimental results can be found in [32].

3.4 Privacy preserving delegated word search

In this section, we extend the scenario tackled by PRISM and assume that querier Q_j can be a different party than data owner O_i and can search for some words only if authorized by O_i. In

addition to all the algorithms defined in PRISM, a delegated word search mechanism should also define Delegate and Revoke algorithms in order for data owner O_i to authorize a querier Q_j to perform the search operation and to remove this search capability at any point in time. Very few solutions [37, 38] propose the ability to delegate the search operation to authorized third parties; unfortunately, the delegation operation in all of these solutions is achieved by providing the data encryption key to the authorized parties and hence causing a re-encryption of the entire data in the case of revocation. Hence, the introduction of these two new algorithms, namely Delegate and Revoke, should neither impact the performance of the overall search operation nor reduce its security. In addition to the cloud, a revoked user should also be considered as potentially malicious and the new solution should ensure that she cannot have access to any additional information after her revocation. Similarly to PRISM, the newly proposed solution [33] also builds on private information retrieval (PIR) to guarantee query privacy. PIR is combined with Cuckoo hashing [39] which helps to efficiently construct a confidential searchable index where each word is assigned to a unique position. The delegation operation is assured thanks to the use of attribute based encryption (ABE) [13] which by definition only allows users holding certain "attributes" to search for words. On the other hand, the revocation operation combines ABE with oblivious pseudo-random functions [40, 41] which allow two parties to jointly compute the output of some pseudo-random function without discovering each other's input. With the use of OPRFs, the proposed protocol allows the cloud server to generate a one-time token for the authorized user without discovering the content of the query.

3.4.1 Preliminaries

Cuckoo hashing Cuckoo hashing [39] is a technique used to build efficient and practical data indexes. It ensures *worst-case* constant look-up and deletion time and *amortized* constant insertion time while minimizing the storage requirements. Cuckoo hashing is defined by the following two algorithms:

- CuckooInsert: in order to store n elements in some index $\mathcal{I}$, Cuckoo hashing uses two hash functions $H : \{0,1\}^* \rightarrow \{1,2,...,L\}$ and $H' : \{0,1\}^* \rightarrow \{1,2,...,L\}$. An element τ_i is either stored in entry $H(\tau_i)$ in hash table T, or in entry $H'(\tau_i)$ in hash table T' but never in both. To insert a new element $\tau_i \in \{0,1\}^*$ into $\mathcal{I}$, we first check whether the entry of T at position $H(\tau_i)$ is empty. If it is the case, then τ_i is inserted in this entry of T and the insertion algorithm converges. Otherwise, if that entry is already occupied by another element τ_j, then τ_j will be removed from its current entry in T and relocated to its other possible entry $H'(\tau_j)$ in T'. Now, if there is an element τ_k in the entry $H'(\tau_j)$ of T', then τ_j will be inserted in entry $H'(\tau_j)$ in table T' while τ_k will be moved to its other possible entry $H(\tau_k)$ in T. This insertion process is repeated iteratively until the insertion of all elements in either T or T'. If this process of insertion does not converge (i.e., there is an element that cannot be inserted), or it takes too long to converge, then all the elements in $\mathcal{I}$ will be rehashed with new hash functions $\mathbf{H}$ and $\mathbf{H}'$.

- CuckooLookup: the lookup operation in $\mathcal{I}$ is simple: given an element $\tau \in \{0,1\}^*$, the two entries at positions $H(\tau_i)$ and $H'(\tau_i)$ are queried in tables T and T' respectively.

Oblivious Pseudo-Random function An oblivious pseudo-random function (ORPF) [40, 41] is a two-party protocol that allows a sender S with input δ and a receiver R with input h to jointly compute the function $f_\delta(\mathsf{h})$ for some pseudo-random function family f_δ, in such a way that receiver R only learns the value $f_\delta(\mathsf{h})$, whereas sender S learns nothing from the protocol

interaction. In the following, we provide a quick overview of the generic algorithms underpinning an OPRF that evaluates the output of f_δ:

- $\mathsf{Setup}_{\mathsf{oprf}}$: executed by sender S, this algorithm takes as input the security parameter ζ and outputs an OPRF secret key δ and a set of public parameters $\mathcal{P}_{\mathsf{oprf}}$ that will be used by subsequent algorithms.
- $\mathsf{Query}_{\mathsf{oprf}}$: this randomized algorithm that is called by receiver R takes as input an element $\mathsf{h} \in \{0,1\}^\kappa$ and outputs a matching OPRF query $\mathscr{Q}_{\mathsf{oprf}}$ that will be sent later to sender S.
- $\mathsf{Response}_{\mathsf{oprf}}$: pperated by sender S, given OPRF query $\mathscr{Q}_{\mathsf{oprf}}$, this algorithm returns the corresponding OPRF response $\mathscr{R}_{\mathsf{oprf}}$ that will be forwarded to the receiver.
- $\mathsf{Result}_{\mathsf{oprf}}$: this final deterministic algorithm that is run by receiver R outputs $f_\delta(\mathsf{h})$ based on the received response $\mathscr{R}_{\mathsf{oprf}}$ for $\mathsf{h} \in \{0,1\}^\kappa$.

3.4.2 Protocol Description

We consider a scenario where data owner O_i outsources F_i which contains n distinct words. Cloud server S stores an encrypted version of F_i together with a searchable index $\mathcal{I}$ of the set of distinct words present in F_i. An authorized querier Q_j has access to a set of credentials that enables her to perform search queries on F_i. This authorized user could for example be an auditor which as part of its auditing task has to search the activity logs of $\mathcal{O}_i$.

Upload phase During the upload phase, data owner O_i sequentially executes the following algorithms:

- Setup: given security parameter ζ, the algorithm outputs the master key MK which regroups a symmetric encryption key K_{enc}, a MAC key K_{mac} and an OPRF secret key δ; Setup also returns public parameters that comprise a MAC function $\mathscr{H}_{\mathsf{mac}} : \{0,1\}^\zeta \times \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ (where N is a safe RSA modulus), a cryptographic hash function $\mathscr{H} : \{0,1\}^* \rightarrow \{0,1\}^t$ and the public parameters $\mathcal{P}_{\mathsf{oprf}}$ of the OPRF $f_\delta(h) = \mathfrak{g}^{1/(\delta+h)}$.
- $\mathsf{BuildIndex}_{\mathrm{O}}$: this algorithm computes a MAC h_k of each word ω_k in F_i with identifier fid_i such that $h_k = \mathscr{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega_k || \mathsf{fid_i})$. This is the very first step to compute the searchable index. The next steps are outsourced to the cloud which will call $\mathsf{BuildIndex}_{\mathrm{S}}$.
- Encrypt: on input of file F_i and encryption key K_{enc}, this algorithm outputs a simple symmetric encryption of F_i.
- Delegate: $\mathcal{O}_i$ first defines the access policy AP that will be associated with file F_i. To delegate the word search capabilities on the encrypted file F to authorized querier Q_j, $\mathcal{O}_i$ encrypts its MAC key K_{mac} under the access policy $\mathsf{AP_i}$ using ABE. This algorithm therefore returns the resulting ciphertext $\mathscr{C}_{\mathsf{mac}} = \mathsf{Enc}_{\mathsf{abe}}(K_{\mathsf{mac}}, \mathsf{AP_i})$. We note that the authorized user $\mathcal{Q}_j$ will possess a set of attributes $\mathcal{A}$ (and therewith a set of credentials cred) that satisfy the access policy $\mathsf{AP_i}$. Hence, $\mathcal{Q}_j$ will be able to derive the MAC key K_{mac} which will be used as an input to the furtherly described Query algorithm.

The outputs of $\mathsf{BuildIndex}_{\mathrm{O}}$, Encrypt and Delegate together with public parameters $param$, the access policy AP_i and a secret OPRF key δ are forwarded (via a secure channel) to cloud server $\mathcal{S}$. This secret key is used to compute some tokens for the dedicated user. The computation of this

token is performed using an oblivious pseudo-random function (OPRF) [41] which allows $\mathcal{U}$ and $\mathcal{S}$ to jointly compute the output of some pseudo-random function without discovering each others' input. Since OPRF is deemed to be demanding, $\mathcal{O}$ requests the cloud server to perform the more computationally intensive operations (i.e. OPRF and Cuckoo Hashing) . Before starting the search phase, cloud server $\mathcal{S}$ finalizes the construction of the searchable index using the $\mathsf{BuildIndex}_{\mathrm{S}}$ as follows:

- $\mathsf{BuildIndex}_{\mathrm{S}}$: this algorithm first calls the OPRF function on the received MAC values $\{h_k\}$ using secret key δ and outputs $\{\tau_k\}$. $\mathcal{S}$ further builds an index with the resulting values using Cuckoo hashing. $\mathsf{BuildIndex}_{\mathrm{S}}$ outputs the searchable index $\mathcal{I}_i = \{H, H', \mathbb{M}, \mathbb{M}'\}$ where H and H' are the two hash functions used by the Cuckoo hashing algorithm and $\mathbb{M} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_t\}$ and $\mathbb{M}' = \{\mathcal{M}'_1, \mathcal{M}'_2, ..., \mathcal{M}'_t\}$ correspond to the $2t$ matrices as described in the previous section.

OPRF phase Before starting the actual search phase, an authorized user Q_j first executes an OPRF protocol with cloud server $\mathcal{S}$. To search file F_i for some word ω_l, the authorized user $\mathcal{Q}_j$ first needs to generate the proper token to construct the word search query. The token generation consists of executing an OPRF protocol between the authorized user $\mathcal{Q}_j$ and the cloud server $\mathcal{S}$. On inputs of the word ω_l, the file identifier $\mathsf{fid_i}$ and the MAC key K_{mac}, $\mathcal{Q}_j$ first computes an OPRF query $\mathscr{Q}_{\mathsf{oprf}}$ to evaluate $f_\delta(\mathsf{h}) = \mathfrak{g}^{1/(\delta+\mathsf{h})}$ and forwards it to cloud server $\mathcal{S}$. Upon receipt of $\mathscr{Q}_{\mathsf{oprf}}$, $\mathcal{S}$ calls the OPRF algorithm $\mathsf{Response}_{\mathsf{oprf}}$. This algorithm uses the secret OPRF key δ and the OPRF query $\mathscr{Q}_{\mathsf{oprf}}$ to output an OPRF response $\mathscr{R}_{\mathsf{oprf}}$. $\mathcal{Q}_j$ further computes the OPRF response $\mathscr{R}_{\mathsf{oprf}}$ by decrypting the received ciphertext and deriving the word search token $\tau_{i,j,l}$ using the OPRF algorithm.

Search phase After obtaining the token $\tau_{i,j,l}$ corresponding to the word ω_l, $\mathcal{Q}_j$ runs the algorithm Query as follows:

- Query: to search file F_i for some word ω_l given $\tau_{i,j,l}$, this algorithm first computes $H(\tau) = (x, y)$ and $H'(\tau) = (x', y')$. Then, it computes two PIR queries (pir_q, pir_q') to retrieve the x^{th} and the x'^{th} row of a (s, t) binary matrix.

Cloud Server $\mathcal{S}$ proceeds by executing the following Response algorithm:

- Response: this algorithm runs PIRResponse over the received two queries (pir_q, pir_q') on the searchable index.

To verify whether ω_l is in the file F_i, the authorized user $\mathcal{Q}_j$ runs the algorithm PIRAnalysis and accordingly checks whether $\vec{b} = \mathscr{H}(\tau)$ or $\vec{b'} = \mathscr{H}(\tau)$. If it is the case, then Verify outputs 1 meaning that $\omega_l \in F_i$; otherwise, Verify outputs 0.

Revocation phase For the sake of simplicity, we assume that the revocation is attribute-based. To revoke an attribute att_j, $\mathcal{O}_i$ runs the algorithm Revoke as follows:

- Revoke: on input of attribute att_j this algorithm simply outputs a new access policy AP' that will further be sent to the cloud server $\mathcal{S}$. For instance, if we assume that the initial access policy AP of $\mathcal{O}_i$ states that auditors from EU and the US can perform word search on $\mathcal{O}_i$'s files, then a revocation of attribute US will lead to a new access policy AP' that says that only auditors from the EU can perform word search. In this manner, auditors from the US will no longer have access to $\mathcal{O}_i$'s file.

3.4.3 Evaluation

In order to evaluate the performance of the proposed solution, all the underlying algorithms were implemented in python and ran on an Intel Core 2 Duo 2.80GHz with a RAM of 3.8GB under Ubuntu 14.04 LTS. Several cases with different index sizes were considered: the number of words per index varies from 1 to 800000. Each algorithm was executed 50 times for each scenario.

Figure 3.2 shows that the cost of the Setup phase both at the user and at the cloud side is linear with respect to the number of words and remains affordable for a lightweight user (uploading 200000 words approximately takes 3 seconds).

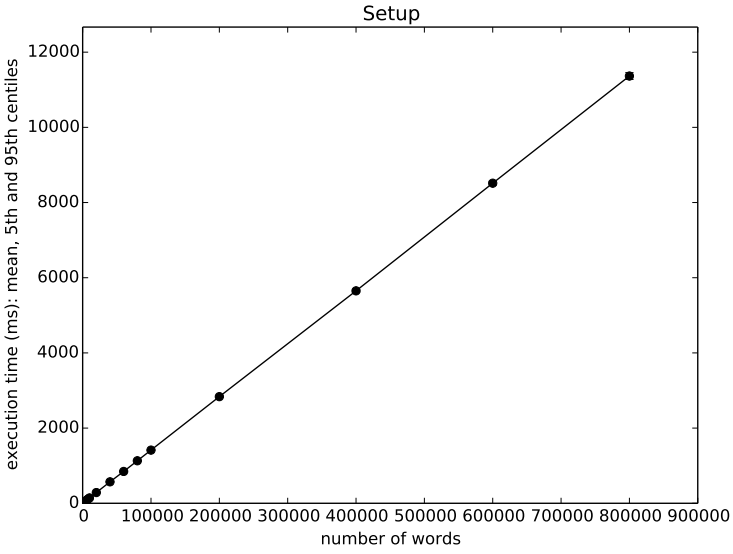

Figure 3.2: Cost of the setup phase at the user

The performance results shown in figure 3.3 consider the cost of the computation of the PIR query, and the extraction of the PIR response at the user side. These figures show that the computation of the PIR queries and responses at the user side depends on the size of the PIR matrix. Finally, the revocation phase is very efficient since it does not require the re-encryption of the outsourced files and only calls for an update of the access policy of the data owner at the cloud server.

3.4.4 Summary

Existing privacy preserving delegated keyword search solutions provide the authorized user with the data encryption key and therefore revocation of a user requires the re-encryption of the entirely outsourced data and the distribution of this new key to the authorized users. Thanks to the combination of ABE and OPRF, the revocation operation of the newly proposed solution does not imply the re-encryption of the outsourced data and only requires an update of the access policy by the data owner which can be considered as a negligible cost. Finally, the data owner in our protocol is only required to perform symmetric operations, whereas the computationally intensive computations are performed by the cloud server, and they can easily be parallelized.

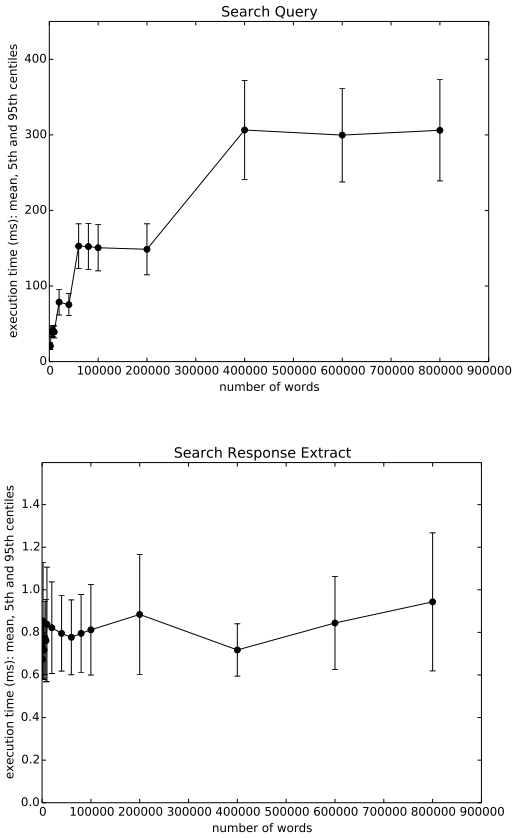

Figure 3.3: Cost of the search phase

3.5 Multi-user privacy preserving word search

As a follow-up to the previously described privacy preserving delegated word search soution, this section focuses on solutions that can efficiently adapt to any scenario with respect to the number of users and describes a dedicated solution proposed in [34]. In this multi-user setting, many different users, named as "writers", upload their data encrypted with their own secret key and many "readers" are allowed to search these different data. While such a scenario can easily be implemented by setting up a parallel instantiation of the previously described delegated word search solution for each data owner, this would incur a serious storage and computational cost to the querier who for example would search for the same word in many documents: the querier would need to store several keys (one per data owner) and generate one query per document. Hence the additional requirement that comes up with multi-user privacy preserving word search is scalability.

3.5.1 Overview

Similarly to the previously described solutions, this new solution [34] also builds up on PIR which is combined with the use of bilinear pairings. As illustrated in figure 3.4, in addition to all the algorithms defined for a delegated word search scheme, a multi-user searchable encryption (MUSE) solution defines two new algorithms executed by an additional actor, namely the proxy $\mathcal{P}$:

- QueryTransform: whenever $\mathcal{P}$ receives a query from Q_j, it transforms it to a number of

queries, each one targeting one data owner O_i through the execution of this algorithm.

- ResponseFilter: on reception of the corresponding responses, $\mathcal{P}$ runs this algorithm to pre-process them and forwards the output to the querier Q_j.

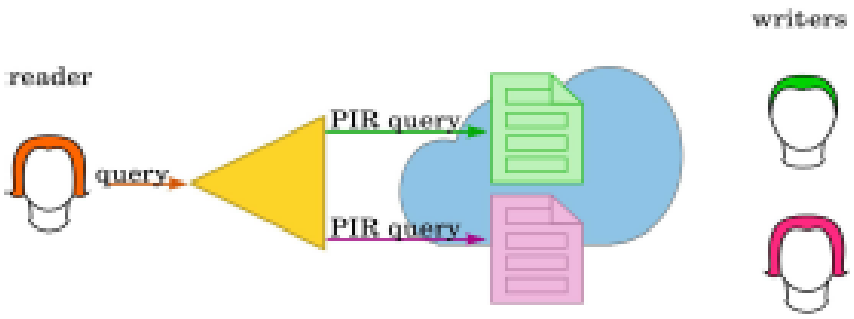

Figure 3.4: Multi-User Searchable Encryption - Overview

3.5.2 Building blocks

Bilinear pairings This new solution uses bilinear pairings as a building block for BuildIndex and Delegate algorithms. By definition, given that G_1, G_2 and G_T are three groups of prime order q and g_1, g_2 generators of G_1 and G_2 respectively, $e : G_1 \times G_2 \rightarrow G_T$ is a bilinear map if e is:

- efficiently computable
- non-degenerate: if x_1 generates G_1 and x_2 generates G_2, then $e(x_1, x_2)$ generates G_T
- bilinear: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}\ \forall (a, b) \in \mathbb{Z}^2$

3.5.3 Description

During the upload phase, a user $\mathcal{U}_i$ first runs algorithm Setup to generate the required keying material to become a "writer" (i.e. data owner, $\mathcal{O}_i$) on the one hand, and a "reader" (i.e. querier, $\mathcal{Q}_i$) on the other hand:

- Setup: similarly to the previously described delegated word search solution, given security parameter ζ, the algorithm outputs the master key MK and the public parameters $param$ defined as follows:
  - MK regroups a symmetric encryption key K_{enc}, a secret *writer key* $\gamma_i \xleftarrow{\$} \mathbb{Z}_q^*$, a private *reader key* $\rho_i \xleftarrow{\$} \mathbb{Z}_q^*$, a public reader key $P_i = g_2^{\frac{1}{\rho}}$, and a *transmission encryption key* K_i which is shared with cloud server $\mathcal{S}$.
  - public parameters $param$ comprise the public parameters of a bilinear map $e : G_1 \times G_2 \rightarrow G_T$ and a cryptographic hash function $\mathscr{H} : G_T \rightarrow [\![0, n-1]\!]$.

Then as a writer, data owner $\mathcal{O}_i$ executes the following Encrypt, Delegate and BuildIndex:

- Encrypt: on input of file F_i and encryption key K_{enc}, this algorithm outputs a simple symmetric encryption of F_i.

- Delegate: provided with the public key P_j of reader $\mathcal{Q}_j$, $\mathcal{O}_i$ executes this algorithm using its secret key γ_i to generate $\Delta_{i,j} = P_j^{\gamma_i}$ the authorization token that authorizes $\mathcal{Q}_j$ to search index I_i. The output $\Delta_{i,j}$ is sent to the proxy which adds it to the set D_j of authorized tokens corresponding to querier $\mathcal{Q}_j$.

- BuildIndex: $\mathcal{O}_i$ executes this algorithm to encrypt each keyword ω_k with her key γ_i. The algorithm outputs all $\tilde{w_k}$ such that: $\tilde{w_k} = e(h(w)^{\gamma_i}, g_2)$.

At the end of the upload phase, while $\mathcal{S}$ receives the encrypted version of F_i, the secure index I_i, $\mathcal{P}$ regroups all $\delta_{i,j}$ received from data owners in token sets D_j each of them corresponding to one querier $\mathcal{Q}_j$. During the search phase, Querier $\mathcal{Q}_j$, Proxy $\mathcal{P}$ and server $\mathcal{S}$ respectively run the following algorithms:

- Query: this algorithm is run by an authorized reader $\mathcal{Q}_j$ to generate a query for keyword ω_l using its private reader key ρ_j. The algorithm draws a randomization factor $\xi \xleftarrow{\$} \mathbb{Z}_q^*$ and outputs $q_j = h(\omega)^{\xi\rho_j}$.

- QueryTransform: whenever proxy $\mathcal{P}$ receives a reader's query q_j, it calls this algorithm together with the set D_j regrouping the authorization tokens for $\mathcal{Q}_j$. For each authorized document, the algorithm first computes $x'_{i,j}||y'_{i,j} =\leftarrow H(\tilde{q}_{i,j})$ where $\tilde{q}_{i,j} \leftarrow e(q_j, \Delta_{i,j})$; then it computes a PIR query for each position $(x'_{i,j}, y'_{i,j})$. The algorithm outputs $\{pir\_q'_{i,j}\}$ which are forwarded to the CSP together with the corresponding identifiers i of the indices.

- Response: on input of the PIR queries, the relevant documents identifiers i and the randomization factor x_i, cloud server $\mathcal{S}$ processes each PIR query $pir\_q'_{i,j}$ over a matrix M_i where, given encrypted word $\tilde{w} \in I_i$ mapped to position (x, y) such that $x||y = H(\tilde{w}^{\xi})$, $M(x, y)$ is set to "1" and the remaining cells are set to "0". These PIR responses $\tilde{r}_{i,j}$ are further encrypted with algorithm Enc and the transmission key K_j of the querying reader. This additional layer of encryption prevents the proxy from reading the result of the query.

Finally, before calling Verify, querier $\mathcal{Q}_j$ receives the response pre-processed by proxy $\mathcal{P}$ using algorithm ResponseFilter as follows:

- ResponseFilter: whenever the proxy receives the response from server $\mathcal{S}$ partly executes the PIRAnalysis function to extract a filtered response which is much smaller than the original response.

- Verify: on receiving the filtered response with the corresponding PIR keying material, $\mathcal{Q}_j$ executes this algorithm using her transmission key K_j. The algorithm further outputs the value of the matrix cell corresponding to the searched word. If this value is 1 then the algorithm outputs 1 and this means that the searched keyword is present, otherwise the algorithm outputs 0.

3.5.4 Evaluation

The proposed scheme achieves a very low cost at the reader side since QueryCreate only consists of one hash computation and one exponentiation and does not depend on the number of searched documents. On the other hand, the ResponseProcess executed over each received response (one

per searched document), only executes one decryption and a PIR of a single bit. Details on the cost of each algorihtm can be found in [34]. Regarding the security analysis, we realize that all existing multi-user searchable encryption schemes [42, 43, 44] suffer from the lack of access pattern privacy (privacy of search responses). We further come up with a new adversary model for MUSE that takes into account new security exposures introduced by the possible collusion of some users with the cloud server and prove the security of MUSE under this new model while considering both the cloud server and the proxy as an adversary. As opposed to existing solutions [42, 43, 44], the proxy is not trusted and the only assumption that the solution makes is that these two parties cannot collude.

3.6 Secure deduplication for cloud storage

With the potentially infinite storage space offered by cloud providers, cloud users tend to use as much space as they can. On the other hand, cloud providers constantly look for techniques that aim at minimizing redundant data and maximize space savings. A technique that is widely adopted is **cross-user deduplication** where duplicate blocks are stored only once. In this section we show that deduplication and encryption are two conflicting technologies and further provide an overview of a recently proposed secure deduplication solution.

3.6.1 Deduplication vs. Confidentiality

As illustrated in figure 3.5, cloud server only store one copy of similar blocks (the yellow and green boxes in this example). Recent experimental results show that deduplication achieve high cost savings. Unfortunately, deduplication and encryption are two conflicting technologies: the result

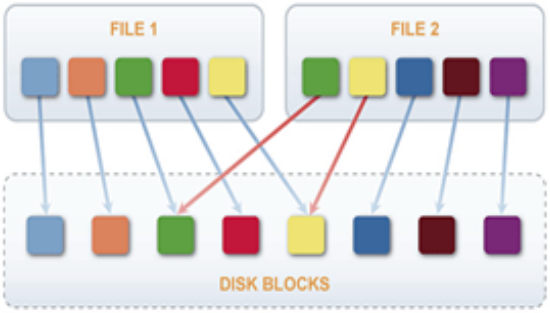

Figure 3.5: Cross-user deduplication

of encryption is to make two identical blocks indistinguishable after their encryption. Although an early proposed solution named **convergent encryption** [45] which derives the data encryption key from the data itself seems to be a promising candidate, it unfortunately suffers from various weaknesses including brute-force or dictionary attacks.

3.6.2 ClouDedup

We initially propose **ClouDedup** [46], depicted in figure 3.6 which while preserving the advantages of deduplication and convergent encryption at the same time, ensures data privacy thanks to its architecture whereby in addition to the basic storage provider, it involves a **metadata manager** and an additional server that implements an additional encryption layer to cope with the

weaknesses of convergent encryption together with a user authentication and an access control mechanism. Therefore ClouDedup involves four parties:

- the **cloud user**'s role is to simply split the file into blocks, to encrypt them with convergent encryption and to outsource the resulting encrypted file;
- the **cloud storage provider (CSP)** is the most simple component of the architecture as it only store the data blocks;
- the **gateway** receiving the user's storage request, first authenticates her and further adds another layer of encryption to the data uploaded by the user. During the phase of file retrieval, the server removes the additional layer of encryption before forwarding it to the intended user. These encryption and decryption operations are totally transparent to the cloud user;
- the **metadata manager(MM)** stores metadata which include some encrypted keying material and block signatures and is in charge of the deduplication operation. MM maintains a linked list and a small database in order to keep track of file ownerships and file composition.

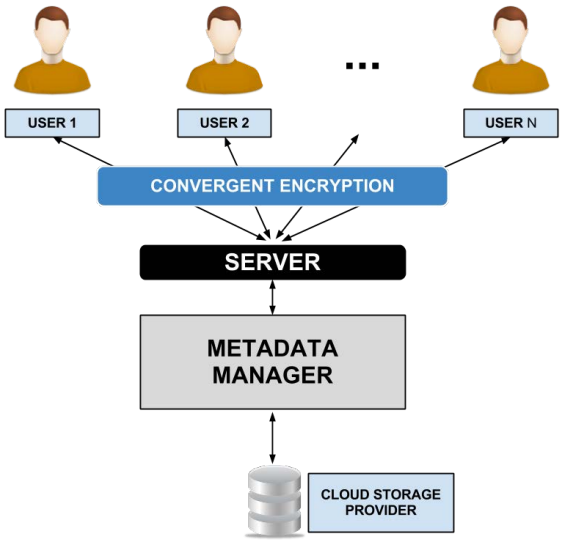

Figure 3.6: ClouDedup - Overview

Thanks to the additional encryption layer implementing a deterministic symmetric encryption algorithm, ClouDedup is secure against all attacks that were successful against convergent encryption (such as dictionary attacks). The system can be compromised only if the involved components collude. No component is completely trusted and the system is secure against each of them, individually. A more detailed analysis of the security of the proposed solution can be found in [46].

3.6.3 PerfectDedup

Although the previously described scheme seems a promising solution to ensure deduplication and data privacy at the same time, it relies on a complex architecture involving several different components. Hence, we propose a more simple solution that similarly to [47], leverages the popularity of the data segment to define the protection level of it. A data segment is defined to be popular whenever it is shared among more than a threshold number of users. We assume that a popular data

segment is likely to be non-confidential and subject to deduplication whereas unpopular (rare) data may contain privacy sensitive information. The proposed solution named PerfectDedup [48] protects popular data segments with convergent encryption and uses semantically secure encryption for unpopular ones. Nevertheless, this approach raises a new challenge: the users need to decide about the popularity of each data segment before storing it and the mechanism through which the decision is taken is exposed to attacks very similar to the ones against convergent encryption. Therefore, PerfectDedup defines a privacy preserving popularity detection mechanism that relies on the use of perfect hashing [49]. Thanks to this primitive, the user can decide which encryption solution to use to protect her data without revealing any information to the untrusted cloud server. Compared to [47], PerfectDedup significantly reduces the storage and communication overhead.

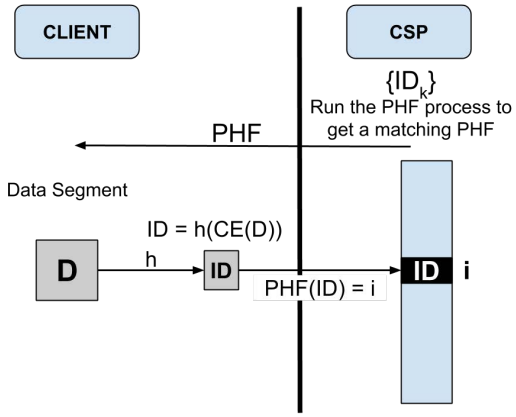

Figure 3.7: PerfectDedup - Overview

3.7 Conclusion and perspectives

This chapter studied the problem of data confidentiality and privacy whenever data is outsourced to the cloud and processed by this untrusted server. While the problem of privacy preserving word search data processing has received considerable attention from the research community, current solutions can still be extended with advanced features such as the search for multiple keywords. The combination of privacy preserving word search techniques with data reduction schemes can raise new challenges. Furthermore, recent advances in secure deduplication show that it is possible to come up with a secure solution without the need for convergent encryption. As a future work, it would be interesting to study the compatibility of such solutions with other privacy preserving or security primitives.

Chapter 4

Verifiable Cloud Computing

In addition to data privacy, cloud service providers need to provide more transparency. Indeed, in order not to loose the control over their data, cloud customers should have some means to verify the correctness of the remote operations/services. Such a requirement is defined as verifiability. In this chapter we first focus on the verifiability of the correct storage of customers' data (section 4.1) and overview a dedicated **proof of retrievability** solution. In section 4.2 we tackle the problem of verifiable computation and describe a recently proposed solution ensuring verifiability for a conjunctive keyword search primitive.

4.1 Verifiable data storage with proofs of retrievability

4.1.1 Introduction

Proofs of retrievability provide a client with the assurance that her previously outsourced data segments are actually present in the remote storage. The obvious solution for that would be to send the data back to the user but this obviously is not possible because of the large amount of data. Data retrievability is a new form of integrity requirement in that the data owner does not keep or get a copy of the data segment in order to get the assurance of its integrity. Early solutions [50, 51] allow clients to verify the possession of the entire data but this remains very costly; recent solutions assure that the server possesses parts (the majority) of the data. The verifier asks the server for integrity proofs for randomly chosen blocks. Some researchers [52, 53] propose the generation of a so-called tag to help the client to verify the possession of a specific block whereas some others [54, 55] secretly include some randomly generated blocks which are called sentinels and retrieve them later on, by revealing their position. In this section, we present a solution named StealthGuard [56] that combines the use of privacy preserving word search solution defined in section 3.3 with the random insertion of randomly generated blocks named watchdogs. StealthGuard is similar to the sentinel-based approach in that the client randomly inserts some watchdogs into the data; but StealthGuard sends a search query for this watchdog in a privacy preserving manner in order not to reveal the position of the watchdog to the server.

4.1.2 Problem statement

We consider a scenario whereby a data owner $\mathcal{O}$ wants to outsource a large file F to a cloud server $\mathcal{S}$. Later a verifier $\mathcal{V}$ interacts with $\mathcal{S}$ in order to check whether $\mathcal{S}$ is still storing F. This verifier can either be data owner $\mathcal{O}$ or any other party authorized by $\mathcal{O}$. A POR scheme should ensure the following security properties:

- if $\mathcal{S}$ correctly stores F, $\mathcal{V}$ should be able to efficiently verify this correct storage.
- $\mathcal{S}$ cannot convince $\mathcal{V}$ on the retrievability of F if not correctly stored; furthermore, $\mathcal{V}$ should rapidly detect such a malicious behavior.
- $\mathcal{V}$ should be able to check the retrievability of F at any time, and as many times as she wants.

With these security goals, a POR scheme is defined in three phases:

- Upload phase:
  During this phase, data owner $\mathcal{O}$ first generates some keying material to prepare F before its upload to the cloud. The main algorithms defined under this phase are the following:
  - Setup: given a security parameter ζ, this algorithm outputs a master key MK for data owner $\mathcal{O}$ and some public parameters $param$ that will be used by subsequent algorithms.
  - Encode: given MK generated by Setup and the file F to be outsourced, this algorithm returns a unique file identifier fid and an encoded file $\hat{F}$. Thanks to this new version of the file, verifier $\mathcal{V}$ will be able to launch the following POR phase.
- POR phase:
  During this phase, verifier $\mathcal{V}$ generates a POR query q_i for file F and sends the query to $\mathcal{S}$ who processes it thanks to the Prove algorithm and sends the corresponding result p_i back to $\mathcal{V}$. The main algorithms defined under this phase are the following:
  - Query: this algorithm takes as input the master secret key MK and the file identifier fid and returns a challenge q_i that will be sent to $\mathcal{S}$.
  - Prove: this response algorithm takes as input the previously generated query q_i and the file identifier fid and returns a proof p_i transmitted to verifier $\mathcal{V}$.
- Result analysis phase:
  At the end of the challenge-response protocol, verifier $\mathcal{V}$ runs the following algorithm to check the validity of the proof sent by $\mathcal{S}$:
  - Verify: Given master key MK, file identifier fid, query q_i and the proof p_i, this algorithm returns 1 if p_i is a valid proof for query q_i or 0 otherwise.

We note that if $\mathcal{V}$ receives at least γ correct responses from the cloud, then it can decide that F is retrievable with very high probability. On the other hand, if $\mathcal{V}$ receives one response that is not valid, then it is convinced that either the file is corrupted or even lost.

4.1.3 StealthGuard

4.1.3.1 Overview

In StealthGuard, before the upload of its data, the owner randomly inserts some watchdogs which are pseudo-randomly generated using a secret key. The random position at which the watchdog needs to be inserted is also computed using a secret key. By only keeping the secret key, the owner does not need to remember the watchdogs or their position for further verification. Further, in order not to let the cloud differentiate the random bogus data (the watchdogs) from the actual data, all data blocks are encrypted and after some additional operations such as error-correcting codes,

the data is uploaded to the cloud. The verifier further applies a privacy preserving word search operation like the one described in section 3.3 to check whether the watchdogs still exist at the given position. The cloud obliviously computes the responses on the existence of the requested watchdogs and send them back to the user. If the verifier obtains 0, then she detects a storage error and can ask for some remediation. Figure 4.1 depicts an overview of the different phases of Stealthguard. The next section provides the details of the solution.

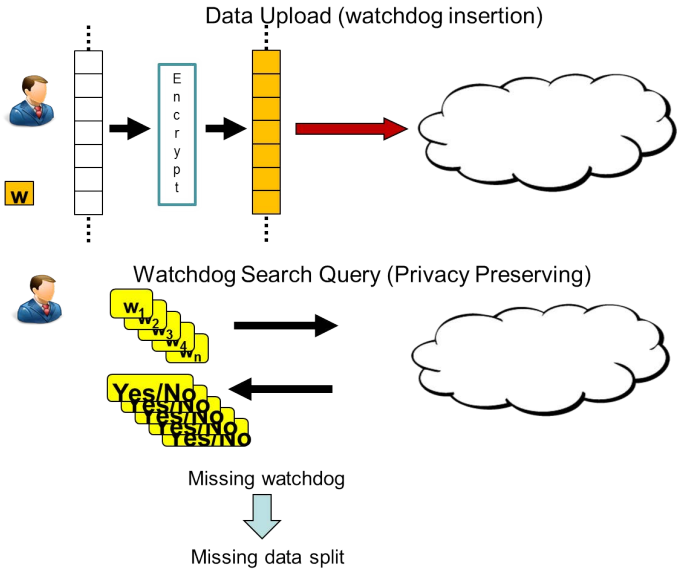

Figure 4.1: StealthGuard - Overview

4.1.3.2 Description

During the upload phase, data owner $\mathcal{O}$ first splits F into n splits S_j each of them comprising m blocks. Then $\mathcal{O}$ sequentially calls the following instantiated algorithms:

- $\mathsf{Setup}(\zeta)$: the algorithm returns $\mathsf{MK} = K$ which will be used to generated the keying material for the further Encode, Query and Verify algorithms. The algorithm also returns the following public parameters $param$: four cryptographic hash functions, namely H_{enc} used to derive a data encryption key; H_{wdog} used to generate pseudo-random watchdogs; H_{permF} and H_{permS} used to generated two permutation keys to be used for two permutation functions $\Pi_F : \{0,1\}^\zeta \times [\![1, n \cdot D]\!] \rightarrow [\![1, n \cdot D]\!]$ and $\Pi_S : \{0,1\}^\zeta \times [\![1, C]\!] \rightarrow [\![1, C]\!]$. $param$ also comprises a pseudo-random function $PRF : \{0,1\}^\zeta \times [\![1, n]\!] \times [\![1, v]\!] \times \{0,1\}^* \rightarrow \{0,1\}^l$.
- Encode: this algorithm is subdivided into the following steps:
  - Given master key K, $\mathcal{O}$ first generates $n+3$ keys that are: $K_{enc} = H_{enc}(K)$, $K_{wdog} = H_{wdog}(K)$, $K_{permF} = H_{permF}(K)$ and for all $i \in [\![1, n]\!]$, $K_{permS,i} = H_{permS}(K, i)$.
  - Each split S_i is expanded with $d-1$ blocks of redundancy thanks to the use of an $[m+d-1, m, d]$ error correcting code ECC. Thus, the new splits are made of $D = m + d - 1$ blocks. The use of ECC helps the owner to automatically correct small errors in the data.

– Each block of file F is further permuted using the permutation function Π_F. Each block is further encrypted with K_{enc} using a symmetric encryption Enc such as AES [36]. This step prevents an adversary to distinguish redundancy blocks from original ones.

– For each split, v l-bit watchdogs are generated using the pseudo-random function PRF. Since the watchdogs are pseudo-randomly generated and the blocks in the split are encrypted, $\mathcal{S}$ cannot distinguish watchdogs from data blocks. These v watchdogs are appended to each split.

– A split-level pseudo-random permutation is then applied to the blocks within the same split in order to randomize the location of the watchdogs:

– Encode finally outputs the resulting $\hat{F}$ and file identifier fid.

$\mathcal{O}$ stores master key K, publishes $param$ and forwards $\hat{F}$ and fid to cloud server $\mathcal{S}$. the setup phase is illustrated in figure 4.2.

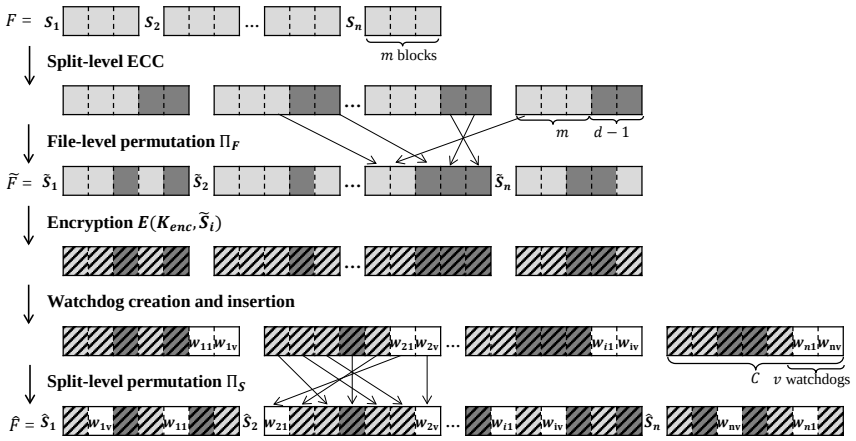

Figure 4.2: StealthGuard - Setup phase

The POR phase involving verifier $\mathcal{V}$ and cloud server $\mathcal{S}$ can now start with several interactive $Query$ and $Prove$ algorithms respectively executed by $\mathcal{V}$ and $\mathcal{S}$. Before the execution of these two algorithms, verifier $\mathcal{V}$ generates all the required keys that are derived from master key K:

- Query(K, fid): this algorithm randomly selects the split and the watchdog to be verified and computes the $wd_i = H_{wdog}(K, i)$. It further sends a word search query on w with a PIR based privacy preserving word search solution such as the one described in section 3.3.

- Prove(fid, q_i): on input of the word search query and the file identifier fid, $\mathcal{S}$ executes this algorithm which simply calls the word search solution's Response.

On reception of proof p_i, $\mathcal{V}$ executes the final Verify algorithm as follows:

- Verify: this algorithm calls the Verify algorithm of the privacy preserving word search solution with the inputs wd_i and p_i and outputs the result.

If Verify outputs 1 then $\mathcal{V}$ concludes that wd_i is found; otherwise, $\mathcal{V}$ detects a malicious behavior.

4.1.4 Summary and Discussion

Stealthguard is proved to be complete and sound. More details on the security and performance evaluation can be found in [56]. The paper shows that given a file F of 4GB, verifier $\mathcal{V}$ needs to send 1719 POR queries at least, to achieve a security degree of $1 - \frac{1}{2^{45}}$. These queries correspond to 0.64% of the size of the original file F. On the other hand the proposed solution can also be improved by considering the case whereby customers' data need frequent updates. Because of the complex Setup phase, StealthGuard cannot efficiently handle such updates; also, such an operation should not disclose any information regarding the watchdogs or their positions. While POR schemes mostly focus on cost optimizations at the customer side (be it data owner $\mathcal{O}$ or verifier $\mathcal{V}$), it would be interesting to combine such schemes with data reduction techniques such as data deduplication in order for them to additionally be cloud-friendly.

4.2 Verifiable keyword search

In this section, we summarize a verifiable word search mechanism proposed in [57], that helps a user searching in an outsourced database to verify the correctness of the lookup result.

4.2.1 Introduction

In addition to get assurance on the storage of the data, a cloud customer would like to have some guarantees on the integrity of the outsourced computations. Unfortunately, by moving her computing tasks to the cloud, the client inherently lends the control to this server which can be considered as being potentially malicious. Indeed, to free-up some of its computational resources, the cloud server can return bogus results. Verifiable computation aims at providing some tools to customers in order to let them gain the control on their data back and therefore force the cloud server to perform the requested operations correctly. While homomorphic signatures [58] could be considered as the perfect tool for verifiability, solutions targeting a specific computation primitive can sometimes be more efficient. More specifically, we focus on the keyword search primitive and design a solution that assures the correctness of the search result.

4.2.2 Problem Statement

We consider a scenario whereby a data owner $\mathcal{O}$ outsources a set of files $\mathcal{F}$ to the cloud server $\mathcal{S}$ and further allows any third party $\mathcal{V}$ to search for a number of words in an efficient manner. Together with the search result, that is the list of files containing the queried words, cloud server $\mathcal{S}$ also returns a proof p on the correctness of the search result. Similarly to all previously described protocols, data owner O should first prepare the to-be-uploaded data. Then, verifier $\mathcal{V}$ can start the search phase followed by the verification phase during which she verifies the correctness of the search result. Therefore a verifiable word search solution consists of five algorithms regrouped in three phases:

- Upload phase:
  - Setup: given a security parameter ζ and a set of files $\mathcal{F}$, this algorithm outputs a public key $PK_{\mathcal{F}}$, and a search key $LK_{\mathcal{F}}$ which regroups the words contained in all files received as an input in an index together with some additional information which will further be used to prove the integrity of the index.
- Search phase:

  - Query: given the public key resulting from Setup and the set of words, this algorithm constructs the query and the corresponding verification key. $\mathcal{V}_i$ sends this query to cloud server $\mathcal{S}$. The algorithm also returns a verification key VK_q in order to allow any verifier, even different from $\mathcal{V}_i$, to perform operations during the verification phase.
  - Search: given the search key $LK_{\mathcal{F}}$ and the encoded query received from verifier $\mathcal{V}_i$, server $\mathcal{S}$ executes this algorithm and computes the search result which consists of the list of files that contain the requested words and the corresponding proof.

- Verification phase:
  - Verify: This algorithm can be executed by any verifier $\mathcal{V}_j$ and takes as input the verification key VK_q,the resulting list of files $\mathcal{F}_q$ and the proof of search correctness p_q. If p_q is correct then, the algorithm outputs the same list of files; otherwise the output is $\perp$.

4.2.3 Requirements

In order for the verifiable computation solution to adapt to the more general case, the generic requirement of verifiability for outsourced computation can be split into two sub-requirements:

- **public delegatability**: any verifier $\mathcal{V}_i$ can issue search requests using Query without having access to the data owner's secret information;
- **public verifiability**: in addition to $\mathcal{V}_i$, any verifier $\mathcal{V}_j \neq \mathcal{V}_i$ can verify $\mathcal{S}$'s response.

These requirements also assure that a verifier can issue as many queries as needed. Furthermore, the major requirement of a verifiable computation solution is the efficiency of the operations performed at the client: Query and Verify should need much less computational resources than the Search algorithm. The proposed solution adopts the so-called *amortized model* where data owner $\mathcal{O}$ engages one-time expensive Setup and BuildIndex algorithms which will be amortized over an unlimited number of fast verifications.

4.2.4 Building blocks

Polynomial accumulators One way of achieving keyword search verifiability is by using cryptographic accumulators [59] which guarantee the membership of an element in a given set. Using polynomial accumulators, data owner $\mathcal{O}$ can define one polynomial for each file where the root of the polynomial is one word present in the file; to check whether a word ω is in all the files, server $\mathcal{S}$ would generate a proof of membership [60] or a proof of non-membership for each file. Hence, a verifiable test of membership is defined with the following three algorithms:

- Acc: on input of a set $S = \{h_1, .., h_n\}$, generates an accumulator $\mathsf{Acc}(S)$ which will be used by subsequent algorithms;
- GenerateWitness: on input of value h and a set S, computes the proof (which is generally called witness) of π_h (non-)membership of h with respect to S.
- VerifyMembership: this algorithm takes as input the actual queried value h, the accumulator $\mathsf{Acc}(S)$, and the output $\pi_h$of GenerateWitness, outputs h if $h \in S$ or halts otherwise.

Additionally, polynomial-based accumulators give way to verify set intersections [61] which will be helpful for efficiently searching over several files. A verifiable set of intersection consists of three algorithms:

- the previously defined accumulator function Acc;
- ProveIntersection: this algorithm takes as input k sets $S_1, .., S_k$ and returns their intersection I together with the proof Π_I of this result;
- VerifyIntersection: with the outputs of ProveIntersection, namely I and Π_I, the accumulators of each set, $\{\mathsf{Acc}(S_i)\}_{1 \leq i \leq k}$ and $\mathsf{Acc}(I)$, this algorithm outputs $Accept$ if the intersection is correctly computed, or $Reject$ otherwise.

Cuckoo hashing Since polynomial based accumulators can be very costly for the problem of verifiable keyword search in the context of very large database, they are combined with the use of the Cuckoo hashing technique (see section 3.4.1) to initially build an efficient index of the keywords and hence perform an efficient search and further verify the integrity.

Merkle trees To improve the performance of the solution even more, the index of keywords is authenticated thanks to the combination of polynomial accumulators with Merkle trees [62]. These binary trees of hashes are generally used to verify the integrity of a large file system in a logarithmic time. While leaf nodes represent a hash of an element, each internal node stores the hash of the concatenation of the values at each of its child nodes. The main algorithms used by the newly proposed solution are the following:

- BuildMT: this algorithm builds the Merkle tree T based on the elements of set S and a hash function H.
- GenerateMTproof: given a value h and the Merkle tree T, the algorithm outputs the authentication path $path$ which regroups all values of the sibling nodes on the path from the corresponding leaf node to the root node.
- VerifyMTproof: given an authentication $path$ for value h and the root value σ of tree T, the algorithm returns $Accept$ if h is correct and $Reject$ otherwise.

4.2.5 Verifiable conjunctive keyword search - Description

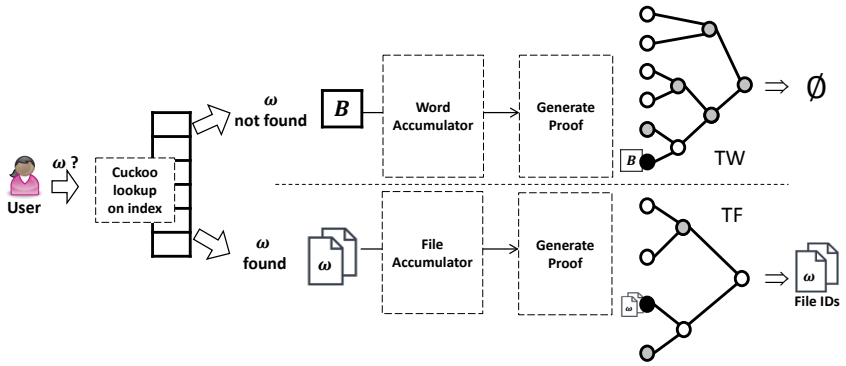

Figure 4.3: Verifiable conjunctive keyword search - Overview

In this section, we briefly describe the proposed solution also illustrated in figure 4.3. During the upload phase, data owner $\mathcal{O}$ builds an index of words together with two Merkle trees: one authenticating the index and the other authenticating the set of files containing a given word.

- Setup: after generating public parameters of the protocol, this algorithm constructs an index I of keywords present in the received set of files $\mathcal{F}$ using CuckooInsert. This index is further authenticated with the combination of accumulators and Merkle trees. For each bucket B_i in index I, data owner $\mathcal{O}$ computes an accumulator $\mathsf{Acc}(B_i)$ and builds a Merkle tree TW using BuildMT with all these accumulators. Additionally, the algorithm identifies the set of files $\mathcal{F}_{\omega_i}$ containing w_i and builds the corresponding accumulators $\mathsf{Acc}(\mathcal{F}_{\omega_i})$ in order to finally construct a second Merkle tree TF. The algorithm outputs the public key $PK_{\mathcal{F}}$ regrouping all public parameters and the search key $LK_{\mathcal{F}}$ which consists in the set of words in $\mathcal{F}$, the set of files $\mathcal{F}_{\omega_i}$ containing word ω_i, the resulting index I and the two Merkle trees TW and TF.

A verifier $\mathcal{V}_i$ generates a conjunctive query using Query which is further received by cloud server $\mathcal{S}$ who will process the query with Search as follows:

- Query: this algorithm simply returns the set of k words to be queried $E_q = \mathcal{W}$ and the public verification key which regroups the previously computed public key $PK_{\mathcal{F}}$ and the set of words $\mathcal{W}$ in the query.
- Search: given the query E_q and the lookup key $LK_{\mathcal{F}}$, this algorithm starts to lookup all the words in $\mathcal{W}$ using CuckooLookup.
  - If all words are found in index I, then for each word ω_i in the query, Search retrieves the set of files $\mathcal{F}_{\omega_i}$ containing this specific word and computes the intersection of these sets $\mathcal{F}_{\mathcal{W}}$ and the proof of this intersection $\Pi_{\mathcal{W}}$ using ProveIntersection. It also returns Merkle tree proofs for each accumulator of $\mathcal{F}_{w_i}$, computed with GenerateMTProof. Therefore, if all words exist in $\mathcal{F}$, Search returns:
  $$\mathcal{E}_R = (\mathcal{F}_{\mathcal{W}}, \Pi_{\mathcal{W}}, \{(\mathsf{Acc}(\mathcal{F}_{\omega_i}), path_{\omega_i})\}_{1 \leq i \leq k})$$
  - If for a given word w_j CuckooLookup returns false, then Search computes the corresponding proof of non-membership by first calling GenerateWitness with h_j and the two buckets $\mathcal{B}_{1j}$ and $\mathcal{B}_{2j}$ in which the word w_j is supposed to be (positions are computed with the two hash functions H_1 and H_2 used by CuckooInsert). In addition to Π_1 and Π_2, the algorithm also returns the proofs of membership for these two buckets $\mathcal{B}_{1j}$ and $\mathcal{B}_{2j}$ by computing their corresponding accumulators $\mathsf{Acc}(\mathcal{B}_{1j})$ and $\mathsf{Acc}(\mathcal{B}_{2j})$ and the Merkle Tree proofs $path_1$ and $path_2$ with respect to tree TW. Hence for the first word ω_j not found in $\mathcal{F}$, Search returns:
  $$\mathcal{E}_R = (\emptyset, \omega_j, \pi_1, \pi_2, \mathsf{Acc}(\mathcal{B}_{1j}), \mathsf{Acc}(\mathcal{B}_{2j}), path_{1j}, path_{1j})$$

Whenever $\mathcal{V}_j$ receives the response from server $\mathcal{S}$ it executes Verify with the verification key VK_{q_j} as follows:

- Verify: given the set of words in the query and $\mathcal{E}_R$, this algorithm proceeds as a follow-up to the two previously described cases:
  - if all words are found, ie. if $\mathcal{E}_R = (\mathcal{F}_{\mathcal{W}}, \Pi_{\mathcal{W}}, \{(\mathsf{Acc}(\mathcal{F}_{\omega_i}), path_{\omega_i})\}_{1 \leq i \leq k})$, the algorithm uses VerifyIntersection to verify $\Pi_{\mathcal{W}}$ and VerifyMTProof to verify $path_{\omega_i}$. If all verifications succeed, then Verify outputs $\mathcal{F}_{\mathcal{W}}$.
  - if, on the other hand, $\mathcal{E}_R = (\emptyset, \omega_j, \pi_1, \pi_2, \mathsf{Acc}(\mathcal{B}_{1j}), \mathsf{Acc}(\mathcal{B}_{2j}), path_{1j}, path_{1j})$, Verify calls VerifyMembership on π_1 and π_2 using $\mathsf{Acc}(\mathcal{B}_{1j})$ and $\mathsf{Acc}(\mathcal{B}_{2j})$ respectively; it also executes VerifyMTProof over $path_{1j}$ and $path_{2j}$. If all verifications succeed, then Verify outputs $\emptyset$.

4.2.6 Summary and Discussion

As opposed to most existing solutions [63, 64, 65, 66], the one we propose does not support verifiable keyword search on encrypted data but satisfies public delegatability and verifiability instead: indeed, any third party can perform search on the outsourced data and verify the correctness of the result. The proposed solution is efficient as the verification complexity is logarithmic in the size of the database. More details on the performance evaluation and the security analysis of the solution can be found in [67]. Similarly to POR schemes, it would be interesting to support frequent data updates in an efficient manner.

4.3 Conclusion

This chapter studied the problem of trust in cloud computing and reviewed some verifiability solutions dedicated to specific cloud storage and computation operations. The two solutions described in this chapter allow users to verify the correct storage of their data and the correctness of the outsourced search operation, respectively. Such schemes will help build and foster trust between cloud service providers and cloud customers. As a future work it would be interesting to reconcile PORs with data deduplication by devising new solutions that operate correctly while ensuring that cloud providers can still optimize their resource usage. On the other hand while the problem of verifiable computation still remains open, the possibility of combining it with the privacy of the data becomes even more challenging.

Conclusions and Perspectives

This thesis outlined various security and privacy issues raised by the emergence of new technologies. Unlike traditional communication protocols, opportunistic protocols use additional information to establish communication among parties: context-based and content-based communications rely on the exchange of sensitive information such as the location of the node or the profile of the forwarding nodes. Chapter 1 summarizes the main privacy preserving primitives dedicated to this new communication paradigm. The thesis further focuses on the problem of privacy in online social networks which are sometimes considered as the first communication channel among people. Chapter 2 analyzes the main privacy questions raised by distributed online social networks and further describes a usage control solution for picture sharing in such distributed networks. Chapters 3 and 4 tackle the different security and privacy challenges raised by the cloud computing technology which allows customers to reduce their maintenance costs by outsourcing their data to cloud servers. While chapter 3 overviews different privacy preserving word search primitives, chapter 4 reports the newly proposed verifiability solutions that help increasing the trust towards this new technology and convince customers on its adoption. Since these solutions only target specific operations such as word search, we aim at analysing other operations outsourced to the cloud and design dedicated privacy preserving and security primitives for them. The ultimate goal would be to come up with generic solutions that can be used for arbitrary computations.

With the rapid advance of technology and the explosion of the amount of (personal) data exchanges between devices and people, the need for security measures and the protection of personal data becomes mandatory. Indeed, smartphones are expected to remotely control various devices such as smart meters, TVs or cars. The EU has already reported on the vulnerability of the Internet of Things (IoT) and the lack of sufficient security measures [68]. On the other hand, cryptographic techniques become more practical, efficient and thus user-friendly. While recent advances on fully homomorphic encryption which allows the arbitraty computation over encrypted data are encouraging, the suitability of such emerging cryptographic techniques to this new ecosystem also remains an interesting and open question.

Bibliography

[1] Stefano Basagni and Marco Conti and Silvia Giordano and Ivan Stojmenovic. *Mobile Ad Hoc Networking*. Wiley-IEEE Press, 2nd edition, 2013.

[2] P. Michiardi and R. Molva. Simulation-based analysis of security exposures in mobile ad hoc networks. In *European Wireless Conference*, 2002.

[3] Melek Önen, Abdullatif Shikfa, and Refik Molva. Optimistic fair exchange for secure forwarding. In *Proceedings of the 1st Workshop on the Security and Privacy of Emerging Ubiquitous Communication Systems (SPEUCS)*, 2007.

[4] L. Buttyan and J.P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM Journal for Mobile Networks (MONET), special issue on Mobile Ad Hoc Networks*, 8(5), October 2003.

[5] S. Zhong, J. Chen, and Y.R. Yang. Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks. In *Proceedings of Infocom*, 2003.

[6] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of Hot-Potato Routing in IP networks. In *Proceedings of ACM SIGMETRICS*, June 2004.

[7] National Institute of Standards and Technology. Advanced Encryption Standard, 2001.

[8] N. Asokan, V. shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[9] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy in context-based and epidemic forwarding. In *Proceedings of the 13rd IEEE International WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC)*, 2009.

[10] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy and confidentiality in context-based and epidemic forwarding. *Computer Communications*, 2010.

[11] Chiara Boldrini, Marco Conti, Jacopo Jacopini, and Andrea Passarella. Hi-BOp: a History Based Routing Protocol for Opportunistic Networks. In *Proceedings of the 8th IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WoWMoM*, 2007.

[12] Hoang Anh Nguyen, Silvia Giordano, and Alessandro Puiatti. Probabilistic Routing Protocol for Intermittently Connected Mobile Ad hoc Network (PROPICMAN). In *Proceedings of the 8th IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WoWMoM*, 2007.

[13] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *13th ACM Conference on Computer and Communications Security (CCS)*, 2006.

[14] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in Cryptology - Asiacrypt'01*, volume LNCS 2139, pages 213–229. Springer-Verlag, 2001.

[15] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proceedings of Eurocrypt*, volume LNCS 3027, pages 506–522. Springer-Verlag, 2004.

[16] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[17] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332ï£¡383, 2001.

[18] Mudhakar Srivatsa and Ling Liu. Secure event dissemination in publish/subscribe networks. In *Proceedings of the 27th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[19] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy in content-based opportunistic networks. In *Proceedings of the 2nd IEEE International Workshop on Opportunistic Networking (WON)*, 2009.

[20] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy-preserving content-based publish/subscribe networks. In *Proceedings of the 24th IFIP International Information Security Conference (SEC)*, 2009.

[21] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSSï£¡02)*, 2002.

[22] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106ï£¡110, 1978.

[23] Abdullatif Shikfa, Melek Önen, and Refik Molva. Bootstrapping security associations in opportunistic networks. In *Proceedings of the 6th IEEE International Workshop on Mobile Peer-to-Peer Computing (MP2P)*, 2009.

[24] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: a privacy preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12), 2009.

[25] Stanley Milgram. The small world problem. *Psychology Today*, pages 60 – 67, May 1967.

[26] Alan Mislove, Massimilliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceeding of the 5ht ACM/USENIX Internet Measurement Conference (IMC'07)*, October 2007.

[27] L. A. Cutillo, R. Molva, and M. Önen. Analysis of privacy in online social networks from the graph theory perspective. In *Proceedings of IEEE Globecom*, 2011.

[28] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *WWW*, 2008.

[29] Michael Mitzenmacher and Eli Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, January 2005.

[30] Matteo Dell amico and Yves Roudier. A measurement of mixing time in social networks. In *5th International Workshop on Security and Trust Management STM, September 24-25, 2009, Saint Malo, France*, 2009.

[31] L. A. Cutillo, R. Molva, and M. Önen. Privacy preserving picture sharing: Enforcing usage control in distributed online social networks. In *5th ACM Workshop on Social Network Systems (SNS)*, 2012.

[32] E.-O. Blass, R. di Pietro, R. Molva, and M. Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. LNCS, July 2012.

[33] Kaoutar ELkhiyaoui, Melek Önen, and Refik Molva. Privacy preserving delegated word search in the cloud. In *Proceedings of 11th International conference on Security and Cryptography (SECRYPT)*, 2014.

[34] Cédric Van Rompay, Refik Molva, and Melek Önen. Multi-user searchable encryption in the cloud. In *18th International Conference on Information Security (ISC)*, 2015.

[35] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords, 1997.

[36] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[37] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 79–88. ACM, 2006.

[38] C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 127–143, Berlin, Heidelberg, 2008. Springer-Verlag.

[39] R. Pagh and F.F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[40] M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 303–324, Berlin, Heidelberg, 2005. Springer-Verlag.

[41] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 577–594. Springer Berlin Heidelberg, 2009.

[42] Raluca Ada Popa and Nickolai Zeldovich. Multi-Key Searchable Encryption, 2013.

[43] Yanjiang Yang, Haibing Lu, and Jian Weng. Multi-User Private Keyword Search for Cloud Computing. In *IEEE CLOUD*, pages 264–271. IEEE, November 2011.

[44] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In *Information Security Practice and Experience*, pages 71–85. Springer, 2008.

[45] John R. Douceur, Atul Adya, William J. Bolosky, P. Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22nd IEEE International Conference in Distributed Computing Systems*, 2002.

[46] P. Puzio, R. Molva, M. Önen, and S. Loureiro. ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage. In *PRoceeings of the , 5th IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM)*, 2013.

[47] S. Janek, A. Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In *Proceedings of the 18th International Conference on Financial Cryptography and Data Security*, 2014.

[48] P. Puzio, R. Molva, M. Önen, and S. Loureiro. PerfectDedup: Secure data deduplication. In *10th International Workshop on Data Privacy Management (DPM)*, 2015.

[49] D. Belazzougui, F. C. Botelho, and M. Dietzfelbinger. Hash, displace and compress. In *Proceedings of Algorithms-ESA*, 2009.

[50] Yves Deswarte and Jean-Jacques Quisquater. Remote integrity checking. In *Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, pages 1–11. Kluwer Academic Publishers, 1 2004.

[51] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. *IACR Cryptology ePrint Archive*, 2006:150, 2006.

[52] Shacham, Hovav and Waters, Brent. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.

[53] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12, 2011.

[54] Ari Juels and Burton S. Jr. Kaliski. PORs: proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[55] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: a scalable cloud file system with efficient integrity checks. In *Proceddings of the 28th Annual Computer Security Applications Conference (ACSAC)* , 2012.

[56] Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva, and Melek Önen. Stealthguard: Proofs of retrievability with hidden watchdogs. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*, 2014.

[57] M. Azraoui, K. ELkhiyaoui, M. Önen, and R. Molva. Publicly verifiable conjunctive keyword search in outsourced databases. In *Proceedings of the 1st IEEE Workshop on Security and Privacy in the Cloud*, 2015.

[58] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 469–477, New York, NY, USA, 2015. ACM.

[59] Lan Nguyen. Accumulators From Bilinear Pairings and Applications. In *Topics in Cryptology–CT-RSA 2005*, pages 275–292. Springer, 2005.

[60] Ivan Damgård and Nikos Triandopoulos. Supporting Non-Membership Proofs with Bilinear-Map Accumulators. *IACR Cryptology ePrint Archive*, 2008:538, 2008.

[61] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable Set Operations over Outsourced Databases. In *Public-Key Cryptography–PKC 2014*, pages 113–130. Springer, 2014.

[62] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Proceedings of Advances in Cryptology (CRYPTO)*, 1987.

[63] Qi Chai and Guang Gong. Verifiable Symmetric Searchable Encryption for semi-Honest-but-Curious Cloud Servers. In *IEEE International Conference on Communications (ICC), 2012*, pages 917–922. IEEE, 2012.

[64] Zachary A Kissel and Jie Wang. Verifiable Phrase Search over Encrypted Data Secure against a Semi-Honest-but-Curious Adversary. In *IEEE 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW), 2013*, pages 126–131. IEEE, 2013.

[65] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. VABKS: Verifiable Attribute-Based Keyword Search over Outsourced Encrypted Data. In *INFOCOM, 2014 Proceedings IEEE*, pages 522–530. IEEE, 2014.

[66] Rong Cheng, Jingbo Yan, Chaowen Guan, Fangguo Zhang, and Kui Ren. Verifiable Searchable Symmetric Encryption from Indistinguishability Obfuscation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '15, pages 621–626, New York, NY, USA, 2015. ACM.

[67] Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. Publicly Verifiable Conjunctive Keyword Search in Outsourced Databases. Technical report, Eurecom, 2015.

[68] Internet of Things, IoT Governance, Privacy and Security Issues, 2015.

Curriculum Vitae

Melek Önen

EURECOM
Campus SophiaTech
450 Route des Chappes
06410 Biot
France

Phone (Office): +33 4 93 00 82 09
Fax : +33 4 93 00 82 00
Email: melek.onen@eurecom.fr
Homepage: http://www.eurecom.fr/~onen/

Appointment

EURECOM, Sophia-Antipolis, France — October 2005 – present
Senior Researcher in the Digital Security department.

Education

EURECOM/TélécomParisTech, Sophia-Antipolis, France, — July 2005
Ph.D. in Computer Science, prepared at EURECOM
Thesis: *Securing Multicast communications in satelline networks: A user-centric approach*
First class honors: "mention très honorable" Advisor: Prof. Refik Molva

Université Pierre et Marie Curie (Paris 6), Paris, France — 2001
M.Sc. Degree in Distributed Systems

Galatasaray University, Istanbul, Turkey, — 2000
B.Sc. degree in Computer Science

Research projects

TREDISEC, H2020-ICT — (2015-2018)
Trust-aware **RE**liable and **D**istributed **I**nformation **SE**curity in the **C**loud
http://tredisec.eu/
Scientific Director and Work Package Leader, steering and management of the project.

CLARUS, H2020-ICT — (2015-2018)
A framework for user centred privacy and security in the cloud
http://clarussecure.eu/
Steering and management of the project.

UCN, IST STREP project — (2013-2016)
User **C**entric **N**etworking
http://usercentricnetworking.eu/
Work Package Leader, steering and management of the project.

A4CLOUD, IST IP project — (2012-2016)
The Cloud Accountability Project
http://a4cloud.eu/
Work Package Leader, steering and management of the project.

RECOGNITION, IST FP7 FET project (2010-2013)
Relevance and cognition for self-awareness in a content-centric Internet
`http://users.cs.cf.ac.uk/Recognition.Project/wordpress/`
Work Package Leader, steering and management of the project.

SOCIALNETS, IST FP7 FET project (2008-2011)
Social networking for pervasive adaptation
`http://www.social-nets.eu/`
Work Package Leader, steering and management of the project.

PROSE, French ANR poject (2008-2011)
Contenu Partagé par Recommandation Pair-à-Pair et Réseaux
Management of the project.

HAGGLE, IST FP6 IP project, (2006-2010)
An innovative Paradigm for Autonomic Opportunistic Communication
`http://www.haggleproject.org/`
Work Package leader, management of the project.

Advising experience

PhD students

Monir Azraoui 2012-2016
Title: Verifiability and Accountability in the Cloud
co-advised with Prof. Refik Molva.

Abdullatif Shikfa 2006-2010
Title: Security for Opportunistic Communications
co-advised with Prof. Refik Molva.

Interns

Duc Trinh 2016
Proofs of retrievability with secure deduplication

Cédric Van Rompay 2014
Privacy preserving delegated word search.

Funmipe-Victor Olofinlade 2013
Proofs of retrievability.

Anna Kozachenko 2012
P2P based usage control.

Wenting Li 2011
Modeling of Safebook.

Anais Rabinoit 2011
Privacy protection in P2P networks.

Waqas Liaqat Ali 2011
Secure data management in Safebook.

Mustafa Zengin 2010
Engineering Safebook: The new generation social network.

Jussi Kyröhonka 2005
Key management in Multicast communications.

Student projects

Abdelkrim Ahmed-Bacha, Amine Eddeghai 2016
Searchable encryption with Cuckoo hashes

Duc Trinh, Khang Nguyen 2016
Verifiable storage and computation

Daniela Matheu Vasco, Javier Quinones 2016
Building an integrated IPSEC laboratory.

Shahriyar Mammadov, Andre Mitsutake Cueto, Cesar Loayza 2016
Building an integrated Firewall laboratory.

Amine Semma, Luc Bonnafoux 2016
Verifiable word search.

Sergio Schena, Mauro Sardara 2015
Multi-user searchable encryption

Montida Pattaranantakul, Quang-An Dang, Dinh Cong Trang TRAN 2015
A PIR-based approach for secure data deduplication in cloud storage.

Rached Jemal, Mohamed Zarrouk 2015
Accountability for Cloud, Policy Enforcement and Auditing.

Efthymios Vlachos, Sajari Ray 2014
Accountability for Cloud and Policy Enforcement.

Sagar Uday-Kumar, Keerthi Kumar Kempaiah Honnappa 2014
CloudDedup: secure deduplication in the cloud.

Sachin Singh, Farhin Sarwar 2014
Dynamic proofs of retrievability.

Markus Suonto, Rached Jemal, Mohamed Zarrouk 2014
Proofs of retrievability.

Ghassane Amchyaa, Cédric Van Rompay 2014
Implementation and Evaluation of Privacy Preserving Delegated Word Search in the Cloud.

César Burini, Guillaume Gruber 2013
Accountability for clouds.

Sarath Mangalat, Ashutosh Pangasa 2012
Proof of Retrievability in Cloud Computing.

Yu Liu, Yuling Shi 2012
Secure data and key management for Safebook.

Hung Phan, Ba Hoang Le 2012
Proof of Ownership for cloud storage systems.

Esko Mattila 2012
Android version of Safebook.

Kasturi Borah, Mohamed Yasr Farook 2011
Privacy preserving word search in Cloud Computing.

Wenting Li 2011
Modelling of Safebook.

Bachir Chraibi, Eric Djatsa-Yota 2011
Private search for Map Reduce.

Etienne Perron, Jean-Baptiste Barrau, Paolo Viotti 2010
Security and privacy features of Safebook user interface.

Alexandre Bouard, Matthieu Gedon-Monaco 2010
Building an integrated PKI Laboratory.

Yao Liu 2009
Safebook simulation.

Paulin Kitieu Nanfack, Ghislain Kamga Nogha 2009
Secret Matching in Delay Tolerant Network.

Saad Bounjoua, Brahim Boulai, Mehdi Khalfaoui 2009
Secret matching in Second Life.

Imam Habibi, Tiab El-Hadi 2008
Prototype for secure network coding.

Gaël Charrière, Nidhal Chouk 2008
Security Manager for DTN.

Charles-Eric Boulangé 2008
Secure mail application for DTN.

Marianna Carrera, Lorenzo Odorico 2006
Security Manager for a Wireless Ad Hoc Network Node.

Imane Boulanouar, Ahamd-Reza Bakhtiari 2006
Simulation of secure data aggregation in sensor networks.

Teaching experience

Building and managing several labs on Networking and Security 2003-2016

Networking

Public Key Infrastructure (PKI)

Firewall

Cryptography

IPSEC

Scientific and Dissemination activities

Conference/Workshop organisation

Co-Chair and Founder of SESOC, IEEE Workshop on Security and Social Networking (2008-2014)

TPCs

(2016): SACMAT, ANT, CHANTS, IoP, RTSI, CCSW, FPS, DPM, SPC, APVP;

(2015): DPM, DIHC, AOC, CCIoT, FPS, PerMoby, TrustCom, WETICE, ANT, WowMoM;

(2014): ANT, CHANTS, DPM, WoWMoM;

(2013): WoWMoM, PETS, ANT, PAIS, PerMoby, CFSE;

(2012): FPS, PADE, DPM, ICCS, MASS, WoWMoM, WiSec, SFCS;

(2011): CANS, MASS, WoWMoM, ACNS, CFSE;

(2010): INTERNET, ICDT, ICCS;

(2009): ICDT, VTC, PIMRC;

(2007): SecureComm.

Invited referee for journals and conferences (each listed only once)

Editor: COMCOM Special Issue on Opportunistic Networks;

Journals: PMC (Pervasive and Mobile Computing), COMCOM (Computer Communications), COMNET (Communication Networks), TOSN (Transactions on Sensor Networks), SCN (Security and Communication Networks), TISSEC (Transactions on Information and System Security), TCC (Transactions on Cloud Computing);

Conferences and Workshops: SRDS 2003, SAR 2004, WISA 2004, NDSS 2004, ISCC 2004, IFIP SEC 2004, ESORICS 2004, EuroPKI 2004, WoWMom 2005, SecureComm 2005, ACM CCS 2006, EuroNGI 2006, ICC 2006, IFIP SEC 2006, PerSec 2006, SecureComm 2006, Tspuc 2006, ACM Computing 2007, ChinaCom 2007, EUC 2007, Globecom 2007, ISIT 2007, MWCN 2007, PerCom 2007, SecureComm 2007, TDSC 2007, UCS 2007, VTC 2007, AfricaCrypt 2008, ESORICS 2008, Financial Crypto 2008, ICCS 2008, IFIP SEC 2008, PerSENS 2008, SecureComm 2008, WiSec 2008, WoWMom 2008, ICCCN 2009, Infocom 2009, WiSec 2009, VTC Spring 2009, ICCS 2010, Infocom 2010, PerCom 2010, WiSec 2010, , Infocom 2011, etc.ICC 2006, IFIP SEC 2006, PerSec 2006, SecureComm 2006, Tspuc 2006, ACM Computing 2007, ChinaCom 2007, EUC 2007, Globecom 2007, ISIT 2007, MWCN 2007, PerCom 2007, SecureComm 2007, TDSC 2007, UCS 2007, VTC 2007, AfricaCrypt 2008, ESORICS 2008, Financial Crypto 2008, ICCS 2008, IFIP SEC 2008, PerSENS 2008, SecureComm 2008, WiSec 2008, WoWMom 2008, ICCCN 2009, Infocom 2009, WiSec 2009, VTC Spring 2009, ICCS 2010, Infocom 2010, PerCom 2010, WiSec 2010, , Infocom 2011, PerCom 2011, Infocom 2012, WONS 2012, NDSS2013, PerSens2013, Financial Crypto 2013, SETOP 2013, CLASP 2014, CNS 2015;

Publications

Articles in refereed conferences

[1] Dimitrios Vasilopoulos, Melek Önen, Kaoutar Elkhiyaoui, and Refik Molva. Message-Locked Proofs of Retrievability with Secure Deduplication. In *Proceedings of the 8th ACM Cloud Computing Security Workshop*, 2016.

[2] Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui, and Refik Molva. Efficient Techniques for Publicly Verifiable Delegation of Computation. In *Proceedings of the ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS)*, 2016.

[3] J. Bringer, B. Gallego, G. Karame, M. Kohler, P. Louridas, M. Önen, H. Ritzdorf, A. Sorniotti, and D. Vallejo. TREDISEC: Trust-aware reliable and distributed information security in the cloud. In *6th International Conference on e-Democracy, E-DEMOCRACY*, 2015.

[4] I. Leontiadis, K. Elkhiyaoui, M. Önen, and R. Molva. PUDA - Privacy and Unforgeability for data aggregation. In *Proceedings of CANS*, 2015.

[5] M. Azraoui, K. Elkhiyaoui, M. Önen, and R. Molva. Publicly verifiable conjunctive keyword search in outsourced databases. In *Proceedings of the 1st IEEE Workshop on Security and Privacy in the Cloud*, 2015.

[6] P. Puzio, R. Molva, M. Önen, and S. Loureiro. PerfectDedup: Secure data deduplication. In *10th International Workshop on Data Privacy Management (DPM)*, 2015.

[7] Cédric Van Rompay, Refik Molva, and Melek Önen. Multi-user searchable encryption in the cloud. In *18th International Conference on Information Security (ISC)*, 2015.

[8] M. Azraoui, K. Elkhiyaoui, M. Önen, K. Bernsmed, A. Santana de Oliveira, and J. Sendor. A-PPL: An accountability policy language. In , 2014.

[9] Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva, and Melek Önen. Stealthguard: Proofs of retrievability with hidden watchdogs. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*, 2014.

[10] Kaoutar ELkhiyaoui, Melek Önen, and Refik Molva. Privacy preserving delegated word search in the cloud. In *Proceedings of 11th International conference on Security and Cryptography (SECRYPT)*, 2014.

[11] I. Leontiadis, R. Molva, and M. Önen. Privacy preserving statistics in the smart grid. In *1st International Workshop for Big Analytics for Security (DASEC), in conjunction with ICDCS*, 2014.

[12] I. Leontiadis, R. Molva, and M. Önen. A P2P based usage control enforcement scheme resilient to reinjection attacks. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WowMoM)*, 2014.

[13] W. Benghabrit, H. Grall, J.-C. Royer, M. Sellami, M. Azraoui, K. Elkhiyaoui, M. Önen, A. Santana de Oliveira, and K. Bernsmed. A cloud accountability policy representation framework. In *4th International Conference on Cloud Computing and Services Science (CLOSER)*, 2014.

[14] P. Puzio, R. Molva, M. Önen, and S. Loureiro. ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage. In *PRoceeings of the , 5th IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM)*, 2013.

[15] I. Leontiadis, M. Önen, R. Molva, M.J. Chorley, and G.B. Colombo. Privacy preserving similarity detection for data analysis. In *Collective Social Awareness and Relevance Workshop (CSAR), co-located with the 3rd international conference on Social Computing and its Applications*, 2013.

[16] E.-O. Blass, R. di Pietro, R. Molva, and M. Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. LNCS, July 2012.

[17] L.A. Cutillo, R. Molva, and M. Önen. PRICE: PRivacy preserving Incentives for Cooperation Enforcement. In *13th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, (WowMoM)*, 2012.

[18] L. A. Cutillo, R. Molva, and M. Önen. Privacy preserving picture sharing: Enforcing usage control in distributed online social networks. In *5th ACM Workshop on Social Network Systems (SNS)*, 2012.

[19] L. A. Cutillo, R. Molva, and M. Önen. Analysis of privacy in online social networks from the graph theory perspective. In *Proceedings of IEEE Globecom*, 2011.

[20] Y. Sun, X. Wang, M. Önen, and R. Molva. CrowdLoc : Wireless jammer localization with crowdsourcing measurements. In *2nd International Workshop on Ubiquitous Crowdsouring (UBICROWD) in conjunction with 13th ACM International Conference on Ubiquitous Computing*, 2011.

[21] Y. Sun, R. Molva, M. Önen, X. Wang, and X. Zhou. Catch the jammer in wireless sensor network. In *22nd Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2011.

[22] Abdullatif Shikfa, Melek Önen, and Refik Molva. Broker based private matching. In *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS)*, 2011.

[23] Abdullatif Shikfa, Melek Önen, and Refik Molva. Bootstrapping security associations in opportunistic networks. In *Proceedings of the 6th IEEE International Workshop on Mobile Peer-to-Peer Computing (MP2P)*, 2010.

[24] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy in context-based and epidemic forwarding. In *Proceedings of the 13rd IEEE International WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC)*, 2009.

[25] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy in content-based opportunistic networks. In *Proceedings of the 2nd IEEE International Workshop on Opportunistic Networking (WON)*, 2009.

[26] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy-preserving content-based publish/subscribe networks. In *Proceedings of the 24th IFIP International Information Security Conference (SEC)*, 2009.

[27] N. Oualha, M. Önen, and Y. Roudier. A security protocol for self-organizing data storage. In *23rd International Information Security Conference (IFIP SEC)*, 2008.

[28] Melek Önen, Abdullatif Shikfa, and Refik Molva. Optimistic fair exchange for secure forwarding. In *Proceedings of the 1st Workshop on the Security and Privacy of Emerging Ubiquitous Communication Systems (SPEUCS)*, 2007.

[29] M. Önen and R. Molva. Secure data aggregation with multiple encryption. In *4th European conference on Wireless Sensor Networks (EWSN)*, 2007.

[30] M. Önen and J. Kyröhonka. Simulation based performance evaluation of a user-centric group rekeying protocol. In *IEEE International Conference on Communications (ICC)*, 2006.

[31] M. Önen and R. Molva. Reliable group rekeying with a customer perspective. In *47th annual IEEE Global Telecommunications Conference (Globecom)*, 2004.

[32] M. Önen and R. Molva. Group rekeying with a customer perspective. In *10th International Conference on Parallel and Distributed Systems (ICPADS)*, 2004.

[33] M. Önen and R. Molva. Denial of service prevention in satellite networks. In *IEEE International Conference on Communications (ICC)*, 2004.

Articles in journals

[34] B. Gallego-Nicasio Crespo, M. Önen, and G. Karame. TREDISEC: Towards realizing a truly secure and trustworthy cloud. *ERCIM News, Special theme: Tackling Big Data in the Life Sciences*, (104), January 2016.

[35] P. Puzio, R. Molva, M. Önen, and S. Loureiro. Block-level de-duplication with encrypted data. *Open Journal of Cloud Computing (OJCC)*, 2014.

[36] A. Shikfa, M. Önen, and R. Molva. Local key management in opportunistic networks. *International Journal of Communication Networks and Distributed Systems, Special Issue on Recent Advances in P2P Systems*, 9(12), 2012.

[37] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy and confidentiality in context-based and epidemic forwarding. *Computer Communications*, 2010.

[38] N. Oualha, M. Önen, and Y. Roudier. Secure P2P data storage and maintenance. *International Journal of Digital Multimedia Broadcasting*, 2010.

Book chapters

[39] P. Puzio, R. Molva, M. Önen, and S. Loureiro. Secure deduplication with encrypted data for cloud storage. In *"Delivery and Adoption of Cloud Computing Services in Contemporary Organizations*, chapter 10. ICI Global, 2015.

[40] M.Önen, R. Molva, and A. Pannetrat. Sécurité des communications en multicast. In A. Benslimane, editor, *Multicast multimedia sur Internet*. Hermes Lavoisier, 2005.

PhD thesis

[41] Melek Önen. *Securing Multicast Communications in satellite networks: A user-centric approach.* PhD thesis, TélécomParisTech, EURECOM, July 2005.

Invited talks

SECODIC: Workshop on Secure and Efficient Outsourcing of Storage and Computation of Data in the Cloud, Salzburg, Austria 2016
Verifiable Polynomial Evaluation and Matrix Multiplication

SEC2: 2nd Workshop on Security in Clouds, Lorient, France 2016
Verifiability for Cloud Storage and Computation

Cybersecurity day - Siber Güvenlik Günü, TOBB-ETÜ, Ankara, Turkey 2016
Data Privacy and Security in Cloud Computing

IFIP Summer School on Identity Management and Privacy, Edinburgh, Scotland 2015
Privacy and Verifiability for Data Storage in Cloud Computing

Atelier Protection de la Vie Privée, Sorèze, France 2011
Safebook: A Distributed Privacy Preserving Online Social Network

DistTrust Workshop, Barcelona, Spain. 2006
Self organizing trust establishment and cooperation enforcement

2nd workshop RECAP, Rennes, France. 2006
SECURECAP : La sécurité dans les réseaux de capteurs

the 57th IETF meeting Vienna, Austria 2003
Research on Multicast security

ECOTEL 2002 Logiciels pour Télécommunications, Golfe-Juan, France 2002
Sécurité des Communications en multicast : applications pour les liens satellites

Atelier " Cryptographie pour les applications spatiales", CNES, Toulouse, France 2002
La sécurité des communications multicast

Selected Publications

The following publications are included into this document to give the reader a more in-depth view of some of the research results:

- Abdullatif Shikfa, Melek Önen, Refik Molva; *Broker-based private matching*, 11th Privacy Enhancing Technologies Symposium (PETS), Waterloo, Canada, 2011.
- Erik-Oliver Blass, Roberto di Pietro, Refik Molva, Melek Önen; *PRISM: Privacy preserving Search in Map Reduce*, 12th Privacy Enhancing Technologies Symposium (PETS), Vigo, Spain, 2012.
- Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva, Melek Önen; *StealthGuard: Proofs of retrievability with hidden watchdogs*, 19th European Symposium on Research in Computer Security (ESORICS), Wroclaw, Poland, 2014.
- Pasquale Puzio, Refik Molva, Melek Önen, Sergio Loureiro; *PerfectDedup: Secure data deduplication*, 10th Internationale Workshop in Data Privacy Management (DPM), Vienna, Austria, 2015.
- Cédric Van Rompay, Refik Molva, Melek Önen; *Multi user searchable encryption*, 18th International Conference on Information Security (ISC), Trondheim, Norway, 2015.
- Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui, Refik Molva; *Efficient techniques for publicly verifiable delegation of computation*, 11th ACM Asia Conference on Computer and Communications Security (ASIACCS), Xi'an, China, 2016.

Broker-Based Private Matching

Abdullatif Shikfa[1], Melek Önen[2], and Refik Molva[2]

[1] Alcatel-Lucent Bell Labs,
Route de Villejust, 91620 Nozay, France,
[2] EURECOM,
2229, route des crêtes, 06560 Sophia Antipolis cedex, France

Abstract. Private matching solutions allow two parties to find common data elements over their own datasets without revealing any additional private information. We propose a new concept involving an intermediate entity in the private matching process: we consider the problem of broker-based private matching where end-entities do not interact with each other but communicate through a third entity, namely the Broker, which only discovers the number of matching elements. Although introducing this third entity enables a complete decoupling between end-entities (which may even not know each other), this advantage comes at the cost of higher exposure in terms of privacy and security. After defining the security requirements dedicated to this new concept, we propose a complete solution which combines searchable encryption techniques together with counting Bloom filters to preserve the privacy of end-entities and provide the proof of the matching correctness, respectively.

1 Introduction

Imagine that a company has an opening for a new position. The posting for new position consists mainly of requirements in terms of education, professional experience and skills. So the company has many selection criteria and is looking for the best suited candidate. Since the company does not want to take care of all the recruitment process itself, it delegates the search phase to a recruitment agency, which is more capable in terms of publishing the posting for new position on a large scale. Candidates are characterized first by their resume and they apply through the recruiting agency if they think they are fit for the job. The recruitment agency upon receiving a resume, looks at the matching ratio between the candidate characteristics and the posting's criteria and calls the best suited candidates for an interview at the company. The best suited candidates are either all candidates above a certain matching ratio threshold, or the top ten candidates for example. In order to prevent resume fraud, candidates should be able to prove the correctness of their resume, with diplomas from a university or validation of experience from a governmental agency.

This interesting scenario raises many security issues. First of all, both company and candidates' privacy should be preserved. The company does indeed not want that competitors learn about the posting, especially if it concerns an

important position because that would give a hint about the company's strategy. So the posting and more specifically the criteria expressed by the company should remain secret from other companies, including the recruitment agency. Candidates' privacy should also be preserved, to enforce equal opportunities among candidates. Therefore resumes should be confidential and anonymous to prevent the recruiting agency from discriminating between candidates on a non-professional basis. The problem is therefore to be able to compute the matching ratio between the posting's criteria and the candidates' resumes while both are encrypted. Furthermore it is important that candidates cannot forge their resume to obtain a higher matching ratio. This problem is especially hard since resumes cannot be checked directly in the case where they are encrypted: privacy and verification present conflicting requirements.

At first glance this problem has a flavor of private matching or private set intersection, whereby two parties want to learn only shared attributes without learning any information about the remaining ones. There is yet an important difference in the presented scenario which makes the problem more complex: the parties owning the private data (the company and the candidates) do not directly interact with each other, but they forward their secret data to a third party. This third party has to take a decision on the matching ratio without having any control or knowledge on the private data it received, and it should not be able to learn anything about the private data of either party in the process: it should just be able to securely compute the matching ratio (it should not even be able to tell which of the encrypted data matched or not). This paper therefore tackles with a new requirement for parties not to interact directly to achieve the matching result thus calling for a non-interactive solution.

In this paper, we analyze the requirements for the non-interactive and private computation of matching ratio and present a complete solution to address this issue. The solution is based on a searchable encryption scheme introduced by Boneh et al. in [3] used in a new mode of operation to allow the company to issue a unique query for all potential (and unknown) candidates. The solution further makes use of counting Bloom filters introduced by Fan et al. in [11], but in a radically new approach: those counting Bloom filters are not used as usual to prove the belonging of an element to a set but to compute the matching ratio without leaking privacy and to provide evidence of the correctness of the matching ratio computation. This solution presents the following advantages:

- it addresses the non-interactive scenario as it does not require the parties owning the private data to interact with each other (such as setting up keys prior to the matching process for example) or even to know each other,
- it allows a third party to compute the matching ratio and to get evidence of its correctness,
- it preserves the privacy of data, because the third party processes encrypted data blindly (in the sense that it handles encrypted data and does not learn any information about it),
- it is efficient, because the third party, which has to process a lot of data from several users, only needs to perform few and non-costly operations for the computation of each matching ratio.

The rest of the paper is structured as follows. Section 2 motivates the need for a broker-based private matching protocol comparing it with the classical two-party mechanisms, defines the security requirements and describes the underlying mechanisms. In section 3, the overall protocol and its security primitives are described in detail. The security and performance of the proposed protocol are evaluated in section 4. Finally, section 5 discusses relevant related work.

2 Problem Statement

2.1 Private matching: introducing a third party

The classical private matching scheme is a two-party protocol that enables both parties P_1 and P_2 to discover common data elements over their own datasets without revealing any additional private information. Assuming that P_1 and P_2 respectively own datasets X_1 and X_2, at the end of the private matching protocol P_1 and P_2 only learn $X_1 \cap X_2$.

In this paper, we propose a complete decoupling between these two parties in order to perform the same operation when the two parties do not interact and are even not aware of each other. The new protocol involves a third party, the Broker, which is in charge of computing the cardinality of the matching set without discovering any of its elements. Private and correct evaluation of the cardinality of the matching set by a third party has many applications, in particular for ranking, or finding friends in social networks or simply in dating sites, and compelling new applications are envisioned in the broad field of cloud computing. All these applications require a third party to take decisions while remaining oblivious to the matched information. This new broker-based private matching protocol consists of three entities, namely the **Query Issuer**, the **Subject** and the **Broker**, where the latter's main role is to discover the cardinality of the matching set originating from the other two entities' datasets. Each entity's role in the new protocol is formally defined as follows:

- the **Query Issuer** $\mathcal{QI}$ issues a query $Q_i = \langle q_{i,1}, ..., q_{i,n} \rangle$ consisting of n selection criteria which are elements of D, the global dataset. In the recruitment example, the company is the Query Issuer and an example of selection criterion could be "Degree = MSc".
- **Subjects** $\mathcal{S}^l$ $(1 \leq l \leq c)$, answers a query Q_i with a matching proof $mp_{i,l}$ based on its profile. Each Subject is indeed characterized by a profile $P^l = \langle p^l_1, ..., p^l_m \rangle$ composed of m attributes which are elements of the same dataset D. These attributes are evaluated with respect to the query defined by the Query Issuer. In the aforementioned scenario, Subjects correspond to the candidates in the recruitment process.
- the additional party, namely the **Broker** $\mathcal{B}$, first publishes the query of $\mathcal{QI}$ to Subjects and collects their answers. The Broker then selects the best suited Subjects: $\mathcal{B}$ computes a matching ratio $\rho_{i,l}$ between a query Q_i and the Subject's answer $mp_{i,l}$ defined as the cardinality of the matching set between the selection criteria and the attributes over the cardinality of the selection criteria. In the example, the Broker is the recruiting agency.

In summary, the major difference between classical private matching and the broker-based private matching protocol is the fact that there is no direct interaction between the Query Issuer $\mathcal{QI}$ and Subjects $\mathcal{S}^l$. All messages go through the Broker $\mathcal{B}$, which is an active entity in the protocol and not a simple relay: the query Q_i of $\mathcal{QI}$ is sent to $\mathcal{B}$ which then publishes it to $\{\mathcal{S}^l\}_{1\leq l\leq c}$; each Subject $\mathcal{S}^l$ sends its answer $mp_{i,l}$ to $\mathcal{B}$ which decides which Subjects correspond to the query the best. Therefore, $\mathcal{QI}$ should be able to send a query without even knowing the Subjects $\{\mathcal{S}^l\}_{1\leq l\leq c}$: there is a complete decoupling between these two entities, and $\mathcal{B}$ is in charge of gathering the necessary data and taking the appropriate decision. Finally $\mathcal{QI}$ should be able to send a query with any selection criteria and is not limited to a set that it owns.

2.2 Security requirements

The introduction of a third party in the private matching protocol requires to revisit all security requirements defined for the two-party protocol.

First of all, we assume that the Query Issuer is interested in getting the best suited Subjects; therefore $\mathcal{QI}$ is assumed to be honest. On the contrary, Subjects are considered to be potentially malicious, because it is in their interest to exhibit a high matching ratio in order to be selected by the Broker. Therefore Subjects might attempt to cheat on their attributes or more generally in the answer they send to $\mathcal{B}$ in order to lure $\mathcal{B}$ into computing a matching ratio higher than their real matching ratio. However we consider that nodes are selfish and that they do not collude with each other.

Concerning the Broker $\mathcal{B}$, we assume it to be honest but curious: $\mathcal{B}$ correctly executes the protocol and computes the matching ratio according to the data it receives, and finally sends to $\mathcal{QI}$ the truly best suited Subjects according to the matching ratio rankings. Yet, $\mathcal{B}$ is curious in the sense that it is interested in unveiling information from the private data it receives, whether being the selection criteria of the query of $\mathcal{QI}$ or the attributes of Subjects.

There are thus two main attacks to be considered:

- attacks by the Broker in an attempt to break the privacy of the other two entities: $\mathcal{B}$ tries to discover and reveal the content of the query of $\mathcal{QI}$, or to discover the attributes of one or many Subjects,
- attacks by Subjects aiming at illegitimately increasing their matching ratio with a given query.

This leads to the following two security requirements:

- **Preserving the privacy of the end entities $\mathcal{QI}$ and $\mathcal{S}^l$**: queries issued by $\mathcal{QI}$ and answers of Subjects are confidential and therefore encrypted. The Broker should be able to compute the matching ratio using these two encrypted values without discovering any information about either the criteria of $\mathcal{QI}$ or the Subject's attributes: the protocol should be semantically secure. Furthermore, as for classical private matching protocol, since the query is forwarded by $\mathcal{B}$ to Subjects, these entities should not be able to derive information about non-matching criteria. These privacy properties can also be formally defined by comparing the real situation in our protocol with an ideal situation where the protocol is run by a trusted external entity, but we do not add this formalization in this article for the sake of clarity.

– **Guaranteeing the correctness of the matching ratio**: the answer $mp_{i,l}$ of a Subject $\mathcal{S}^l$ should enable the Broker to correctly compute the matching ratio between the query Q_i and the attributes of $\mathcal{S}^l$. This requirement is very different from the privacy one, but the latter hardens the task of verifying the correctness of the matching ratio. Indeed, a simple solution to this provably correct matching ratio computation would consist in the Subjects sending their attributes to the Broker, but this solution blatantly exposes the privacy of the Subjects. The challenge for the Broker is to be able to compute the matching ratio corresponding to a set of attributes while verifying their correctness without having access to their content.

2.3 Security primitives

Based on the security requirements of the broker-based private matching protocol, we define the following security primitives:

– `SQE` (Secure Query Encoding): in order to ensure the confidentiality of the query Q_i, this primitive, used by the Query Issuer $\mathcal{QI}$, securely encodes Q_i and returns Q'_i. $\mathcal{QI}$ can express its queries on any selection criteria in the global dataset D, hence `SQE` should be a public function in that it should not require any secret information on input. Furthermore, this function should be randomized to prevent dictionary attacks.
– `SLU` (Secure Look-up): A Subject $\mathcal{S}^l$ uses this primitive to look-up its attributes against an encoded query, and outputs the corresponding answer $mp_{i,l}$. This function should be public but requires secret information (credentials) to be processed.
– `SMRC` (Secure Matching Ratio Computation): on input of a Q'_i and a corresponding $mp_{i,l}$, this primitive first verifies the correctness of $mp_{i,l}$:
  1. if $mp_{i,l}$ is invalid ($\mathcal{S}^l$ attempted to cheat), the process breaks;
  2. otherwise, the primitive outputs the correct matching ratio $\rho_{i,l}$.

In the next section, these three primitives are formally described based on a combination of different cryptographic mechanisms: searchable encryption and counting Bloom filters.

3 Solution

We now present our solution by first introducing the underlying mechanisms and further by formally describing the overall protocol divided into two phases.

3.1 Overview

In order to allow the correct execution of the new protocol, Subjects first need to retrieve their credentials (private information corresponding to their profile) from a certain authority that approves their validity. Therefore, a trusted authority is initially available during a setup phase. This authority does not play any role during the execution of the matching protocol, namely the runtime phase.

In this second phase, the broker-based private matching protocol actually takes place, and it features four main steps:

1. Query: The Query Issuer $\mathcal{QI}$ issues a query Q_i. It encodes this query using the `SQE` primitive and sends the result Q'_i to the Broker. Based on the query Q_i, $\mathcal{QI}$ also constructs a counting Bloom filter CBF_i, called a matching reference and sends it to the Broker along with the encoded query.

2. Publish: The Broker publishes the encoded query $Q'i$ to all Subjects. The matching reference is not forwarded.
3. Look-up: Each subject $\mathcal{S}^l$ looks-up its credentials in the encoded query Q'_i to determine which conditions $\mathcal{S}^l$ matches. Based on these matched conditions, $\mathcal{S}^l$ constructs another counting Bloom filter $CBF_{i,l}$, called a matching proof. This matching proof $mp_{i,l}$ is sent to the Broker.
4. Verify: The Broker compares the matching reference and the matching proof to assess first whether the matching proof is valid or not, and then to compute the matching ratio $\rho_{i,l}$. Finally, the Broker informs $\mathcal{QI}$ about the Subjects best suited to its query Q_i.

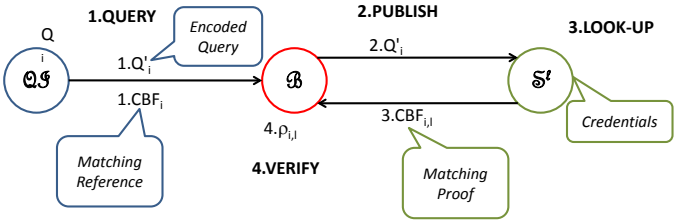

Fig. 1. High level description of the protocol

The protocol is summarized in figure 1. A major advantage of our solution is that it enables some computation on encrypted data to preserve end-entities privacy: the Broker is able to compute the matching ratio based on two encrypted data structures, the matching reference and the matching proof. This computation on encrypted data is achieved thanks to an extension of a searchable encryption mechanism that allows a third node to verify whether an encrypted keyword is included in a database or not. This mechanism is also combined with counting Bloom filters in order to prove the correctness of the computation of the matching ratio. Before formally describing the new protocol, these two mechanisms are briefly presented in the next section.

3.2 Background-Tools

Searchable encryption Searchable encryption is a general concept which enables a third entity to store an encrypted list destined to a certain party and to look-up encrypted keywords on behalf of this party without learning additional information both on the keyword and the encrypted list.

One of the main searchable encryption approaches was proposed by Boneh et al. in [3] and it uses three main operations:

- `SE-Encrypt`: a public encryption function used to encrypt the list that is stored by the third party. This function requires the knowledge of the public key of the destination.
- `SE-Trapdoor`: a private function which gives the capability of looking-up a specific keyword, called a trapdoor. This function requires the private key of the recipient and hence can only be computed by the recipient.
- `SE-Test`: on input of a trapdoor and an encrypted keyword, the third party uses this operation to verify whether the private keyword is included in the

list or not. Hence, this function returns 1 if the trapdoor corresponds to the encrypted keyword and 0 otherwise.

Due to its non-interactivity this searchable encryption proposal looks appropriate for the new broker-based private matching scenario, where the `SE-Test` operation can be implemented by the Broker while Query Issuers may encrypt some keywords with `SE-Encrypt` and the Subjects run the `SE-Trapdoor`. Unfortunately, the use of this mechanism is not straightforward because:

- As opposed to the `SE-Test` operation, the Broker should only be able to compute the global matching ratio and not individual matching attributes;
- The Query Issuer does not know the Subjects in advance, hence it does not have knowledge of their public keys and cannot use `SE-Encrypt` easily.

To circumvent these two main constraints, we propose to introduce a Trusted Third Party which alleviates the requirement of the knowledge of the (unknown) recipient's public key in our scheme (see section 3.3).

Bloom filters A Bloom filter is a probabilistic data structure which was first introduced by Burton Bloom ([5]). The classical use of Bloom filters is to test whether an element is a member of a set in a space-efficient way. We focus on an extension of Bloom filters called counting Bloom filters that were proposed by Fan et al. in [11] to support the dynamic deletion of an element.

A counting Bloom filter CBF is an array of ϕ positions (also called buckets) used to represent a set $\mathcal{X}$ with the aid of u hash functions $\{h_1, .., h_u\}$ mapping an element of $\mathcal{X}$ to one of the ϕ array positions. Counting Bloom filters implement the following three functions:

- $\texttt{Insert}(x, CBF)$: on input of an element x, the digest of this element is computed using each of the u hash functions. The values of the filter CBF at these positions are incremented by 1.
- $\texttt{Query}(x, CBF)$: this function verifies with a certain probability whether x is an element of the filter or not.
- $\texttt{Delete}(x, CBF)$: this operation consists of decrementing the value at each of the u positions resulting from the hash functions evaluated over x, by 1.

In the sequel of this article, we denote by $\texttt{CBF}(x_1, ..., x_n)$ the counting Bloom filter obtained by inserting the elements x_i for $1 \leq i \leq n$.

The weight w_{CBF} of a counting Bloom filter CBF is defined as the sum of the values of all positions: $w_{CBF} = \sum_{0 \leq i \leq \phi - 1} CBF[i]$. An important property of counting Bloom filters is that the weight w_{CBF} of a counting Bloom filter CBF is linearly dependent on the number of elements inserted in it:

$$w_{\texttt{CBF}(x_1,...,x_n)} = n \cdot u.$$

Hence, counting Bloom filters are useful for our broker-based private matching as they enable computing the cardinality of a set without revealing the elements of the set (see section 3.3).

3.3 Construction

As mentioned in section 3.1, the solution features two phases: a setup phase where Subjects retrieve their credentials, and a runtime phase where the private matching protocol is executed.

Setup phase Contrary to $\mathcal{QI}$ which can choose any selection criteria in Q_i, $\mathcal{S}^l$ should answer Q'_i correctly based on their profile. Since the correctness of private matching operations depends on the correctness of these profiles, the latter should be certified, and we refer to the certified attributes as credentials. These credentials are retrieved during a setup phase which features a fourth entity, called the **Authority** $\mathcal{A}$. This Authority is required to define general parameters of the system and to provide Subjects with their matching credentials.

The general parameters are generated according to a security parameter which impacts the size of the groups that are used, as well as the size of keys. In particular, the Authority generates a private and public key pair $sk_{\mathcal{A}}/pk_{\mathcal{A}}$. In the recruitment example, universities delivering a diploma or governmental agencies can be considered as authorities.

In addition to the three security primitives defined in section 2.3, we define a fourth one, `SCE` (Secure Credential Extraction), which is used by $\mathcal{A}$ to provide $\mathcal{S}^l$ with the credentials corresponding to its profile (this primitive is similar to the private key extraction primitive in Identity-Based Encryption). On input of a Subject's profile, `SCE` returns a set of credentials. These credentials are used as matching capabilities and correspond to trapdoors in searchable encryption.

To be more precise, Subjects $\mathcal{S}^l$ first contact the Authority $\mathcal{A}$ and show their profile $P^l = \langle p^l_1, ...p^l_m \rangle$. $\mathcal{A}$ verifies the validity of P^l (this verification step is out of the scope of this paper), and then provides $\mathcal{S}^l$ with the corresponding credentials T^l which are computed using the `SE-Trapdoor` function applied over the Subject's attributes and the secret key of $\mathcal{A}$. Hence, at the end of the setup phase, each S^l receives the following set of credentials:

$$T^l = \langle t^l_1, ..., t^l_m \rangle = \langle \texttt{SE-Trapdoor}(p^l_1, sk_{\mathcal{A}}), ..., \texttt{SE-Trapdoor}(p^l_m, sk_{\mathcal{A}}) \rangle.$$

Runtime phase As described in section 3.1, the runtime phase consists of four main steps that we describe formally in the following:

1. **Query:** During this step, $\mathcal{QI}$ expresses a query Q_i by choosing a set of selection criteria and performs a secure encoding of the query using the `SQE` primitive. The output of this primitive are the encoded query Q'_i and the matching reference mr_i: $\texttt{SQE}(Q_i, pk_{\mathcal{A}}) = (Q'_i, mr_i)$.
   As previously introduced, the `SQE` primitive should be a randomized public cryptographic function, such as `SE-Encrypt`. However, `SE-Encrypt` requires the public key of the recipient and this key is unknown to $\mathcal{QI}$, hence we propose a new configuration where the public key of $\mathcal{A}$ is used instead. Therefore, the encoded query is computed as follows:

   $$Q'_i = \langle q'_{i,1}, ..., q'_{i,n} \rangle = \langle \texttt{SE-Encrypt}(q_{i,1}, pk_{\mathcal{A}}), ..., \texttt{SE-Encrypt}(q_{i,n}, pk_{\mathcal{A}}) \rangle.$$

   On the other hand, the matching reference should help the Broker to compute the matching ratio correctly. To this extent, during the execution of the `SE-Encrypt` algorithm, $\mathcal{QI}$ also retrieves some intermediate values which can only be computed by itself or by the nodes that own the corresponding trapdoors. Indeed, the `SE-Encrypt` primitive makes use of a cryptographic

hash function H at the last step of the computation[3]. For $1 \leq j \leq n$, we denote the preimage of $q'_{i,j}$ by $x_{i,j}$:

$$q'_{i,j} = \texttt{SE-Encrypt}(q_{i,j}, pk_{\mathcal{A}}) = H(x_{i,j}).$$

Thanks to the inherent security of the hash functions with pseudorandom inputs, a malicious user cannot compute $x_{i,j}$ based on the knowledge of $q'_{i,j}$. Hence, $\mathcal{QI}$ constructs the matching reference mr_i as a counting Bloom filter CBF_i, in which it inserts the elements $x_{i,j}$ for $1 \leq j \leq n$:

$$mr_i = CBF_i = \texttt{CBF}(x_{i,1}, ..., x_{i,n}).$$

At the end of this first step, $\mathcal{QI}$ sends mr_i and Q'_i to the Broker.

2. **Publish:** The Broker forwards the encoded query Q'_i to all Subjects but keeps the matching reference mr_i.
3. **Look-up:** On input of an encoded query Q'_i and a set of credentials T^l, the SLU primitive returns a matching proof $mp_{i,l}$:

$$\texttt{SLU}(Q'_i, T^l) = mp_{i,l}.$$

By using the SE-Test function, Subjects can indeed detect selection criteria corresponding to their profile: for $1 \leq j \leq n, 1 \leq k \leq m$ $\texttt{SE-Test}(q'_{i,j}, t^l_k)$ returns 1 for matching elements and 0 for the others. Moreover, the Subject can compute the corresponding preimage $x_{i,j}$ for matching criteria. Hence $\mathcal{S}^l$ can construct a counting Bloom filter $CBF_{i,l}$ in which it includes all the preimages that it managed to compute and which is used as matching proof $mp_{i,l} = CBF_{i,l}$ and sent to the Broker.

4. **Verify:** On input of a matching reference mr_i and a matching proof $mp_{i,l}$ the primitive SMRC returns a matching ratio $\rho_{i,l}$.

The Broker first compares the counting Bloom filters CBF_i and $CBF_{i,l}$ to assess the validity of the latter. To this extent, the Broker checks whether:

- $\forall 0 \leq i_1 \leq \phi - 1, CBF_{i,l}[i_1] \prec CBF_i[i_1]$ denoted as $CBF_{i,l} \prec CBF_i$, otherwise it means that $CBF_{i,l}$ was not constructed only with (a subset of) $x_{i,1}, ...x_{i,n}$,
- the weight $w_{CBF_{i,l}}$ of $CBF_{i,l}$ is a multiple of u, because each inserted element leads to an increase of the weight by u.

If one of the verifications fails, the protocol aborts (the Subject attempted to cheat), otherwise the Broker accepts the answer of $\mathcal{S}^l$ as being valid and computes the matching ratio as follows:

$$\texttt{SMRC}(mr_i, mp_{i,l}) = \frac{w_{CBF_{i,l}}}{w_{CBF_i}}.$$

The protocol is consistent in that:

Proposition 1. *If $CBF_{i,l}$ is generated as specified in the protocol, then the matching ratio between the query and the attributes of a Subject corresponds to the output of SMRC:*

$$\rho_{i,l} = \mathit{SMRC}(mr_i, mp_{i,l}).$$

[3] See [3] for the detailed construction of PEKS. We roughly have $x_{i,j} = \hat{e}(H_1(q_{i,j}), r \cdot pk_{\mathcal{A}})$, and $q'_{i,j} = \langle rP, H(x_{i,j}) \rangle$, where $\hat{e}$ is a bilinear map, r a random scalar, and P a point on an elliptic curve.

This proposition is a direct consequence of the fact that the weight of a counting Bloom filter is linearly dependent with the number of its elements.

This concludes the presentation of our solution, and we now evaluate its security and performance.

4 Evaluation

The security of the new broker-based private matching protocol is analyzed based on the attacker model and the security requirements defined in section 2.2. We assume that the communication channels between $\mathcal{QI}$ and $\mathcal{B}$ and between $\mathcal{B}$ and $\mathcal{S}^l$ are secured, hence eavesdroppers cannot access the messages exchanged in the protocol in clear. They thus have less information than any of the entities running the protocol, and we do not further take them into account.

4.1 Privacy

Privacy is the most important requirement in classical private matching. In this section, we assume that entities are curious and try to discover information that they should not access. We first show that our solution preserves the privacy of end-entities and we further prove that the introduction of a third party (the Broker) does not threaten the Query Issuer's and Subjects' privacy.

First, the privacy of the $\mathcal{QI}$ is preserved with respect to $\mathcal{S}^l$. Indeed, in [3], Boneh et al. proved that their construction is semantically secure against a chosen keyword attack in the random oracle model, assuming that the Bilinear Diffie-Hellman problem is hard. It is thus unfeasible for an entity to discover the value of an encoded selection criteria unless it knows the corresponding trapdoor, in other words $\mathcal{S}^l$ can only discover the matching selection criteria. Furthermore, since only the Authority $\mathcal{A}$ knows the private key $sk_{\mathcal{A}}$, nodes cannot forge trapdoors. Recovering the private key $sk_{\mathcal{A}}$ amounts to a discrete logarithm computation which is assumed to be hard.

Second, we prove that the introduction of $\mathcal{B}$ does not threaten the privacy of end-entities. On one hand, as an intermediate node, $\mathcal{B}$ receives the same encoded queries that $\mathcal{S}^l$ receives, but $\mathcal{B}$ has no trapdoors and thus cannot discover the value of the encoded queries. Furthermore, $\mathcal{B}$ cannot link successive queries even if they correspond to the same selection criteria because the encoding mechanism is inherently randomized. On the other hand, in addition to the queries, $\mathcal{B}$ receives matching reference and matching proofs from $\mathcal{QI}$ and $\mathcal{S}^l$ respectively. As proven in the following theorem, the knowledge of a counting Bloom filter does not enable the Broker to recover the elements $x_{i,j}$ inserted in it.

Theorem 1. *Let $x_1, ..., x_n$ be n elements randomly chosen from a group G of order q. Let CBF be a counting Bloom filter of size ϕ in which the n elements $x_1, ..., x_n$ were inserted using u hash functions $h_1, ..., h_u$. Then, there are more than $\frac{q}{\phi^u}$ possible sets of elements of G^n leading to the same counting Bloom filter:*

$$|\{(x'_1, ..., x'_n) \in G^n | \texttt{CBF}(x'_1, ..., x'_n) = \texttt{CBF}(x_1, ..., x_n)\}| > \frac{q}{\phi^u}$$

The proof is given in section 7.1. This result is a lower bound on the set of preimages but the actual result can be multiplied by a factor of up to $u!$ depending on the outputs of the hash functions, and is multiplied even further if more elements are inserted. Note that this result does not even take into account the complexity required to find the corresponding set of preimages.

From the perspective of an attacker, being able to solve the equations would lead to an advantage as it reduces the size of the space of possibilities from q down to $\frac{q}{\phi^u}$. However, careful setting of the parameters q, ϕ and u, makes the size of this set large enough to prevent brute force guessing (see section 7.3).

In summary, the counting Bloom filter cannot be reversed to obtain the entries that were inserted in it, which guarantees the privacy of the Query Issuer and Subjects. We now focus on the security of the matching ratio computation.

4.2 Correctness of the matching ratio

Concerning the security of the matching ratio computation, we consider now a malicious $\mathcal{S}^l$ trying to convince $\mathcal{B}$ that its matching ratio is higher than its actual value, and we show that the probability of success of such an attack is negligible.

To be more precise, we assume that $\mathcal{S}^l$ does not know the matching reference mr_i, thus the only information known by $\mathcal{S}^l$ on CBF_i are the global parameters: the hash functions used h_1,...,h_u and the size ϕ. $\mathcal{S}^l$ also knows Q'_i and therefore the number n of elements $x_{i,j}$ inserted in CBF_i.

The goal of the malicious $\mathcal{S}^l$ is to claim a matching ratio $\rho_{i,l}^{claim}$ higher than the actual ratio $\rho_{i,l}$. To this extent, $\mathcal{S}^l$ needs to claim a corresponding counting Bloom filter $CBF_{i,l}^{claim}$. For $\mathcal{S}^l$ to be successful, $CBF_{i,l}^{claim}$ has to verify the following conditions:

1. it should be considered valid by $\mathcal{B}$, as required by the last step of the protocol described in section 3.3, which implies that:
   - $CBF_{i,l}^{claim} \prec CBF_i$,
   - the weight $w_{CBF_{i,l}^{claim}}$ of $CBF_{i,l}^{claim}$ is a multiple of u,
2. it should lead to $\rho_{i,l}^{claim} > \rho_{i,l}$, hence the weight of $CBF_{i,l}^{claim}$ needs to verify $w_{CBF_{i,l}^{claim}} > w_{CBF_i}$.

The probability of success of $\mathcal{S}^l$ is exponentially decreasing in the malicious ratio increment $\rho_{i,l}^{claim} - \rho_{i,l}$, as shown in the following theorem.

Theorem 2. *Let Q'_i be an encoded query concerning n selection criteria. Let CBF_i be the corresponding matching reference.*

The probability of success $\mathcal{P}_{adv}[\rho_{i,l} \to \rho_{i,l}^{claim}]$ of an adversary $\mathcal{S}^l$ in generating an array $CBF_{i,l}^{claim}$ which is accepted by $\mathcal{B}$ and results in an increase of the matching ratio from $\rho_{i,l}$ to $\rho_{i,l}^{claim}$ is upper bounded by:

$$\mathcal{P}_{adv}[\rho_{i,l} \to \rho_{i,l}^{claim}] \leq \left(1 - e^{-\frac{(1-\rho_{i,l})n \cdot u}{\phi}}\right)^{(\rho_{i,l}^{claim} - \rho_{i,l})n \cdot u}$$

The proof is given in section 7.2. The formula of $\mathcal{P}_{adv}[\rho_{i,l} \rightarrow \rho_{i,l}^{claim}]$ shows that the probability of success of an adversary decreases exponentially with the malicious ratio increase $(\rho_{i,l}^{claim} - \rho_{i,l})$ and, decreases also depending on the value of the legitimate matching ratio $\rho_{i,l}$.

It is possible to go further and bound $\mathcal{P}_{adv}[\rho_{i,l} \rightarrow \rho_{i,l}^{claim}]$ independently of $\rho_{i,l}$ and $\rho_{i,l}^{claim}$, by observing that:

- the function $x \mapsto \alpha^x$ decreases with x for $0 < \alpha < 1$,
- $0 < \left(1 - e^{-\frac{(1-\rho_{i,l})n \cdot u}{\phi}}\right) < \left(1 - e^{-\frac{n \cdot u}{\phi}}\right) < 1$,
- $u < (\rho_{i,l}^{claim} - \rho_{i,l})n \cdot u$.

Hence, the probability of success of the adversary is bounded by $\mathcal{P}_{adv}$:

$$\mathcal{P}_{adv} = \left(1 - e^{-\frac{n \cdot u}{\phi}}\right)^u.$$

The security of the scheme hence depends on the general parameters of the counting Bloom filter and we now show how to optimize these settings.

First of all, we assume that the maximum number of selection criteria in a query is bounded and known in advance; we designate it as n_{max}. For all queries, the probability of success of the adversary is thus bounded by

$$\mathcal{P}_{adv} \leq \left(1 - e^{-\frac{n_{max} u}{\phi}}\right)^u.$$

If we fix ϕ, then the function $p_{max} : u \mapsto \left(1 - e^{-\frac{n_{max} u}{\phi}}\right)^u$ is $\mathcal{C}^\infty$ on $[1, +\infty[$, and it reaches its minimum in $u_0 = \frac{\phi}{n_{max}} ln(2)$ and $p_{max}(u_0) = 2^{-u_0}$. Therefore, for a fixed n_{max}, increasing u and ϕ exponentially increases the security, but increasing ϕ linearly impacts on the performance of the scheme. We propose the following strategy to optimize the trade-off between security and performance:

1. Set n_{max} the maximum number of criteria per query,
2. Choose a security parameter u: $\mathcal{P}_{adv}$ is then bounded by 2^{-u},
3. Set the size ϕ of the counting Bloom filter as $\phi = \left\lceil \frac{n_{max} u}{\ln(2)} \right\rceil$.

This strategy prioritizes security over performance: it defines the desired security level ($\mathcal{P}_{adv} \leq 2^{-u}$) and then sets the minimal size ϕ to achieve this security level. Note that u does not need to be very large, because $\mathcal{P}_{adv}$ is an upper bound and is obtained with very restrictive conditions:

- $n = n_{max}$, which means that $\mathcal{QI}$ uses n_{max} selection criteria,
- $\mathcal{S}^l$ has a legitimate matching ratio of 0 ($\rho_{i,l} = 0$).

With these conditions, $\mathcal{S}^l$ has a probability less than 2^{-u} of success in making $\mathcal{B}$ believe that its matching ratio is $1/n_{max}$ instead of 0. In many cases, this would not be of any use to the attacker, because the attacker needs to claim the highest matching ratio among the Subjects in order to take advantage of its attack. The attacker does not even know the matching ratio of the other Subjects, so the only way for the malicious $\mathcal{S}^l$ to be sure to benefit from its attack is to claim a matching ratio of 1, and the probability of $\mathcal{S}^l$ succeeding falls down to $2^{-u \cdot n_{max}}$.

4.3 Performance evaluation

Following the analysis of the trade-off between security and performance in the previous section, we now evaluate the overall communication and computational overhead resulting from the proposed protocol.

Communication overhead We consider that the cost originating from the setup phase is negligible given that it takes place offline. We only evaluate the communication overhead during the runtime phase.

The size of encoded queries is linear in the number of selection criteria that it includes. Each encoded criterion is the output of the `SE-Encrypt` primitive and thus has size $2q$ bits, where q is the size of the group used in the searchable encryption scheme.

Concerning the size of counting Bloom filters, they are arrays containing ϕ buckets. According to [11], we choose $\beta = 4$ bits for the size of each bucket to keep a negligible probability of overflow, thus the communication overhead incurred by the matching reference or the matching proof is 4ϕ bits.

Computational overhead The primitives of searchable encryption rely on elliptic curve operations which cost is of the same order of magnitude as classical asymmetric cryptography [18]. The most costly operation is the pairing computation: our mechanism requires one pairing computation per encoding and one per `SE-Test` evaluation, the cost is thus linear in the number of selection criteria used in the queries. In comparison, the cost of generating the counting Bloom filters which amounts to $n \cdot u$ hash computations is negligible.

The aforementioned computations are performed by the end-entities, but the Broker only carries on simple operations to compute the matching ratio:

- $\mathcal{B}$ verifies that the matching proof is smaller than the matching reference which requires ϕ integers inequality checks,
- $\mathcal{B}$ computes the weight of the matching proof and reference (a sum of ϕ integers) and performs a division.

The overhead on $\mathcal{B}$ is thus very small which shows that our scheme is scalable and efficient to disseminate a query to multiple Subjects.

5 Related work

Several previously studied problems in the literature show similarities with broker-based private matching. We list them in two main categories and show how they differ from our problem.

5.1 Private matching and private set intersection

Private matching came up as a generalization of private equality tests. A first approach introduced a Trusted Third Party (TTP) as proposed in [2] and [15]. In theses proposals, the TTP is completely trusted, computes $X_1 \cap X_2$ and sends the result back to P_1 and P_2. This solution is not satisfying from a privacy perspective as it is fully dependent on the honesty of the TTP which has full

access to the parties' sets. This three-party protocol is thus very different from our broker-based private matching solution.

In [1], Agrawal et al. propose a protocol performing private matching without a TTP, building on a previous work by Huberman et al. [14] by using a pair of commutative encryption schemes. Building on this work, Li et al. formalize in [17] the security requirements of private matching and identify the issue of spoofing, which consists in one of the entities claiming elements that it does not own. The issue of spoofing is similar to Subjects cheating in their matching proof (however this issue is not relevant for the Query Issuer). To solve this issue, Li et al. further introduce a Trusted Third Party which provides Data Ownership Certificates (similar to the Authority providing credentials) and propose a modified version of the Agrawal protocol.

A different approach was investigated by Freedman et al. in [12]: they propose a solution derived from secret sharing protocols based on Oblivious Polynomial Evaluation. They also study some variants of private matching, among which the private cardinality matching, which is very close to our matching ratio computation. The solution for the latter is only proposed for semi-honest parties but the case of malicious entities is not considered. Kissner and Song [16] proposed multi-party protocols that apply to several set operations (including set intersection) and that are secure in the presence of honest-but-curious adversaries. They also propose a construction secure in the presence of malicious adversaries based on zero-knowledge proofs. For the same problem, Dachman-Sold et al. propose a more efficient solution in [10].

In [8], Camenisch and Zaverucha introduce the notion of certified sets: a trusted third party provides credentials to users prior to the private set intersection protocol. This trusted third party plays the same role as $\mathcal{A}$ in our solution.

Finally, we note the recent work of De Cristofario and Tsudik, who propose in [9] more efficient protocols to various flavors of private set intersection.

All these protocols cannot readily be applied to our scenario, because they are interactive protocols between two entities (a client and a server) that interact directly (possibly in several rounds), and there is no clear translation of this two-party setting to our problem. The presence of an active Broker indeed introduces different privacy threats while enabling a decoupling between Query Issuer and Subjects. Furthermore, one of the entities in our scenario, namely the Query Issuer, can express queries on any selection criteria and is not limited to a predefined set contrary to P_1 limited to X_1 in classical private matching.

5.2 Oblivious keyword search

Oblivious Keyword Search is a generalization of Oblivious Transfer [21, 4, 7, 13] where the client receives all messages related to a given private keyword instead of requesting a message at a particular position. It was proposed by Ogata and Kurosawa in [20] who showed the relationships between both notions and presented two efficient methods to achieve oblivious keyword search.

Oblivious Keyword Search is relevant to our problem because it can be used to construct private set intersection protocols [12], and more importantly they

can be combined with Public Encryption with Keyword Search (PEKS) to offer additional properties as presented in [6]. The latter scheme, that we refer to as PEOKS, enhances PEKS by introducing the notion of committed blind anonymous identity-based encryption, which allow Subjects $\mathcal{S}^l$ to privately request trapdoors for attributes without revealing the attributes to the Authority $\mathcal{A}$: $\mathcal{S}^l$ commit to their attributes which allows $\mathcal{A}$ to request proofs of statement from users later on. Furthermore, the trapdoors are unique to each subject (even for the same attribute), making the scheme robust and **secure against colluding attackers**. Those properties make PEOKS more suitable to our scenario than PEKS but it is also more difficult to expose briefly and could stray the focus from our contributions and in particular the main novelty of our scheme, which is the introduction of counting Bloom filters and their use in an original way. We keep the advanced version of our scheme based on PEOKS for the extended version of the article.

6 Conclusion

In this paper, we have presented a new private matching protocol which involves an intermediate node that performs some of the matching operations on behalf of the end-entities. Contrary to classical private matching settings, where the client and the server interact directly in the process, in our new scenario the Query Issuer and the Subjects do not interact at all, and do not even need to know each others' identity. The new protocol is based on the combination of searchable encryption mechanisms and counting Bloom filters used in a radically different mindset and allows a third entity, namely the Broker, to correctly compute the matching ratio based on encrypted information only. While introducing this third entity allows a decoupling between the end-entities, it raises new privacy and security issues. We have proved that the proposed protocol preserves the privacy of end-entities thanks to the semantic security of the underlying searchable encryption mechanisms. The security against malicious Subjects cheating on the matching ratio has been analyzed and proved by bounding the probability of the success of the malicious Subject. Finally we have identified an interesting trade-off between security and performance, and we have computed the optimal parameters for an efficient execution of the protocol under a certain security level.

As future work, we plan to implement this mechanism with a PEOKS scheme to mitigate the impact of colluding attackers. We also envision to introduce multiple authorities to reduce the importance and the capabilities of $\mathcal{A}$.

References

1. Agrawal, R., Evfimievski, A., and Srikant, R. Information sharing across private databases. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (2003), ACM, pp. 86–97.
2. Ajmani, S., Morris, R., and Liskov, B. A trusted third-party computation service. Tech. Rep. MIT-LCS-TR-847, MIT, may 2001. www.pmg.csail.mit.edu/ ajmani/papers/tep.pdf.

3. Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G. Public Key Encryption with keyword Search. In *Advances in Cryptology - EUROCRYPT 2004* (2004), Springer Berlin/Heidelberg, pp. 506–522.
4. Brassard, G., Crépeau, C., and Robert, J.-M. All-or-nothing disclosure of secrets. In *Proceedings on Advances in cryptology—CRYPTO '86* (1986), Springer-Verlag, pp. 234–238.
5. Broder, A., and Mitzenmacher, M. Network applications of bloom filters: A survey. In *Internet Mathematics* (2002), pp. 636–646.
6. Camenisch, J., Kohlweiss, M., Rial, A., and Sheedy, C. Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09* (2009), Springer-Verlag, pp. 196–214.
7. Camenisch, J., Neven, G., and Shelat, A. Simulatable adaptive oblivious transfer. In *Proceedings of the 26th annual international conference on Advances in Cryptology* (2007), EUROCRYPT '07, Springer-Verlag, pp. 573–590.
8. Camenisch, J., and Zaverucha, G. M. Private intersection of certified sets. In *Financial Cryptography and Data Security* (2009), Springer-Verlag, pp. 108–127.
9. Cristofaro, E. D., and Tsudik, G. Practical private set intersection protocols with linear complexity. In *Financial Cryptography'10* (2010), pp. 143–159.
10. Dachman-Soled, D., Malkin, T., Raykova, M., and Yung, M. Efficient robust private set intersection. In *Proceedings of the 7th International Conference on Applied Cryptography and Network Security* (2009), ACNS '09, Springer-Verlag, pp. 125–142.
11. Fan, L., Cao, P., Almeida, J., and Broder, A. Z. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking 8*, 3 (2000), 281–293.
12. Freedman, M. J., Nissim, K., and Pinkas, B. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004* (2004), Springer Verlag.
13. Green, M., and Hohenberger, S. Blind identity-based encryption and simulatable oblivious transfer. In *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security* (2007), ASIACRYPT'07, Springer-Verlag, pp. 265–282.
14. Huberman, B. A., Franklin, M., and Hogg, T. Enhancing privacy and trust in electronic communities. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce* (1999), ACM, pp. 78–86.
15. Jefferies, N., Mitchell, C. J., and Walker, M. A proposed architecture for trusted third party services. In *Proceedings of the International Conference on Cryptography: Policy and Algorithms* (1995), Springer-Verlag, pp. 98–104.
16. Kissner, L., and Song, D. Privacy-preserving set operations. In *Proceedings of CRYPTO '05* (August 2005).
17. Li, Y., Tygar, D., and Hellerstein, J. M. Private matching. Tech. Rep. IRB-TR-04-005, Intel Research Laboratory Berkeley, 2004. http://www.eecs.berkeley.edu/~tygar/papers/Private_matching.pdf.
18. Lynn, B. The pairing-based cryptography library, 2006. http://crypto.stanford.edu/pbc/.
19. Menezes, A., Vanstone, S., and Okamoto, T. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing* (1991), pp. 80–89.
20. Ogata, W., and Kurosawa, K. Oblivious keyword search. *Journal of Complexity - Special issue on coding and cryptography 20* (April 2004), 356–371.

21. Rabin, M. O. How to exchange secrets with oblivious transfer, 1981. Harvard University Technical Report.

7 Appendix: proofs and example

7.1 Proof of theorem 1

Theorem. *Let $x_1, ..., x_n$ be n elements randomly chosen from a group G of order q. Let CBF be a counting Bloom filter of size ϕ in which the n elements $x_1, ..., x_n$ were inserted using u hash functions $h_1, ..., h_u$. Then, there are more than $\frac{q}{\phi^u}$ possible sets of elements of G^n leading to the same counting Bloom filter:*

$$|\{(x'_1, ..., x'_n) \in G^n | \mathtt{CBF}(x'_1, ..., x'_n) = \mathtt{CBF}(x_1, ..., x_n)\}| > \frac{q}{\phi^u}$$

Proof. Let us examine the simplest case of $n = 1$ and $CBF = \mathtt{CBF}(x_1)$. In that case the positions $h_1(x_1); ...; h_u(x_1)$ are incremented in CBF. The security argument is based on two main observations:

- The first observation is that the hash functions $h_1, ..., h_u$ are not invertible, even though they are not necessarily cryptographic hash functions. Indeed, these functions map elements of G (a group of order q) to a small set (the integers smaller than ϕ). Therefore, if the hash functions have a uniformly distributed output then each output has $\frac{q}{\phi}$ preimages. If we combine the u equations corresponding to the u hash functions, the number of inputs simultaneously verifying u conditions on their digests is $\frac{q}{\phi^u}$.
- The second observation is that there is an information loss in the construction of this structure: the order of the hash functions is lost once the element is inserted in the counting Bloom filter, and it is impossible to know which hash function resulted in the incrementation of a given position in the filter. This second fact further increases the size of the potential preimages by a factor of up to $u!$: it is possible to set many sets of equations for the same counting Bloom filter.

As a result, the set of possible preimages corresponding to a counting Bloom filter containing a single element is at least $\frac{q}{\phi^u}$. This set is even larger when considering several elements. □

7.2 Proof of theorem 2

Theorem. *Let Q'_i be an encoded query concerning n selection criteria. Let CBF_i be the corresponding matching reference.*

The probability of success $\mathcal{P}_{adv}[\rho_{i,l} \to \rho_{i,l}^{claim}]$ of an adversary $\mathcal{S}^l$ in generating an array $CBF_{i,l}^{claim}$ which is accepted by $\mathcal{B}$ and results in an increase of the matching ratio from $\rho_{i,l}$ to $\rho_{i,l}^{claim}$ is upperly bounded by:

$$\mathcal{P}_{adv}[\rho_{i,l} \to \rho_{i,l}^{claim}] \leq \left(1 - e^{-\frac{(1-\rho_{i,l})n \cdot u}{\phi}}\right)^{(\rho_{i,l}^{claim} - \rho_{i,l})n \cdot u}$$

Proof. We first observe that $\mathcal{S}^l$ cannot know whether the first property (that is $CBF_{i,l}^{claim} \prec CBF_i$) is met or not as $\mathcal{S}^l$ does not know CBF_i. $\mathcal{S}^l$ can only make guesses based on the general parameters of CBF_i. We thus first establish a probabilistic model of counting Bloom filters in order to evaluate the probability of having the three aforementioned properties validated without the knowledge of CBF_i.

We consider a counting Bloom filter CBF of length ϕ containing n unknown elements which were inserted using u hash functions. Given that the probability distribution of the values in CBF_i follows a binomial distribution at each position, the probability $\mathcal{P}'(i_2)$ that the value $CBF[i_1]$ at position i_1 is greater than a given i_2 can be computed as follows: $\forall 0 \leq i_1 \leq \phi - 1, \forall 1 \leq i_2 \leq n \cdot u$,

$$\mathcal{P}'(i_2) = \mathcal{P}[CBF[i_1] \geq i_2] = 1 - \sum_{i_3=0}^{i_2-1} \binom{n \cdot u}{i_3} \left(1 - \frac{1}{\phi}\right)^{n \cdot u - i_3} \left(\frac{1}{\phi}\right)^{i_3}.$$

Based on this result, we then prove by induction[4] that the probability $\mathcal{P}'(i_2)$ decreases faster than a geometric series of ratio $\mathcal{P}'(1)$, or to be more precise that, for $1 \leq i_2 \leq n \cdot u$,

$$\mathcal{P}'(i_2) \leq (\mathcal{P}'(1))^{i_2} \tag{1}$$

assuming that $n \cdot u \leq \phi - 1$.

We then consider ARR to be an array of size ϕ (the matching proof). The probability $\mathcal{P}[ARR \prec CBF]$ that ARR is smaller than CBF can be computed as follows:

$$\mathcal{P}[ARR \prec CBF] = \prod_{i_1=0}^{\phi-1} \mathcal{P}'(ARR[i_1]).$$

Following the result in inequation 1, this probability can be upperly bounded as follows:

$$\mathcal{P}[ARR \prec CBF] \leq \prod_{i_1=0}^{\phi-1} \mathcal{P}'(1)^{ARR[i_1]}$$

Finally, based on the approximation of the Taylor series development of $\mathcal{P}'(1)$ we obtain the following upper bound:

$$\mathcal{P}[ARR \prec CBF] \leq \left(1 - e^{-\frac{n \cdot u}{\phi}}\right)^{w_{ARR}} \tag{2}$$

The last step of the demonstration consists in applying this important result to the matching reference CBF_i and the matching proof $CBF_{i,l}$ where the parameters CBF and ARR are replaced by the challenging reference counting Bloom filter CBF_i and the malicious matching proof $CBF_{i,l}$, respectively. However, this modification is not straightforward because while CBF was assumed to contain n random elements, a malicious Subject $\mathcal{S}^l$ knows some of the elements, that are the ones corresponding to the selection criteria that $\mathcal{S}^l$ matches.

[4] The (long) details of this proof are not included due to page constraints

Thus, the following modifications have to be performed to evaluate the probability $\mathcal{P}_{adv}[\rho_{i,l} \to \rho_{i,l}^{claim}]$ of success of a Subject in increasing its matching ratio from $\rho_{i,l}$ to $\rho_{i,l}^{claim}$:

- We first define by $CBF_i^{chal} = CBF_i - CBF_{i,l}$ the challenging reference counting Bloom filter, that is the part of the counting Bloom filter unknown to $\mathcal{S}^l$. The weight of CBF_i^{chal} is $w_{CBF_i^{chal}} = n(1-\rho_{i,l}) \cdot u$
- Moreover, $CBF_{i,l}^{mal} = CBF_{i,l}^{claim} - CBF_{i,l}$ defines the part of the matching proof which is malicious which weight is denoted by $w_{CBF_{i,l}^{mal}}$ which is computed as follows: $w_{CBF_{i,l}^{mal}} = w_{CBF_{i,l}^{claim}} - w_{CBF_{i,l}} = (\rho_{i,l}^{claim} - \rho_{i,l})n \cdot u$

We therefore obtain the following inequality:

$$\mathcal{P}[CBF_{i,l}^{mal} \prec CBF_i^{chal}] \leq (1 - e^{\frac{n(1-\rho_{i,l})\cdot u}{\phi}})^{(\rho_{i,l}^{claim} - \rho_{i,l})n \cdot u} \tag{3}$$

which corresponds to $\mathcal{P}_{adv}[\rho_{i,l} \to \rho_{i,l}^{claim}]$ if $\rho_{i,l}^{claim} - \rho_{i,l}$ is a multiple of $\frac{1}{n}$ (if $w_{CBF_{i,l}^{mal}}$ is a multiple of u) to satisfy the second of the aforementioned conditions (otherwise the claimed counting Bloom filter would be rejected). □

7.3 Typical figures

To illustrate the performance of the global solution more concretely, we provide some figures of a typical scenario.

First of all, the maximum number of selection criteria that can be used in each query should be reasonably small as it directly leads to an increase in the communication and computation complexity. We therefore set this maximum number to $n_{max} = 20$.

The level of security in groups over elliptic curves depends on a security parameter called the MOV degree [19]: by carefully choosing the elliptic curve it is possible to adjust the trade-off between key size and computation time, while maintaining a given level of security. We choose a curve with a small MOV degree of 2 and a group of order q of 512 bits length to have a security equivalent to 1024 bits RSA.

The size of an encoded query is then less than $2q \cdot n_{max} \approx 20$ Kbits. To put this size into perspective, note that in the case where there is no privacy protection (where queries and replies are sent in clear) and where each selection criteria is stored in a string with 16 8bits-characters, the size of queries is approximately 2.5 Kbits. The size of encoded queries is therefore 8 times larger than their queries in clear, but this is a deliberate choice to optimize the computation performance. If the communication overhead is considered as more important, it is possible to use curves with a higher MOV degree of 6: in that case it is possible to consider groups of smaller order and the overhead would be reduced to 2.5 times.

Concerning the parameters of counting Bloom filters, in addition to n_{max}, we need to define ϕ and u.

First of all, u is used as a security parameter, since the probability of success of an adversary can be bounded by 2^{-u}. As explained in section 4.2, it is not necessary to choose a very high value for u as it does not lead to revealing a secret but only to being able to cheat on the matching ratio. By choosing $u = 10$ for example, the probability of success of an attacker would still be bounded by 10^{-3} in the most favorable case. Other probabilities of success are presented in table 1. This table shows that the probability of success for significant attacks is very low (for reference the typical security margin for symmetric encryption is $2^{-80} \approx 10^{-24}$). It is of course possible to choose a higher value for u to make sure that even in the most favorable case the attacker would not succeed with probability more than 2^{-80} but u impacts first on the construction of counting Bloom filter (each element requires the computation of u hash values) and second and more importantly on the size of counting Bloom filters. We therefore believe that choosing a smaller value for u (as we did) is a better trade-off.

Table 1. Probability $\mathcal{P}_{adv}[\rho_{i,l} \rightarrow \rho_{i,l}^{claim}]$ of an adversary $\mathcal{S}^l$ with legitimate matching ratio $\rho_{i,l}$ to successfully claim a matching ratio of $\rho_{i,l}^{claim}$ with an encoded query Q_i' containing n selection criteria. The general parameters used for the counting Bloom filter are $n_{max} = 20$, $u = 10$, and $\phi = 289$.

n \ $\mathcal{P}_{adv}$	$0 \rightarrow \frac{1}{n}$	$0 \rightarrow \frac{2}{n}$	$0 \rightarrow \frac{1}{2}$	$0 \rightarrow 1$	$\frac{1}{2} \rightarrow \frac{1}{n} + \frac{1}{2}$	$\frac{1}{2} \rightarrow 1$	$1 - \frac{1}{n} \rightarrow 1$
6	5.10^{-8}	3.10^{-15}	1.10^{-22}	2.10^{-44}	9.10^{-11}	7.10^{-31}	2.10^{-15}
10	5.10^{-6}	2.10^{-11}	2.10^{-27}	4.10^{-54}	1.10^{-8}	1.10^{-40}	2.10^{-15}
20	1.10^{-3}	9.10^{-7}	7.10^{-31}	5.10^{-61}	5.10^{-6}	4.10^{-54}	2.10^{-15}

The number of positions ϕ of the counting Bloom filter according to the strategy explained in section 4.2 should be $\phi = \left\lceil \frac{n_{max} \cdot u}{\ln(2)} \right\rceil$ which is equal to 289 when $n_{max} = 20$ and $u = 10$. We choose to allocate 4 bits for each position in the counting Bloom filter, thus the total size of the filter is slightly more than 1 Kbit while the probability of a bucket overflow to happen would be less than 2.10^{-12}. The size of the counting Bloom filters is therefore really negligible in comparison with the size of the queries, thus the use of counting Bloom filters really offers a decisive advantage from a performance perspective on top of the advantage from a privacy point of view.

On this matter, we mentioned in section 4.1 that the size of the set of possible preimages that lead to a counting Bloom filter is around $\frac{q}{\phi^u} \approx 2^{448}$. This proves that a brute-force attack to break the privacy-preserving properties of the computation assurance solution is out of reach of current computing power.

PRISM – Privacy-Preserving Search in MapReduce

Erik-Oliver Blass[1], Roberto Di Pietro[2], Refik Molva[3], and Melek Önen[3]

[1] Northeastern University, Boston, MA
[2] Università di Roma Tre, Rome, Italy
[3] EURECOM, Sophia Antipolis, France

Abstract. We present PRISM, a privacy-preserving scheme for word search in cloud computing. In the face of a curious cloud provider, the main challenge is to design a scheme that achieves privacy while preserving the efficiency of cloud computing. Solutions from related research, like encrypted keyword search or Private Information Retrieval (PIR), fall short of meeting real-world cloud requirements and are impractical. PRISM's idea is to transform the problem of word search into a set of parallel instances of PIR on small datasets. Each PIR instance on a small dataset is efficiently solved by a node in the cloud during the "Map" phase of MapReduce. Outcomes of map computations are then aggregated during the "Reduce" phase. Due to the linearity of PRISM, the simple aggregation of map results yields the final output of the word search operation. We have implemented PRISM on Hadoop MapReduce and evaluated its efficiency using real-world DNS logs. PRISM's overhead over non-private search is only 11%. Thus, PRISM offers privacy-preserving search that meets cloud computing efficiency requirements. Moreover, PRISM is compatible with standard MapReduce, not requiring any change to the interface or infrastructure.

1 Introduction

Today, users take advantage of public clouds operated by large companies like Google or Amazon. Instead of setting up and maintaining their own data centers, users reduce their costs by outsourcing both storage and processing to a cloud. One prominent example allowing cloud-based storage and processing is Hadoop MapReduce [3], a variant of Google's MapReduce system [17]. Hadoop MapReduce is widely used, and public MapReduce clouds are offered by companies such as Amazon [2, 25].

The advantages of cloud computing unfortunately come with a high cost in terms of new security and privacy exposures. Apart from classical security challenges of shared services, outsourcing of data storage and processing raises new challenges in the face of potentially malicious cloud providers. *Privacy* of outsourced data appears to be a major requirement in this context. Some regulations are already provisioned as to the privacy protection of outsourced governmental documents [11, 12, 18]: these regulations usually aim at assuring privacy against *curious clouds* or against clouds with data centers located in "rogue" countries or with insufficient security guarantees; they also are defined to avoid data leakage in case of operational failures in the cloud. Along these lines, there is also a raising corporate concern about the privacy of sensitive business data stored in the cloud [14]. Although cloud providers thrive to meet the increased privacy

S. Fischer-Hübner and M. Wright (Eds.): PETS 2012, LNCS 7384, pp. 180–200, 2012.

demand by certifying their services [24], malicious insiders have still been identified as one of the top threats in cloud computing [15].

While encryption of outsourced data by the users seems to be a viable protection against most privacy problems, traditional data encryption does not suit the requirements of cloud computing: the cloud not only serves as high capacity memory, but is also involved in data processing such as statistical data analysis, log analysis, indexing, data mining, and searching [25]. However, data processing performed by the cloud would not be feasible or would be *inefficient* with encrypted data.

Among data processing primitives, word search, i.e., verifying, whether a certain word is part of a dataset, is not only one of the most fundamental operations, but surprisingly also one of the most demanded applications in, e.g., MapReduce cloud computing [25]. **Related work** on search in encrypted data, e.g., Boneh et al. [6], Ogata and Kurosawa [31], falls short of meeting cloud computing privacy and performance requirements. These techniques are *impractical* as they are designed for centralized execution models that are incompatible with today's highly parallel cloud architectures.

In this paper, we present PRISM, a new scheme for *privacy-preserving and efficient word search* for MapReduce clouds. PRISM pursues two specific objectives: 1.) privacy against potentially malicious cloud providers and 2.) high efficiency through the integration of security mechanisms with the operations performed in the cloud.

In order to achieve efficiency, PRISM takes advantage of the inherent parallelization akin to cloud computing: the word search problem on a very large encrypted dataset is partitioned into several instances of word search in small datasets that are executed in parallel ("Map" phase). The individual word search operations performed in the cloud yield a result amenable to straightforward *aggregation* in the ultimate phase ("Reduce" phase) of the word search operation. The word search operation builds on a Private Information Retrieval (PIR) [29] technique which is extended in order to generate intermediate search results that are still encrypted and that can be *combined* through linear operations to yield the global result of the word search over the entire dataset.

Summarizing our contributions, PRISM:

• **is suited to cloud computing:** PRISM is the first privacy-preserving search scheme suited to cloud computing; it brings together storage and search privacy with high performance by leveraging the efficiency of the MapReduce paradigm. PRISM is parallelizable and also allows efficient combination of individual results. Its efficiency has been evaluated through searching in DNS logs provided by an Internet Service Provider. Although PRISM's overhead within the core Map function is large compared to non-privacy-preserving search (factor of 9), the total system overhead is only 11%.

• **preserves privacy in the face of potentially malicious cloud providers:** PRISM allows carrying out these critical operations in the cloud without trusting the cloud.

• **is compatible to standard MapReduce:** PRISM only requires a standard MapReduce interface without modifications in the underlying system. PRISM can thus be integrated on any cloud that provides a standard MapReduce interface such as Amazon.

• **provides flexible search:** In contrast to traditional encrypted keyword search techniques, PRISM is not limited to searching for a fixed set of *predetermined* keywords to be known in advance, but offers flexible search for any words.

2 Problem Statement and Adversary Model

Throughout this paper, we will use an application example to motivate our work. Inspired by recent events [30], we envision a *data retention* scenario. Due to regulatory matters, a small, residential Internet Service Provider (ISP) must retain logs of client accesses. Due to the sheer amount of data to retain, the ISP outsources logfiles to the cloud. Files are encrypted as they contain sensitive data. Still, e.g., law enforcement authorities will contact the ISP to search for words (strings, text, ...) in outsourced files.

More concretely, assume service provider $\mathcal{U}$ (the cloud "user") providing DNS services to clients. $\mathcal{U}$ logs each client's access, i.e., $\mathcal{U}$ logs the tuple (timestamp, client ID, hostname queried). Due to the large amount of log data and cost reasons, $\mathcal{U}$ outsources its logfiles into a cloud. Regularly, say each day i, $\mathcal{U}$ creates a new logfile L_i. At the end of a longer period, $\mathcal{U}$ wants to (or is forced to) find out, whether there was an interest in a suspicious host w. So, $\mathcal{U}$ checks, at which day, i.e., in which logfiles L_i, word w occurs. $\mathcal{U}$ queries the cloud for w, and the cloud responses with an answer R telling $\mathcal{U}$ which of the L_i contain(s) w.

Note that $\mathcal{U}$ does not know in advance which word w it has to search for. This automatically disqualifies protocols for *predefined* keyword search, such as PEKS [6] and derivatives. Also, data retention regulations require outsourced data to be fully recoverable; storing only digests of data in the cloud, e.g., hash values, is insufficient.

The cloud is assumed to be untrusted, more precisely *semi-honest* ("honest-but-curious"). Regulatory matters imply that the cloud must not learn any information about the content it hosts and search queries performed. This implies both, the encryption of data by $\mathcal{U}$ before outsourcing it to the cloud and "obliviously" processing queries on encrypted data by the cloud.

Before we formalize our privacy requirements, we first define the main components for a cloud word search scheme.

Definition 1 (Cloud Word Search). *Let $\mathcal{L}$ denote a sequence of files $\mathcal{L} := \{L_1, \ldots, L_m\}$ and Σ the set of possible words. Each file L_i consists of a sequence of words $L_i := \{w_{(i,1)}, w_{(i,2)}, \ldots, w_{(i,|L_i|)}\}, w_{(i,j)} \in \Sigma$.*

A cloud word search scheme comprises the following algorithms:

1. $KeyGen(s)$*: using a security parameter s, this algorithm outputs a secret $\mathcal{S}$.*
2. $Encrypt(\mathcal{S}, \mathcal{L})$*: uses the secret $\mathcal{S}$ to encrypt the content of files $L_i \in \mathcal{L}$ and outputs the set of resulting encryptions $\mathcal{E} := \{E_{L_1}, \ldots, E_{L_m}\}$. Here, E_{L_i} denotes the encryption of file L_i.*
3. $Upload(\mathcal{E})$*: uploads $\mathcal{E}$ to the cloud.*
4. $PrepareQuery(\mathcal{S}, w)$*: takes $\mathcal{S}$, the word w to search for, and produces a query Q.*
5. $Process(\mathcal{E}, Q)$*: with encryptions $\mathcal{E}$ and query Q, this algorithm produces result R.*
6. $Decode(\mathcal{S}, R, w)$*: taking result R, secret $\mathcal{S}$, and word w, this algorithm outputs the set of indices $\mathcal{I} := \{i_1, \ldots, i_r\}$ such that $\forall i \in \mathcal{I} : L_i \in \mathcal{L} \wedge w \in L_i$, if $R = Process(\mathcal{E}, Q)$ with $Q = PrepareQuery(\mathcal{S}, w)$, and $\mathcal{E} = Encrypt(\mathcal{S}, \mathcal{L})$.*

The basic interaction between user $\mathcal{U}$ and the cloud can be summarized as follows: first, user $\mathcal{U}$ encrypts and uploads files. Then, $\mathcal{U}$ prepares a search query Q for word w and sends Q to the cloud. The cloud processes this query using algorithm $Process$

and produces output R. This output is sent back to $\mathcal{U}$. Using output R and another algorithm $Decode$, $\mathcal{U}$ can compute the list of files containing w. While describing PRISM's details later in Section 4, we will show how they map to these algorithms.

Note that, in a cloud setting, $\mathcal{U}$ executes $KeyGen$, $Encrypt$, $Upload$, $PrepareQuery$, and $Decode$, while the cloud executes $Process$. The idea is that algorithms $KeyGen$, $Encrypt$, $Upload$, $PrepareQuery$, and $Decode$ are computationally very "lightweight" for $\mathcal{U}$ compared to $Process$. The main computational burden lies on the cloud side.

Privacy Requirements

Intuitively, our application demands for two main types of privacy. The cloud (now called "adversary $\mathcal{A}$") must neither be able to infer any details about stored files nor learn details about $\mathcal{U}$'s queries and results delivered back to $\mathcal{U}$. This implies not only the secrecy or confidentiality of the content, but also the inability to compute statistics on the content. Informally, in our setting:

- given $\mathcal{E}$, $\mathcal{A}$ must not learn the content of $\mathcal{L}$ and must not discover whether files L_i contain a word w, e.g., multiple times;
- given a set of queries $\{Q_i\}$, $\mathcal{A}$ must not learn the words $\{w_i\}$ $\mathcal{U}$ is looking for and must not discover whether the same word is queried multiple times;
- given the result R_i of a query Q_i, $\mathcal{U}$ must not learn which file(s) contain the word corresponding to this specific query W_i.

Instead, the adversary should only learn "trivial" properties, such as the total number of files, the file size, and the total number of queries. Along the same lines as traditional indistinguishability [23], $\mathcal{A}$ should not be able to infer any additional information from encrypted files, queries, and results. We formally define privacy for a cloud word search scheme using a game between adversary $\mathcal{A}$ (the cloud) and a challenger (user $\mathcal{U}$).

Definition 2 (Privacy). *Let $\mathcal{W}$ denote a sequence of words $\mathcal{W} := \{w_1, \ldots, w_n\}$. The game* GAME *is played as follows.*

1. *The challenger executes $KeyGen(s)$ to derive secret $\mathcal{S}$.*
2. *$\mathcal{A}$ selects a distinct pair of sequences of files and words $(\mathcal{L}_0, \mathcal{W}_0)$ and $(\mathcal{L}_1, \mathcal{W}_1)$, where $|\mathcal{L}_0| = |\mathcal{L}_1|$, $\forall(L_i^0 \in \mathcal{L}_0, L_i^1 \in \mathcal{L}_1) : |L_i^0| = |L_i^1|$, and $|\mathcal{W}_0| = |\mathcal{W}_1|$.*
   $\mathcal{A}$ sends $(\mathcal{L}_0, \mathcal{W}_0)$ and $(\mathcal{L}_1, \mathcal{W}_1)$ to the challenger.
3. *The challenger randomly selects $b \in \{0, 1\}$ and*
   - *executes $Encrypt(\mathcal{S}, \mathcal{L}_b)$, i.e., the challenger computes encrypted files $\mathcal{E}_b = \{E(\mathcal{S}, L_i) | L_i \in \mathcal{L}_b\}$.*
   - *executes $Upload(\mathcal{E}_b)$ to send encrypted files back to $\mathcal{A}$.*
   - *executes $PrepareQuery(\mathcal{S}, w)$ for each w in $\mathcal{W}_b$. This results in the sequence of queries $\mathcal{Q}_b := \{Q_1, \ldots, Q_{|\mathcal{W}_b|}\}$ that the challenger also sends to $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs $b' \in \{0, 1\}$. The outcome of* GAME *is "1" iff $b' = b$.*

A cloud word search scheme is called privacy-preserving iff

$$Pr(\text{GAME}(\mathcal{A}) = 1) \leq \frac{1}{2} + \epsilon(s)$$

for all probabilistic polynomial-time adversaries $\mathcal{A}$. Here, $\epsilon(s)$ is a negligible function, $\epsilon(s) < \frac{1}{P(s)}$ for every polynomial P with sufficiently large security parameter s.

The specification of the $(\mathcal{L}_i, \mathcal{W}_i)$ in step 2. of Definition 2 reflects the fact that $\mathcal{A}$ can learn the total number of files, the size of each file, and the number of queries.

Limitations: We consider semi-honest clouds. Fully malicious clouds might perform DoS-attacks or deviate from protocol execution. Similar to "reaction attacks" [26], the cloud might return garbage to $\mathcal{U}$, to observe $\mathcal{U}$'s reaction (e.g., sending the same query). Although realistic, we leave such attacks for future work. Also, our privacy definition does not capture trivial privacy properties, e.g., the size of outsourced files. Mitigation strategies (e.g., padding files) might be contradictory to cloud efficiency. We conjecture that, for many applications, losing "trivial" privacy properties is acceptable.

3 Background

3.1 MapReduce

We target a system suited for the MapReduce [3] paradigm. We will now give a condensed overview of MapReduce, focusing on aspects necessary to understand PRISM.

Upload. A MapReduce cloud comprises a set of "slave" node computers and a "master" computer. While $\mathcal{U}$ uploads files into the MapReduce cloud, each file is automatically split into blocks called *InputSplits*. InputSplits have a fixed size $S_{\text{InputSplit}}$ which is a pre-configured system parameter. If S_{File} denotes the size of an uploaded file, the number of InputSplits c computes to $c = \frac{S_{\text{File}}}{S_{\text{InputSplit}}}$. For each InputSplit, a workload sharing algorithm selects a slave node and places the InputSplit on it.

In addition to data, the MapReduce also allows $\mathcal{U}$ to upload "operations", i.e., compiled Java classes. These classes represent the implementation of three functions.
1.) $Scan(\text{INPUTSPLIT}) \rightarrow [(k, v)])$, a functions that takes an InputSplit as an input, parses it, i.e., scans it and generates a set of key-value pairs $[(k, v)]$ out of it.
2.) $Map(k, v) \rightarrow [(k', v')]$, a function that takes as an input a single key-value pair (k, v) and outputs a set of "intermediate" key-value pairs $[(k', v')]$.
3.) $Reduce([(k', v')]) \rightarrow \text{FILE}$, a function that takes as an input a set of intermediate key-value pairs $[(k', v')]$ and writes arbitrary output into a file.

Uploaded Java classes are sent to all slave nodes storing an InputSplit.

Map Phase. After data and implementations have been uploaded, $\mathcal{U}$ specifies one uploaded file and triggers MapReduce operations on that file. The first phase of operation is the "Map" phase. Each slave node becomes a "mapper" node. Each mapper executes $\mathcal{U}$'s $Scan$ function on the InputSplit it stores locally. This generates a set of key-value pairs on each mapper. Furthermore, the mapper node executes $\mathcal{U}$'s Map function on this generated key-value pairs to produce a set of intermediate key-value pairs.

Reduce Phase. MapReduce starts the "Reduce" phase. Slave nodes are scheduled to become "reducers". For each of the intermediate pairs (k', v'), MapReduce selects a reducer and sends (k', v') to this reducer. MapReduce selects the *same* reducer for all pairs (k', v') having the same key. Each reducer executes $\mathcal{U}$'s reduce function on its set of intermediate key-value pairs and writes the output to a file. This file is sent to $\mathcal{U}$.

3.2 Trapdoor Group Private Information Retrieval

PIR allows a user to retrieve data from a server without revealing which data is retrieved. For PRISM, we make use of a simple and efficient PIR mechanism as previously suggested by Trostle and Parrish [37]. As this mechanism is just a building block for PRISM, we will only give a summary of its mode of operation and rationale.

Overview: Matrix $\mathcal{M}$ is a $t \times t$ matrix of elements in $\mathbb{Z}_N$ stored at a server. For example, $N = 2$ for a binary matrix. User $\mathcal{U}$ is only interested in receiving elements of the k^{th} row in $\mathcal{M}$, but the server must not learn k. The idea is now that $\mathcal{U}$ sends two "types" of values to the server. For each row that $\mathcal{U}$ is not interested in, he sends a value of the "first" type. For the one row that $\mathcal{U}$ is interested in, he sends a single value of the "second" type. To prevent the server from distinguishing between the two types of values, $\mathcal{U}$ blinds each value with a blinding factor b. This blinding factor can later be removed by $\mathcal{U}$. The server now performs simple additions with received values and elements stored in $\mathcal{M}$. The result is sent back to $\mathcal{U}$ who removes the blinding factor and determines the values of the row of his interest.

Preparation: Assume $\mathcal{U}$ is interested in row k. $\mathcal{U}$ chooses a group $\mathbb{Z}_p$, with a prime p of m bits. $\mathcal{U}$ also chooses a random $b \in \mathbb{Z}_p$ and t random values $a_i \in \mathbb{Z}_p$. Therewith, $\mathcal{U}$ computes t values $e_i < \frac{p}{t \cdot (N-1)}$ such that: $e_k := 1 + a_k \cdot N$ **and** $\forall i \neq k : e_i := a_i \cdot N$.

Finally, $\mathcal{U}$ computes $\alpha_i := b \cdot e_i \mod p$ and sends the α_i to the server. Other values $(p, m, b, \{e_i\}, \{a_i\})$ remain secret. The server treats α_i as large integers and performs the following integer operations, i.e., without any modulo.

Server computation: Let $\boldsymbol{u}$ be the vector $\boldsymbol{u} := (\alpha_1, \ldots, \alpha_t)$. The server computes the matrix product $\boldsymbol{v}$ and sends it back to $\mathcal{U}$,

$$\boldsymbol{v} := (\beta_1, \ldots, \beta_t) = \boldsymbol{u} \cdot \mathcal{M} = (\sum_{i:=1}^{t} \alpha_i \cdot \mathcal{M}_{i,1}, \ldots, \sum_{i:=1}^{t} \alpha_i \cdot \mathcal{M}_{i,t}).$$

Result analysis: Upon receipt, in order to "un-blind" values, $\mathcal{U}$ computes the t inverse values $z_i := \beta_i \cdot b^{-1} \mod p$. Now, U can conclude that $z_i \mod N$ equals the i^{th} element of the k^{th} row in $\mathcal{M}$. Therewith, $\mathcal{U}$ has retrieved the t elements of the k^{th} row of $\mathcal{M}$ in a privacy-preserving fashion. Note the linearity for two β_i and β_i' received during different PIR runs: $\beta_i \cdot b^{-1} + \beta_i' \cdot b^{-1} = (\beta_i + \beta_i') \cdot b^{-1} \mod p$. That is, the sum of two individually un-blinded vectors equals un-blinding the sum of two received vectors $\boldsymbol{v}, \boldsymbol{v'}$. We will later use this linearity during PRISM's reduce phase.

Security Rationale: Security and privacy of this protocol are based on the trapdoor group assumption. With only knowledge of α_i, but not secret trapdoor p, it is computationally hard for the server to infer any information about low order bits, i.e., the modulo of z or e_i, cf., Trostle and Parrish [37].

Discussion: Again, we stress that this particular PIR scheme is an exchangeable building block. In general, any of the "traditional" PIR techniques based on group homomorphic encryption [29, 33] is suited for use within PRISM. We have chosen Trostle and Parrish [37] only due to the straightforward way to implement it (Section 6). Other PIR schemes might reduce the (already small) overhead, but this is out of scope here.

4 PRISM Protocol

PRISM comprises three parts: *upload* of data into the cloud, the MapReduce *search*, and the *result analysis* where the user decides whether the word has been found. We will briefly give an overview about each part.

1.) Upload. During upload, $\mathcal{U}$ encrypts each word (of a logfile) using symmetric encryption. Ciphertexts are stored in a file, and this file is sent to the MapReduce cloud. The cloud automatically splits large files and distributes splits (*InputSplits*) among *mapper* nodes. We use a standard blockcipher (AES) to perform ciphering of words. However, to ensure privacy as of Definition 2, plaintext is modified before encryption using a "stateful cipher" construction. Therewith, $\mathcal{U}$ can still search for some word w, but the cloud cannot compute statistics about ciphertexts.
2.) Search. Eventually, $\mathcal{U}$ wants to search his encrypted files for some word w. Therefore, $\mathcal{U}$ sends implementations of "algorithms" for the map and reduce phases to the MapReduce cloud, and the cloud executes these on uploaded data. For example, $\mathcal{U}$ sends Java ".class" files for the mappers and Java ".class" files for *reducer* nodes. MapReduce distributes these implementations to each mapper and reducer, respectively. PRISM's rationale is to *transform* the word search problem into a set of small PIR instances. To do so, each mapper, scanning through its locally stored InputSplit, creates a binary matrix. Ciphertexts in the InputSplit are assigned to individual elements in that matrix. If a ciphertext is present in an InputSplit, its corresponding element in the matrix is set to either "0" or "1". Using private information retrieval techniques, PRISM can extract the value of a single element in the matrix with the mapper being totally oblivious to which element is extracted. Consequently, $\mathcal{U}$ can specify which element to extract in a privacy-preserving way. All mappers send their obliviously extracted elements as key-value pairs to reducers. Reducers simply sum up received values and return sums to $\mathcal{U}$. Therewith, neither mappers nor reducers can learn any information about which ciphertext $\mathcal{U}$ was interested in.
3.) Result analysis. Finally, $\mathcal{U}$ receives an encrypted sum for each of the originally uploaded files from reducers. $\mathcal{U}$ can decrypt them and decide which of the files contain w. However, due to the probability of "collisions" in matrices, i.e., two different ciphertexts can be assigned to the same element, and due to ambiguities of received sums, $\mathcal{U}$'s decision whether w is inside some file might be wrong. Therefore, PRISM repeats the above process in a total of q so called "rounds". In each of the rounds, a new matrix is generated, elements are set to "1" or "0" depending on the round number, and results are returned as described. This reduces the probability of $\mathcal{U}$ making incorrect decisions.

Initialization: Before the actual uploading, initially, and only once, $\mathcal{U}$ has to execute $KeyGen$. In PRISM, $KeyGen$ outputs secret $\mathcal{S} := \{K, N, p\}$, where $|K|, |N|, |p|$ are

Input: words w_i
Output: ciphertexts C_i uploaded to cloud

```
1 Initialize all γ to 0;
2 foreach word w_i do
3 |   γ_{w_i} := get (w_i);  //from hash table
4 |   γ_{w_i} := γ_{w_i} + 1;
5 |   insert (w_i, γ_{w_i});  //into hash table
6 |   C_i := E_{K_d}(w_i, γ_{w_i});
7 |   upload C_i ;
8 end
```

Algorithm 1. "Stateful Cipher" example and upload to MapReduce

specified by security parameter s. K is a symmetric key, and N and p are Trapdoor Group PIR parameters as presented in Section 3.2.

4.1 Upload

Overview. In our scenario, cloud user $\mathcal{U}$ continuously logs customer access and sends logfiles to the cloud. Each day, $\mathcal{U}$ starts using a new logfile. For simplicity, we assume that entries logged by $\mathcal{U}$ are simple words. Each logfile is encrypted word by word using a "stateful cipher" E_K, and resulting ciphertexts are written to a file, respectively. The encrypted files are sent to the cloud.

Definition 3 (Stateful Cipher). *Given standard symmetric encryption E_K with key K, e.g., AES, we extend E to a* stateful cipher *by adding "counters" γ_{w_i} that count the history of inputs w_i. Each time E encrypts w_i, counter γ_{w_i} is increased by one.*

In conclusion, a stateful cipher is a cipher that knows how often it has encrypted a specific plaintext. The following presents one trivial stateful cipher construction used in PRISM to encrypt before uploading.

Stateful Cipher Example (see Algorithm 1): For simplicity, user $\mathcal{U}$ uses a secret key K to derive a different key for each day d, e.g., $K_d := \mathrm{HMAC}_K(d)$.

For each day, $\mathcal{U}$ maintains a hash table containing the list of counters γ_{w_i} in $\mathcal{U}$'s local storage. At the beginning of each day, $\mathcal{U}$ initializes all counters to 0, i.e., $\gamma_{w_i} = 0$. Now, for each logentry w_i that should be stored in the cloud, $\mathcal{U}$ computes γ_{w_i} and increases γ_{w_i} by 1. Then, $\mathcal{U}$ computes ciphertext $C_i := E_{K_d}(w_i, \gamma_{w_i})$. User $\mathcal{U}$ sends ciphertext C_i to the cloud that stores it in this day's file. For the (AES) encryption $E_{K_d}(w_i, \gamma_{w_i})$, "," denotes an unambiguous pairing of inputs. We discuss the reason for using a "stateful cipher" over using, e.g., a CBC mode of encryption in Section 5.1.

Summarizing, with respect to Definition 1, $Encrypt$ in PRISM takes K to derive a separate key K_d for each file to be encrypted. Actual encryption of each file is performed word by word using the stateful cipher and key K_d, so $\mathcal{E} := \{E_{L_1}, \ldots, E_{L_m}\}$ where $E_{L_i} := \{E_{K_i}(w_1, \gamma_{w_1}), \ldots, E_{K_i}(w_{|L_i|}, \gamma_{w_{|L_i|}})\}$. $Upload$ in PRISM can be regarded as simply sending the encrypted files $\mathcal{E}$ to MapReduce.

4.2 Search

User $\mathcal{U}$ wants to search a set of files for word w within a period of time. For ease of understanding, we will restrict our description below to PRISM working on a single file

specified by the user, i.e., the file of day d. With multiple files, all files will be separately (but in parallel) processed with PRISM exactly like with a single file.

$\mathcal{U}$ sends map and reduce implementations of PRISM to MapReduce, and the map phase starts. In the following, we describe the PRISM algorithms for, first, the mappers and in Section 14 the reducers. We would like to stress that the PRISM algorithms, e.g., Java ".class" files, are not encrypted and not specially protected against a curious cloud. Even though mappers and reducers know what operations they perform, they cannot deduce any private information about stored data or details about the search.

Overview. Before scanning through its local InputSplit, a mapper node creates a matrix with all elements initialized to "0". PRISM's main idea is that while the mapper scans the ciphertexts in its InputSplit, each ciphertext is assigned to one position, a certain element in the matrix by computing a hash of the ciphertext. Additionally, for each ciphertext, the mapper computes a single bit hash, and if the hash output bit is "1", the mapper puts a "1" in the matrix at the assigned position. The idea is that user $\mathcal{U}$ can also compute the position in the matrix and the one bit hash output for a word w he is looking for. Roughly speaking, $\mathcal{U}$ now queries the mapper for the value of that bit in the matrix using private information retrieval. If the bit retrieved from the mapper differs from the bit computed by $\mathcal{U}$, then $\mathcal{U}$ can decide, e.g., that w is not in this InputSplit.

Problem is that due to the limited size of the matrix and the properties of the hash function, there might be collisions in the assignment process. That is, by chance there can be two different ciphertexts being assigned with the same position in the matrix. By chance, the bit retrieved by $\mathcal{U}$ can therefore be unrelated to w. This problem is amplified by the fact that $\mathcal{U}$ does not only receive a single bit for a single InputSplit, but a combination (the *sum*) of all bits from all mappers working on InputSplits. To mitigate this problem, PRISM repeats generation and filling of matrices a total of q rounds. Also, setting an element in a matrix to "1" depends on the round number. After q rounds, the probability that the information $\mathcal{U}$ retrieved from this mapper is unrelated to w therefore decreases, and $\mathcal{U}$ can finally decide whether w is inside this file.

Definition 4 (PIR Matrix). *A binary $t \times t$ matrix $\mathcal{M}$ with $t = 2^i, i \in \mathbb{N}$ is called a PIR matrix. The mapper uses $\mathcal{M}$ to implicitly perform the privacy-preserving word search.*

Definition 5 (Candidate Position). *For each ciphertext C_i in an InputSplit, the candidate position $(\mathcal{X}_i, \mathcal{Y}_i)$ of C_i in $\mathcal{M}$ is computed by $(\mathcal{X}_i || \mathcal{Y}_i) := \lfloor C_i \rfloor_{2 \cdot \log_2(t)}$. Here, $\lfloor \ldots \rfloor_{2 \cdot \log_2(t)}$ denotes truncation after $2 \cdot \log_2(t)$ bits. So, the first $\log_2 t$ bits of C_i determine $\mathcal{X}_i$, and the second $\log_2 t$ bits determine $\mathcal{Y}_i$.*

Definition 6 (PIR Input). *If $\mathcal{U}$ is interested in a specific element $(\mathcal{X}, \mathcal{Y})$ in $\mathcal{M}$, he computes PIR input $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$, where $\alpha_{\mathcal{X}} := b \cdot (1 + a_{\mathcal{X}} \cdot N) \mod p$, and $\forall i \neq \mathcal{X}, \alpha_i := b \cdot (a_i \cdot N) \mod p$. Random values b and a_i are chosen as for the Trapdoor Group PIR scheme presented in Section 3.2.*

Definition 7 (Column Sum). *The column sum σ_i of the i^{th} column of PIR matrix $\mathcal{M}$ is defined as*

$$\sigma_i := \sum_{\mathcal{M}_{1 \leq j \leq t, i} = 1} \alpha_j,$$

where $\mathcal{M}_{1 \leq j \leq t, i} = 1$ denotes the entries in the i^{th} column of $\mathcal{M}$ that are set to 1.

Note that additions in this definition are integer additions.

The above computation of column sums is simply a digest of the PIR technique by Trostle and Parrish [37]. In short, if a mapper computes such a column sum on a given PIR matrix $\mathcal{M}$ and given PIR inputs α_i, it is impossible for the mapper to derive $(\mathcal{X}, \mathcal{Y})$. $\mathcal{U}$, however, can compute whether $\mathcal{M}_{\mathcal{X},\mathcal{Y}} = 1$, because $\mathcal{M}_{\mathcal{X},\mathcal{Y}} = 1$ **iff** $(\sigma_{\mathcal{Y}} \cdot b^{-1} \mod p) \mod 2 = 1$ holds.

It is important to point out that not only a mapper can compute a candidate position for some ciphertext in its InputSplit, but also $\mathcal{U}$ can compute candidate positions. More precisely, as $\mathcal{U}$ is looking for w, he can compute $E(w, 1)$ and candidate position $(\mathcal{X}||\mathcal{Y}) := \lfloor E(w, 1) \rfloor_{2 \cdot \log_2(t)}$. If w has been uploaded into a particular InputSplit at least once, then this InputSplit contains at least $E(w, 1)$ (maybe also $E(w, 2)$, $E(w, 3)$, ...). Therefore, it is sufficient for $\mathcal{U}$ to search for $E(w, 1)$. We will now give detailed descriptions of $PRISM$'s Map and Reduce algorithms.

Query preparation – User. To start, $\mathcal{U}$ chooses parameters $t, q \in \mathbb{N}$, where t determines the size of the PIR matrix and q the number of rounds. For day d that $\mathcal{U}$ wants to search for w, he determines key $K_d := \mathrm{HMAC}_K(d)$ and the target candidate position $(\hat{\mathcal{X}}||\hat{\mathcal{Y}}) := \lfloor E_{K_d}(w, 1) \rfloor_{2 \cdot \log_2(2)}$. To prepare PIR, $\mathcal{U}$ computes t PIR Inputs $\{\alpha_1, \alpha_2, \ldots, \alpha_t\}$ as described above. $\mathcal{U}$ sends all α as part of the following map algorithm implementation to the cloud.

The above preparation of PIR Input depending on w represents $PrepareQuery$ of Definition 1 in PRISM. The algorithm's output Q is the PIR Input. PRISM's implementation of $Process$, i.e., the cloud's operation on the encrypted file using a query Q comprises the following cloud-side Map as well as the whole cloud-side reduce below.

Map Details – Cloud. On the cloud side, all mappers process PRISM in parallel, each of them on its own, locally stored InputSplit of the current file. More precisely, a mapper executes Algorithm 2. Initially, the mapper generates q PIR matrices M_l, where each element is initially set to 0. We will now write $\mathcal{M}_{l,\mathcal{X},\mathcal{Y}}$ to denote an element $(\mathcal{X}, \mathcal{Y})$ in matrix $\mathcal{M}_l$.

The mapper node *scans* its local InputSplit consisting of ciphertexts $\{C_1, \ldots, C_n\}$. For each ciphertext C_i, the mapper creates a key-value pair (i, C_i). Then, the mapper fills matrices $\mathcal{M}_l, 1 \leq l \leq q$. For pair (i, C_i),

- the mapper computes candidate position $(\mathcal{X}_i||\mathcal{Y}_i) := \lfloor C_i \rfloor_{2 \cdot \log_2(t)}$.
- the mapper puts in PIR matrix $\mathcal{M}_j$, in element $\mathcal{M}_{j,\mathcal{X}_i,\mathcal{Y}_i}$, a "1", if the bit $bit_j := \lfloor h(C_i, j) \rfloor_1 = 1$. Here, h denotes a cryptographic hash function and "," again an unambiguous pairing of inputs. If $bit_j = 0$, element $\mathcal{M}_{j,\mathcal{X}_i,\mathcal{Y}_i}$ remains untouched. This means that entries in $\mathcal{M}_j$ can flip from 0 to 1, but never from 1 back to 0.

After all q PIR matrices are filled, the mapper computes for each matrix the t *column sums* $\sigma_{1 \leq j \leq t, 1 \leq l \leq q}$ based on $\mathcal{U}$'s input $\{\alpha_1, \ldots, \alpha_t\}$: values α_k with corresponding element $\mathrm{M}_{l,k,j}$ set to "1" are simply added. Finally, the mapper outputs intermediate key-value pairs (k, v). The *key* comprises the name of the file of the InputSplit this mapper was working on, e.g., the file name could be day d, and the number of the column sum of $\mathcal{M}_l$. The *value* consists of a list of the q column sums. These intermediate key-value pairs will now be input for the reducers during the Reduce phase.

Input: pairs (i, C_i), values $\{\alpha_1, \ldots, \alpha_t\}$
Output: intermediate key-value pairs (k, v)

```
1  for l := 1 to q do
2  |  INITIALIZE M_l;
3  end
4  SCANTHROUGHINPUTSPLIT;
5  foreach pair (i, C_i) do  //Fill
   matrices
6  |  (X_i||Y_i) := ⌊C_i⌋_{2·log2(t)};
7  |  for j := 1 to q do
8  |  |  bit_j := ⌊h(C_i, j)⌋_1;
9  |  |  if bit_j = 1 then
10 |  |  |  M_{j,x_i,y_i} := 1;
11 |  |  end
12 |  end
13 end
14 for l := 1 to q do  //q rounds
15 |  for j := 1 to t do  //Compute
   |  column sums
16 |  |  σ_{j,l} := Σ_{M_{l,1≤k≤t,j}=1} α_k;
17 |  end
18 end
19 for j := 1 to t do  //Intermediate
   (k,v) pairs
20 |  (k, v) :=({FILE, j},{σ_{j,1}, ..., σ_{j,q}});
21 |  OUTPUT (k, v);
22 end
```

Algorithm 2. Computation of matrices $\mathcal{M}$

Input: reducers' files FILE
Output: decision whether $w \in$ FILE

```
1  foreach file FILE do
2  |  for i := 1 to q do
3  |  |  if ⌊h(C, i)⌋_1 = 1 then
4  |  |  |  U reads s_{FILE,Y,i};
5  |  |  |  s_i := (s_{FILE,Y,i} · b^{-1}
   |  |  |  mod p)  mod N
6  |  |  |  //s_i = bit_j, see
   |  |  |  Alg. 2
7  |  |  |  if s_i = 0 then
8  |  |  |  |  OUTPUT w ∉ FILE;
   |  |  |  |  //Contradiction
9  |  |  |  |  break;
10 |  |  |  end
11 |  |  end
12 |  end
13 |  OUTPUT w ∈ FILE;
14 end
```

Algorithm 3. $\mathcal{U}$ decides $w \in$ FILE

Reduce Phase – Overview. Recall that there are c InputSplits and therefore c mappers. A single reducer receives from all the c mappers working on the same file all their q column sums for the *same* column. The reducer simply adds these received sums and writes the result into a file which is sent back to $\mathcal{U}$.

Reduce Phase – Details. For all key-value pairs [($\{$FILE, $i\}$,$\{\sigma_{i,1}, \ldots, \sigma_{i,q}\}$)] using the same $\{$FILE, $i\}$ as key, the MapReduce framework designates the same reducer. This reducer receives from all c different mappers working on the same file all intermediate key-value pairs with the same key. That is, a reducer receives c pairs which we rewrite as $(\{\text{FILE}, i\}, \{\sigma_{i,1,1}, \ldots, \sigma_{i,q,1}\}), \ldots, (\{\text{FILE}, i\}, \{\sigma_{i,1,c}, \ldots, \sigma_{i,q,c}\})$.

Here, for a given $\sigma_{i,j,k}$, i, $1 \leq i \leq t$, denotes the column, j, $1 \leq j \leq q$, denotes the round, and k, $1 \leq k \leq c$, the InputSplit.

Using integer addition, reducer computes q "final PIR sums" $s_{\text{FILE},i,j} := \sum_{k=1}^{c} \sigma_{i,j,k}$, $1 \leq j \leq q$, and stores values $\{s_{\text{FILE},i,1}, \ldots, s_{\text{FILE},i,q}\}$ into an output file R. To summarize, $s_{\text{FILE},i,j}$ represents the sum of column sums of all the mappers of one particular column i in PIR matrix j. This concludes the cloud's $Process$ algorithm in PRISM. The output file R is downloaded by $\mathcal{U}$.

4.3 Result Analysis

The only piece left is the $Decode$ algorithm of Definition 1 which we will describe in the following. For each outsourced file (day d), user $\mathcal{U}$ retrieves an output file generated

by reducers. Now, $\mathcal{U}$ analyzes retrieved files' content to finally conclude which of the outsourced files contain w (using $\mathcal{S}$). Again for ease of understanding, we restrict our description to the analysis of the result generated from PRISM on a single outsourced file called FILE. $\mathcal{U}$ repeats this process with all other results from the other files accordingly.

Definition 8 (Collision). *Assume $\mathcal{U}$ is looking for w, so $C := E_{K_d}(w,1)$. Similar to hash functions, a* collision *in PIR matrix $\mathcal{M}$ denotes the case of an event where the candidate position $(\mathcal{X}', \mathcal{Y}')$ of another ciphertext $C' \neq C$ matches the candidate position $(\hat{\mathcal{X}}||\hat{\mathcal{Y}}) = \lfloor E(w,1) \rfloor_{2\cdot\log_2(t)}$ of w in $\mathcal{M}$. That is, $\lfloor C \rfloor_{2\cdot\log_2(t)} = \lfloor C' \rfloor_{2\cdot\log_2(t)}$.*

Definition 9 (One-Collision). *A* one-collision *is the event where in an InputSplit a ciphertext $C' \neq E_{K_d}(w,1)$ puts a 1 into the same candidate position in $\mathcal{M}$ as $E_{K_d}(w,1)$.*

Overview: The rationale for the result analysis protocol of PRISM is to observe the candidate position of C over q rounds to mitigate the effect of one-collisions. Of particular interest will be rounds where $\lfloor h(C,i) \rfloor_1 = 1$.

First, $\mathcal{U}$ un-blinds all values received from reducers. Based on the result, $\mathcal{U}$ distinguishes two cases.

Case 1.) If a reducer, reducing for a specific file FILE, has returned the value 0 for C's candidate position, then U knows for sure that all mappers have output 0 for this candidate position. Consequently, the candidate position in matrix $\mathcal{M}$ of each mapper is 0. Therefore, C has not been in any of the InputSplits of FILE, and $\mathcal{U}$ reasons $w \notin$ FILE. If C would have been in one InputSplit, then at least the mapper working on this InputSplit would have returned a 1 in this round.

Definition 10 (Contradiction). *Let w be the word $\mathcal{U}$ is looking for, and C its ciphertext. If in some round i, $\lfloor h(C,i) \rfloor_1 = 1$ holds, and the reducer for file* FILE *sends $\mathcal{U}$ a value of 0 then this is called a* contradiction.

In case of such a contradiction, $\mathcal{U}$ for sure knows that w is not in file FILE.

Case 2.) If, however, this reducer returns a value > 0, then w was in at least one InputSplit *or* a one-collision has occurred in at least one InputSplit. User $\mathcal{U}$ can neither decide $w \notin$ FILE nor $w \in$ FILE with absolute certainty.

U's strategy is to keep the probability for one-collisions low and run multiple rounds q, such that eventually a contradiction occurs ($\Rightarrow \mathcal{U}$ decides $w \notin$ FILE), or, if no contradiction occurs, $\mathcal{U}$ decides $w \in$ FILE with only a small error probability P_{err}.

Details: $\mathcal{U}$ executes Algorithm 3. For each file, $\mathcal{U}$ is only interested in row $\hat{\mathcal{Y}}$ of matrices $\mathcal{M}$, as they can refer to candidate position $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$, only. Therefore, $\mathcal{U}$ keeps values $\{s_{\text{FILE},\hat{\mathcal{Y}},1}, \ldots, s_{\text{FILE},\hat{\mathcal{Y}},q}\}$ only and discards the rest. In each round where $\lfloor h(C,i) \rfloor_1 = 1$, un-blinds $s_{\text{FILE},\hat{\mathcal{X}},i}$ to get value $s_i := \sum_{j=1}^{c} bit_j$. If $s_i = 0$, then we have a contradiction, and $\mathcal{U}$ can infer $w \notin$ FILE. If none of the s_i values has been 0 after all the q rounds, then $\mathcal{U}$ will decide $w \in$ FILE. $\mathcal{U}$ will be wrong with P_{err}.

Note that, although PIR matrices are binary matrices, $\mathcal{U}$ sets $N > c$ to cope with the larger possible values that sums might take due to collisions.

In conclusion, $\mathcal{U}$'s strategy can be summarized by: output $w \notin$ FILE, if $\exists i, s_i = 0$ **or** output $w \in$ FILE, if $\forall i, s_i \neq 0$. We will compute $\mathcal{U}$'s error probability P_{err} for the latter case and dependencies between P_{err} and values t and q in Section 5.2.

Saving Computation: To save some computation in PRISM, we can modify the hash-based mechanism that determines whether to put a "1" or a "0" in a certain element in $\mathcal{M}$. Recall that the first $2 \cdot log_2(t)$ bit of a ciphertext C are used to determine its position (element) in $\mathcal{M}$. However, instead of computing an expensive hash function $\lfloor h(C_i, j) \rfloor_1$ to get a single bit in round j, we can simply replace the hash and take C's bit on position $(2 \cdot log_2(t) + j)$. Assuming that cipher E has good security properties (each bit of C is "1" with probability $\frac{1}{2}$), this results in the same property as using the hash: eventually two different ciphertexts that collide in $\mathcal{M}$ will differ and lead to a contradiction. We use this computation reduction in our evaluation in Section 6.

5 PRISM Analysis

5.1 Privacy

We will now show why PRISM is privacy-preserving. The main *rationale* behind our proof is to show that pairs of output generated by both our stateful cipher construction $E_K(w_i, \gamma_{w_i})$ (Section 4.1) and the PIR-based search mechanism (Section 4.2) are computationally indistinguishable for $\mathcal{A}$. Below, we assume a sufficiently large security parameter s and probabilistic polynomial time adversaries $\mathcal{A}$.

Theorem 1. *PRISM is a privacy-preserving cloud word search scheme assuming pseudorandom properties for E and the trapdoor group property of the PIR scheme.*

Proof (Sketch). Assume there would be an adversary $\mathcal{A}$ with $Pr(\text{GAME}(\mathcal{A}) = 1) > \frac{1}{2} + \epsilon(s)$, i.e., $\mathcal{A}$ has non-negligible advantage over guessing. As PRISM generates $\mathcal{E}_b$ and $\mathcal{Q}_b$ independently from each other, this would indicate that $\mathcal{A}$ has non-negligible advantage over guessing in determining b from either $\mathcal{E}_b$ or $\mathcal{Q}_b$ (or both).

We will now show with the following two lemmas that this is impossible.

Lemma 1. *In PRISM, any pair of sequences of ciphertexts (files E_L and $E_{L'}$) generated by a pair of sequences of words (files L and L') is computationally indistinguishable for $\mathcal{A}$, assuming E is a pseudorandom permutation and "," an unambiguous pairing of inputs.*

Proof (Sketch). First, note that our stateful-cipher uses a different random key for each file. In a learning phase, $\mathcal{A}$ makes a number of queries to two stateful-cipher oracles encrypting with two different keys K_0, K_1. Then, $\mathcal{A}$ prepares word w, submits to a challenge oracle and gets back $E_{K_b}(w, \gamma_{b,w})$, $b \in \{0, 1\}$. $\mathcal{A}$ has to output b correctly with only negligible advantage over guessing.

However, we now show by using the hybrid argument [28] that the distributions generated by $E_{K_i}(w, \gamma_{i,w})$ are computationally indistinguishable for $\mathcal{A}$. That is, pairs $E_{K_0}(w, \gamma_{0,w})$, $E_{K_1}(w, \gamma_{1,w})$ are distinguishable with only negligible advantage over guessing). As "," is an unambiguous pairing, we now write w'_i instead of $(w, \gamma_{i,w})$.

Our hybrid distributions are: (1.) $PRP_{K_0}(w'_0)$, (2.) $RP_{K_0}(w'_0)$, (3.) $RF_{K_0}(w'_0)$, (4.) $RF_{K_1}(w'_1)$, (5.) $RP_{K_1}(w'_1)$, and (6.) $PRP_{K_1}(w'_1)$. "PRP" means pseudorandom permutation, "RP" random permutation, and "RF" random function.

(1.) - (2.) and (5.) - (6.): by definition of pseudorandom permutation, the probability to distinguish $PRP_K(w'_i)$ from $RP_K(w'_i)$ is negligible.

(2.) - (3.) and (4.) - (5.): the probability to distinguish a random permutation from a random function is negligible, cf., Section 3.6.3 in Katz and Lindell [28].

(3.) - (4.): If $\mathcal{A}$ observes $RF_{K_0}(w'_0) = RF_{K_1}(w'_1)$ for a pair w'_0, w'_1, then this only indicates a collision in RF_{K_0} and RF_{K_1}. Even if $\mathcal{A}$ queries the same w multiple times, output $RF_{K_0}(w, \gamma_{0,w})$ or $RF_{K_1}(w, \gamma_{1,w})$ will always be different as counters increase. If RF_{K_0} (or RF_{K_1}) outputs the same value twice (unlikely), this only indicates a collision in RF_{K_0} (or RF_{K_1}). The advantage over guessing in distinguishing $RF_{K_0}(w'_0)$ from $RF_{K_1}(w'_1)$ is zero.

$\mathcal{A}$'s advantage over guessing in distinguishing pairs $E_L, E_{L'}$ is negligible. □

Lemma 2. *Based on the trapdoor group assumption ("*TGA"*), PRISM's PIR-search produces computationally indistinguishable pairs of queries Q_i.*

Proof (Sketch). Assume $\mathcal{A}$ submits two words w_1, w_2 to an oracle. The oracle picks $\hat{b} \in \{0, 1\}$ and returns $Q_{\hat{b}} := \{\alpha_{\hat{b},1} := b \cdot e_{\hat{b},1} \mod p, \ldots, \alpha_{\hat{b},t} := b \cdot e_{\hat{b},t} \mod p\}$, with $e_{\hat{b},\mathcal{X}_{\hat{b}}} := 1 + a_{\mathcal{X}_{\hat{b}}} \cdot N, \forall i \neq \mathcal{X}_{\hat{b}} : e_{\hat{b},i} := a_i \cdot N$. Here, $\mathcal{X}_{\hat{b}} := \lfloor E_K(w_{\hat{b}}, 1) \rfloor_{\log_2(t)}$, and a_i are chosen randomly. $\mathcal{A}$ has to output $\hat{b}$ with non-negligible advantage over guessing.

However, we will now show that any pair of sequences of α values is computationally indistinguishable for $\mathcal{A}$. The proof is a direct implication of the security of the PIR protocol, based on *TGA*: for all adversaries $\mathcal{A}$, $Pr[\mathcal{A}(b \cdot e_1, \ldots, b \cdot e_t) = LSB(e_1, \ldots, e_t)] = \epsilon(s)$. That is, given $b \cdot e_i$, the probability that $\mathcal{A}$ computes low order bits of $e_i \mod N$ ("LSB") is negligible [37].

Assume that $\mathcal{A}$ can distinguish sequences $Q_0 = \{\alpha_{0,i}\}$ and $Q_1 = \{\alpha_{1,i}\}$ with non-negligible advantage. This would violate *TGA* as follows. First, note that besides $\alpha_{0,\mathcal{X}_0}, \alpha_{1,\mathcal{X}_1}$ all elements in both sequences $\{\alpha_{0,i}\}$ and $\{\alpha_{1,i}\}$ are created in the same way (multiplication of b with a random number). Therefore, besides $\alpha_{0,\mathcal{X}_0}, \alpha_{1,\mathcal{X}_1}$, any pair $(\alpha, \alpha') \in \{\alpha_{0,i}\} \cup \{\alpha_{1,i}\}$ is computationally indistinguishable for $\mathcal{A}$. If $\mathcal{A}$ can still distinguish between sequences $\{\alpha_{0,i}\}$ and $\{\alpha_{1,i}\}$, then $\mathcal{A}$ can determine with non-negligible probability $\mathcal{X}_0$ or $\mathcal{X}_1$ and thus value i with $e_i = 1 \mod N$, violating *TGA*. □

5.2 Statistical Analysis

We now discuss how $\mathcal{U}$ chooses parameters t and q to get a certain error probability P_{err}. This probability describes the chance that, despite $w \notin$ FILE, $\mathcal{U}$ wrongly outputs $w \in$ FILE after q rounds without a contradiction, cf., Algorithm 3. Let n be the number of ciphertexts in one InputSplit, $n := \frac{S_{\text{InputSplit}}}{\text{CipherBlockSize}}$. The total number of ciphertexts stored in the cloud is $(c \cdot n)$. We consider for simplicity only rounds where $\lfloor h(C, i) \rfloor_1 = 1$, cf., Algorithm 3. With h a cryptographic hash, $\lfloor h(C, i) \rfloor_1 = 1$ in $q' \approx \frac{q}{2}$ rounds.

While inserting any ciphertext, the collision probability is $P_{\text{collision}} := \frac{1}{t^2}$. The probability for a one-collision is $P_{\text{one-collision}} := \frac{P_{\text{collision}}}{2}$. If w is *not* inside an InputSplit,

the probability that, after inserting the n ciphertexts of that InputSplit into $\mathcal{M}$, the candidate position is *not* set to 1 is $P_{\text{InputSplit,no-one-collision}} := (1 - P_{\text{one-collision}})^n$.

If $w \notin$ FILE, i.e., in *none* of the InputSplits, the probability that the candidate position is not set to 1 in *any* InputSplit is $P_{\text{contradiction}} := (P_{\text{InputSplit,no-one-collision}})^c$. This is the probability that a contradiction occurs in a single round. If $w \notin$ FILE, the probability that a contradiction occurs in *at least one* round is $P_{\text{contradiction,q-rounds}} := 1 - (1 - P_{\text{contradiction}})^q$.

After q rounds without a contradiction, $\mathcal{U}$ automatically decides that w is in FILE. In case that $w \notin$ FILE, and *no* contradiction occurs in q rounds, $\mathcal{U}$ is therefore wrong with $P_{\text{err}} := 1 - P_{\text{contradiction,q-rounds}} = (1 - (1 - \frac{1}{2 \cdot t^2})^{cn})^q$.

Given a certain file size, the size of InputSplits, and the blocksize of the symmetric cipher, $\mathcal{U}$ computes c and n. Therewith, $\mathcal{U}$ can target a false-positive probability by appropriately selecting t and q. We evaluate this using a real-world scenario in Section 6.

6 Evaluation

To show its real-world feasibility, we have implemented and evaluated PRISM with the scenario described in the introduction. The source code is available for public download [1]. We received 16 days of log data from May 2010 from a small local Internet provider. This provider logs and retains all customers' DNS resolve requests for possible forensic analysis and intrusion detection. Log data is split into files on a daily basis. Each file contains one day of logged 3-tuples: timestamp, customer IP (anonymized by provider for regulatory matters), hostname. The scenario for our evaluation is to use PRISM to upload this data encrypted to MapReduce and perform a search for specific hostnames in a privacy-preserving manner. This is useful for, e.g., "passive DNS analysis" to determine at which day certain command-and-control centers of botnets have been accessed by customer machines, cf., Bilge et al. [5]. The goal of our experiments was to analyze the computational overhead induced by PRISM's privacy mechanism, i.e., the additional time consumed by PRISM over non-privacy-preserving MapReduce.

6.1 Setup

For the 16 days, the log data contains $\approx 3 \cdot 10^8$ log entries, i.e., $\approx 2 \cdot 10^7$ per file/day. The total space required by all files uploaded into MapReduce using PRISM is 27 GByte, on average 1.7 GByte per file.

Our experiments have been performed on a small "cloud" comprising 1 master computer and 9 slaves. Computers featured a 2.5 GHz Pentium Dual Core and 4 GByte of RAM, running a standard desktop installation of Fedora 11. With this hardware configuration, a total of 18 CPUs were available for maps and reduces. We installed Hadoop version 0.20.2 on our cloud. Being aware that tailoring MapReduce's configuration parameters can have a huge impact on performance, we use the standard, out-of-the-box configuration of Hadoop 0.20.2 without any configuration tweaks. Performance tuning is out of scope of this paper. Similarly, as the InputSplit size is recommended to be between 64 MByte and 128 MByte, we chose $S_{\text{InputSplit}} = 96$ MByte (InputSplits must be dividable by $3 \cdot 32$ Byte, since log entries are 3-tuples).

Table 1. Parameters t, q to achieve $P_{\text{err}} < 0.01$

	File size (GByte)															
	0.45	1.21	1.32	1.36	1.38	1.45	1.52	1.67	1.78	1.93	2.00	2.08	2.09	2.14	2.21	2.25
t	2^{10}	2^{11}							2^{12}							
q	100	60						80	20							

In addition to the evaluation with 96 MByte InputSplits, we also performed a second measurement with larger InputSplits of 120 MByte. We expected a slightly improved performance of PRISM due to the fact that for the larger files the total number of InputSplits c reduced to less than our 18 available CPUs. Therewith, no costly (re-)scheduling takes places, and mappers do not have to process 2 InputSplits sequentially.

Finally, to put timing results into perspective, we implemented and measured a trivial, non-privacy-preserving MapReduce search called *Baseline*. Baseline search consists of an empty map phase, where mappers simply scan over InputSplits and compare each word of the InputSplit with a predetermined one, but do not generate any key-value pairs. Only at the end of the map phase, a single intermediate key-value pair per mapper (e.g., "found") is sent to reducers. Reducers discard this key-value pair and write empty files to disk. This trivial baseline only serves in deducing the overhead implied by PRISM, not taking MapReduce specific delays due to rescheduling, speculative execution of backup tasks etc. [17, 34] into account. Note that linear scanning through the entire InputSplit is mandatory, as we assume our data to be unordered and unsorted.

For the private information retrieval algorithm, we set $m = 400$ as suggested by Trostle and Parrish [37] for good security. Our Java implementation is a naive, straightforward implementation using Java's BigInteger without any performance optimizations. As symmetric encryption cipher, we used AES with 256 Bit blocksize from the GNU Crypto Library V2.0.1 [20]. As individual DNS entries occurred way less than 2^{16} times per day, we reserved $|\gamma| = 2$ Bytes and truncated entries longer than 30 Byte down to the last 30 Byte. Because the size of input $|w_i| + |\gamma_{w_i}|$ is less than E's blocksize (using standard padding for w_i), *concatenation* provides an unambiguous pairing.

Simulating $\mathcal{U}$, we computed n and c using blocksize, InputSplit size $S_{\text{InputSplit}}$, and individual file size S_{File}. Assuming that $\mathcal{U}$ targets an error probability of $P_{\text{err}} < 0.01$, we derived t and q. Table 1 summarizes parameters (t, q) computed for each file individually. Compared to q, we observed that parameter t has a much higher impact on P_{err}, but a comparatively lower impact on computations. Therefore, we increased preferably t than q. Higher values for (t, q) will achieve even smaller values for P_{err}, but Table 1 shows the computationally "cheapest" combination of (t, q).

6.2 Results

Computational overhead at the cloud is low as indicated by Figure 1 (PRISM's timing results). We have sorted the 16 files based on their size in an increasing order, i.e, the size of the smallest log file we received from the Internet provider was 0.45 GByte, the largest one was 2.25 GByte. PRISM's execution time was clocked on each file 6 times, respectively, and Fig. 1 shows the average. For each file, Fig. 1 shows two stacked boxes, respectively: the first one for 96 MByte and the second one for 120 MByte

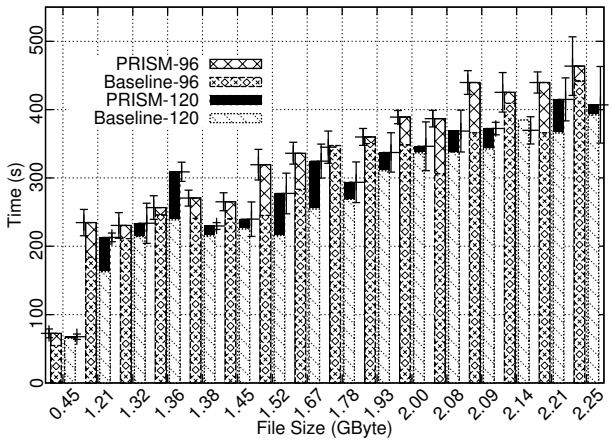

Fig. 1. Wall clock timings for PRISM and Baseline, with $S_{\mathrm{InputSplit}} = 96$ and 120 MByte

InputSplit size. Each of the stacked boxes comprises, first, the baseline timing and, second, the additional time required to run PRISM. To give trust into the evaluation, Fig. 1 also shows 95% confidence intervals drawn right next to each box.

Timings shown in Fig. 1 are "wall clock" timings. This captures the *complete* time elapsed from submitting the PRISM map and reduce classes and starting the job until the end of the reduce phase. In the real-world, wall clock time reflects the time a cloud, e.g., Amazon [2], would charge a user $\mathcal{U}$. In conclusion, the additional overhead over the trivial Baseline MapReduce jobs was on average 11% with a 95% confidence interval of ± 3. The largest overhead seen was 24% over Baseline. This overhead is mostly computational overhead, as there is no difference in disk access between Baseline and PRISM and network volume increases only little by sending slightly larger values during "Reduce". These results do not only show the feasibility of PRISM in practice, but also demonstrate the low overhead implied by PRISM over the non-privacy preserving MapReduce job. We claim that a performance optimized (not based on Java BigInteger) implementation improves performance significantly and furthermore reduces overhead.

The simple increase from 96 MByte InputSplit size to 120 MByte InputSplit size has reduced wall clock times for MapReduce jobs by 9% on average (95% confidence interval of ± 4). Files of size smaller than 2 GByte are split into $\leq$ 18 InputSplits, and both jobs, PRISM and Baseline, are processed completely in parallel. This indicates that a careful configuration of Hadoop MapReduce's many system parameters, handcrafted and specific to the scenario and jobs to be executed, will lead to substantial performance improvements. This also indicates that in a cloud with more CPUs than in our small setup, the increased number of CPUs will enable to configure way *smaller* InputSplits being processed in parallel. Substantially smaller InputSplits will be beneficial for the overall performance of PRISM or any MapReduce job. However, increasing the number of InputSplits also implies a performance penalty due to (re-)scheduling and coordination activities of the central job tracker, cf., Pavlo et al. [34], so a trade-off has to be found. MapReduce configuration optimizations are, however, out of scope.

To better understand the cloud's computational overhead, we also measured the **computation time for a PRISM mapper**. On a single CPU, execution of an isolated PRISM map function on a single InputSplit is ca. 9 times slower than Baseline (9.3 for 96 MByte and 9.1 for 120 MByte, 95% confidence interval of ± 0.1). While this seems to be a lot, we remark that 1.) this map overhead is constant for an InputSplit and does not depend on or scale with the total size of the data, 2.) there is a lot of potential to improve our map implementation, 3.) this overhead is obviously amortized by other MapReduce aspects such as the Reduce phase and also disk latency, network overhead etc., and 4.) a user is charged for the total system time, i.e., the wall clock time.

Computational overhead at the User is also low in PRISM: per file, the preparation of, e.g., 2^{12} α values for the underlying PIR scheme is barely measurable (≈ 200 ms) on a PC with 2.5 GHz CPU. During result analysis, $\mathcal{U}$ automatically discards all received values that he is not interested in, i.e., all besides $s_{\text{FILE},\mathcal{Y},1\leq i\leq q}$. For these q values, a total of q Java BigInteger multiplications with modulo have to be performed. For our examples with $q \leq 100$, this was not measurable at less than 1 ms.

Memory consumption for $\mathcal{U}$ is, on the one hand, constant; $\mathcal{U}$ only stores the 256 bit AES key K. On the other hand however, the cloud user $\mathcal{U}$'s memory consumption scales linearly with $O(\Sigma)$, i.e., the number of *different* words. This is due to the construction of our stateful cipher that stores counters γ in a hashtable. In our straightforward implementation with Java's standard Hashtable, memory consumption of this hashtable was 548 MByte for the largest log file. While this is certainly a lot of RAM, we conjecture this to be available on PC hardware – moreover, as there is a large potential for performance tuning with such data structures.

Communication overhead for PRISM is dominated by the underlying PIR scheme. $\mathcal{U}$ sends, besides .class files once, only the t α values per file to the cloud. For example, with $t = 2^{12}$ and $m = 400$ Bit, this computes to 200 KByte per file. The response from the cloud is, for each round, t values of size m. The most expensive configuration in terms of communication in our experiments has been $t = 2^{10}, q = 100$; this results in ≈ 5 MByte communication overhead. Note that communication complexity in the underlying PIR scheme by Trostle and Parrish [37] is linear in the square root of the total table size, i.e., $O(t)$. This can be further reduced by using recursive PIR queries to $O(t^{\epsilon})$, for any $\epsilon > 0$ [29]. Those optimizations as well as amortization techniques discussed by Ostrovsky and Skeith [33] are out of scope.

In conclusion, PRISM is very lightweight for a user using standard PC hardware.

Discussion: On a larger cluster in a more professional environment (hundreds or even thousands of CPUs [25]), all files will be processed in parallel. As shown in Fig. 1, total time for the 2 GByte file is ≈ 350 s. However, already ≈ 340 s are required by MapReduce just to "scan" through the various InputSplits, see Baseline. Such inefficiency with non-optimal configurations has been observed before, and our results are along the lines of Pavlo et al. [34]. Here, a "grep"-like MapReduce job on 1 TByte of data took $\approx 1,500$ s on 50 CPUs which would be ≈ 20 times faster than our Baseline. However, Pavlo et al. [34] use a slightly tuned configuration and moreover a more efficient scanning through InputSplits (100 Byte text values instead of 32 Byte binary values in our case) which is known to lead to significant performance increases [27].

7 Related Work

Private Information Retrieval: Private Information Retrieval (and similarly oblivious transfer and oblivious RAM) has received a lot of attention [9, 13, 19, 22, 29, 32, 33, 35]. In PIR, a user retrieves a specific data from a database. The only "privacy" goal in PIR is access privacy whereby the server should not discover which data a user is interested in. Note that PIR does not ensure privacy of data in the database. PRISM, however, focuses on *searching* for a word and uses PIR only as a tool.

Searchable Encryption: With searching on encrypted data techniques [6], user privacy is guaranteed thanks to the encryption of the queries and the stored data. However, PRISM offers *higher* privacy guarantees since in existing searchable encryption solutions [4, 6–8, 10, 16, 21, 31, 36], the result ("found" or "not found") originating from a query is known to the adversary; therefore as opposed to PRISM, standard searchable encryption techniques do not ensure *query privacy*. Moreover, existing mechanisms *cannot* be easily extended to leverage from a parallelized cloud setup: while in theory the search on encrypted data itself could be run in parallel on subsets of data, today's solution do not support the *combination* (*aggregation*) of results (as in a reduce phase). To conclude, PRISM not only ensures both *storage privacy* and *query privacy*, but also enables the aggregation of results originating from intermediate parallelized operations.

8 Conclusion

PRISM is the first privacy-preserving search scheme suited for cloud computing. That is, PRISM provides storage and query privacy while introducing only limited overhead. PRISM is specifically designed to leverage parallelism and efficiency of the MapReduce paradigm. Moreover, PRISM is compatible with any standard MapReduce-based cloud infrastructure (such as Amazon's), and does not require modifications to the underlying system. Thanks to this compatibility, PRISM has been efficiently implemented on an experiemental cloud computing environment using Hadoop MapReduce. Besides a throughout analysis, performance of PRISM has been evaluated on that environment through search operations in DNS logs provided by an ISP. PRISM's overhead over non-privacy-preserving search is only 11% on average, acertaining its efficiency.

References

[1] PRISM source code (2012), `http://www.ccs.neu.edu/~blass/prism.tgz`
[2] Amazon. Elastic mapreduce (2010), `http://aws.amazon.com/elasticmapreduce/`
[3] Apache. Hadoop (2010), `http://hadoop.apache.org/`
[4] Bellovin, S.M., Cheswick, W.R.: Privacy-enhanced searches using encrypted Bloom filters (2007), `http://mice.cs.columbia.edu/getTechreport.php?techreportID=483`
[5] Bilge, L., Kirda, E., Krügel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive dns analysis. In: Proceedings of 18th Annual Network and Distributed System Security Symposium, San Diego, USA, pp. 195–211 (2011) ISBN 1891562320

[6] Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public Key Encryption with Keyword Search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
[7] Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith III, W.E.: Public Key Encryption That Allows PIR Queries. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 50–67. Springer, Heidelberg (2007)
[8] Brassard, G., Crépeau, C., Robert, J.M.: All-or-Nothing Disclosure of Secrets. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 234–238. Springer, Heidelberg (1987)
[9] Cachin, C., Micali, S., Stadler, M.A.: Computationally Private Information Retrieval with Polylogarithmic Communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–412. Springer, Heidelberg (1999)
[10] Chang, Y.-C., Mitzenmacher, M.: Privacy Preserving Keyword Searches on Remote Encrypted Data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
[11] Chief Information Officer's Council. Proposed security assessment & authorization for U.S. government cloud computing (2010), http://www.digitalgovernment.com/media/Knowledge-Centers/asset_upload_file652_2491.pdf
[12] Chief Information Officer's Council. Privacy recommendations for the use of cloud computing by federal departments and agencies (2010), http://www.cio.gov/
[13] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proceedings of Symposium on Foundations of Computer Science, Milwaukee, USA, pp. 41–51 (1995)
[14] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing (2009), https://cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf
[15] Cloud Security Alliance. Top cloud computing threats (2010), https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf
[16] Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of Conference on Computer and Communications Security, CCS, Alexandria, USA, pp. 79–88 (2006)
[17] Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of OSDI, San Francisco, USA, pp. 137–150 (2004)
[18] EU, Eu information management instruments (2010), http://europa.eu/
[19] Gertner, Y., Ishai, Y., Kushilevitz, E.: Protecting data privacy in private information retrieval. In: Proceedings of Symposium on Theory of Computing, Dallas, USA, pp. 151–160 (1998) ISBN 0-89791-962-9
[20] GNU, The gnu crypto project (2011), http://www.gnu.org/software/
[21] Goh, E.-J.: Secure indexes. Cryptology ePrint Archive Report 2003/216 (2003), http://eprint.iacr.org/2003/216
[22] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious ram. Journal of the ACM 45, 431–473 (1996) ISSN 0004-5411
[23] Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984) ISSN 0022-0000
[24] Google. Google apps for government (2010), http://googleenterprise.blogspot.com/2010/07/google-apps-for-government.html
[25] Hadoop. Powered by hadoop, list of applications using hadoop mapreduce (2011), http://wiki.apache.org/hadoop/PoweredBy

[26] Hall, C., Goldberg, I., Schneier, B.: Reaction Attacks against Several Public-Key Cryptosystem. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 2–12. Springer, Heidelberg (1999)
[27] Jian, D., Ooi, B.C., Shi, L., Wu, S.: The performance of mapreduce: An in-depth study. Proceedings of the VLDB Endowment 3(1), 472–483 (2010)
[28] Katz, J., Lindell, Y.: Introduction to modern cryptography. Chapman & Hall/CRC (2008) ISBN 978-1-58488-551-1
[29] Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: Proceedings of Symposium on Foundations of Computer Science, Miami Beach, USA, pp. 364–373 (1997)
[30] McCullagh, D.: Fbi wants records kept of web sites visited (2010), http://news.cnet.com/8301-13578_3-10448060-38.html
[31] Ogata, W., Kurosawa, K.: Oblivious keyword search. Journal of Complexity – Special Issue on Coding and Cryptography 20, 356–371 (2004) ISSN 0885-064X
[32] Ostrovsky, R., Skeith III, W.E.: Private Searching on Streaming Data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005)
[33] Ostrovsky, R., Skeith III, W.E.: A Survey of Single-Database Private Information Retrieval: Techniques and Applications. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 393–411. Springer, Heidelberg (2007)
[34] Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: Proceedings of International Conference on Management of Data, Rhode Island, USA, pp. 165–178 (2009)
[35] Sion, R., Carbunar, B.: On the computational practicality of private information retrieval. In: Proceedings of Network and Distributed Systems Security Symposium, San Diego, USA, pp. 1–10 (2007)
[36] Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of Symposium on Security and Privacy, Berkeley, USA, pp. 44–55 (2000)
[37] Trostle, J., Parrish, A.: Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 114–128. Springer, Heidelberg (2011)

StealthGuard: Proofs of Retrievability with Hidden Watchdogs

Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva, and Melek Önen*

EURECOM, Sophia Antipolis, France
{azraoui, elkhiyao, molva, onen}@eurecom.fr

Abstract. This paper presents **StealthGuard**, an efficient and provably secure proof of retrievabillity (POR) scheme. **StealthGuard** makes use of a privacy-preserving word search (WS) algorithm to search, as part of a POR query, for randomly-valued blocks called watchdogs that are inserted in the file before outsourcing. Thanks to the privacy-preserving features of the WS, neither the cloud provider nor a third party intruder can guess which watchdog is queried in each POR query. Similarly, the responses to POR queries are also obfuscated. Hence to answer correctly to every new set of POR queries, the cloud provider has to retain the file in its entirety. **StealthGuard** stands out from the earlier sentinel-based POR scheme proposed by Juels and Kaliski (JK), due to the use of WS and the support for an unlimited number of queries by **StealthGuard**. The paper also presents a formal security analysis of the protocol.

Keywords: Cloud storage, Proofs of Retrievability, Privacy-preserving word search

1 Introduction

Nowadays outsourcing, that is, delegating one's computing to external parties, is a well established trend in cloud computing. Along with unprecedented advantages such as lower cost of ownership, adaptivity, and increased capacity, outsourcing also raises new security and privacy concerns in that critical data processing and storage operations are performed remotely by potentially untrusted parties. In this paper we focus on data retrievability, a security requirement akin to outsourced data storage services like Dropbox [1] and Amazon Simple Storage Service [2]. Data retrievability provides the customer of a storage service with the assurance that a data segment is actually present in the remote storage. Data retrievability is a new form of integrity requirement in that the customer of the storage or the data owner does not need to keep or get a copy of the data segment in order to get the assurance of retrievability thereof. A cryptographic building block called Proof of Retrievability (POR) was first developed by Juels and Kaliski [13] (JK) to meet this requirement. In the definition of [13], a successful execution of the POR scheme assures a verifier that it can retrieve F in its entirety. Classical

* Authors are listed in alphabetical order.
[1] Dropbox - https://www.dropbox.com/
[2] Amazon Simple Storage Service - http://aws.amazon.com/fr/s3/

integrity techniques such as transferring F with some integrity check value are not practical since they incur very high communication or computational costs that are linear with the size of F. POR schemes aim at much lower cost both in terms of communications and processing by avoiding transmission or handling of F in its entirety. To that effect, POR schemes require the prover to perform some operations on some randomly selected parts of F and the verifier is able to check the result returned by the prover with the knowledge of very brief reference about the data like a secret key. Most POR schemes thus are probabilistic and their performance is measured in the trade-off between the bandwidth and processing overhead and the rate of retrievability assurance.

In this paper we develop **StealthGuard**, a new POR scheme that achieves good retrievability assurance with acceptable costs. The main idea behind the new scheme is a combination of a privacy-preserving word search (WS) algorithm suited to large datastores with the insertion in data segments of randomly generated short bit sequences called **watchdogs**. In **StealthGuard**, the user inserts these watchdogs in randomly chosen locations of the file F and stores the resulting file in the cloud. In order to check the retrievability of F the user issues lookup queries for selected values of watchdogs using the WS scheme. The user decrypts the WS replies from the cloud server in order to get the proof of retrievability for each segment targeted by the WS queries. Each positive result is the proof of presence for the corresponding data segment. Thanks to the features of the WS, neither the cloud server nor a third party intruder can guess which watchdogs are targeted by each WS query or response.

Even though there is an analogy between the watchdogs used in **StealthGuard** and the sentinels akin to the JK scheme [13], there is a major difference between the two schemes due to the use of WS by **StealthGuard**: the number of POR queries that can be issued in **StealthGuard** without requiring any update of the watchdogs is unbounded whereas in the JK scheme a given set of sentinels can be used for a finite number of POR queries only. **StealthGuard** only requires the transfer of some additional data that is a small percentage of F in size and a good POR rate can be achieved by only processing a fraction of F. In addition to the description of our proposal, we give a new security model that enhances existing security definitions of POR schemes [13, 17]. We state a generic definition of the soundness property that applies to any POR scheme.

Contributions. To summarize, this paper offers two main contributions:

− We present **StealthGuard**, a new POR scheme based on the insertion of watchdogs that requires a light file preprocessing and on a privacy-preserving WS that allows a user to issue an unbounded number of POR queries. Besides, the user is stateless since it only needs to keep a secret key to be able to run the POR protocol.

− We propose a new security model which improves existing security definitions [13, 17]. We also provide a formal proof of our proposal under this new security model.

The rest of the paper is organized as follows. Section 2 defines the entities and the algorithms involved in a POR scheme. Section 3 describes the adversary models that are considered in this paper. Section 4 provides an overview of **StealthGuard** and Section 5 gives details of the protocol. Section 6 analyses its security properties. Section 7 evaluates its security and its efficiency. We review the state of the art in Section 8.

2 Background

Before presenting the formal definition of PORs and the related security definitions, we introduce the entities that we will refer to in the remainder of this paper.

2.1 Entities

A POR scheme comprises the following entities:

- Client $\mathcal{C}$: It possesses a set of files $\mathcal{F}$ that it outsources to the cloud server $\mathcal{S}$. Without loss of generality, we assume that each file $F \in \mathcal{F}$ is composed of n splits $\{S_1, S_2, ..., S_n\}$ of equal size L bits. In practice, if the size of F is not a multiple of L, then padding bits will be added to F. We also suppose that each split S_i comprises m blocks of l bits $\{b_{i,1}, b_{i,2}, ..., b_{i,m}\}$, i.e., $L = m \cdot l$.
- Cloud Server $\mathcal{S}$ (a potentially malicious prover): For each file $F \in \mathcal{F}$, the cloud server $\mathcal{S}$ stores an "enlarged" verifiable version $\hat{F}$ of that file, that enables it to prove to a verifier $\mathcal{V}$ that the client $\mathcal{C}$ can still retrieve its original file F.
- Verifier $\mathcal{V}$: It is an entity which via an interactive protocol can check whether the cloud server $\mathcal{S}$ (i.e., the prover) is still storing some file $F \in \mathcal{F}$ or not. The verifier can be either the client itself or any other *authorized* entity, such as an auditor.

2.2 POR

A POR scheme consists of five polynomial-time algorithms (cf. [13, 17]):

- $\mathsf{KeyGen}(1^\tau) \rightarrow K$: This probabilistic key generation algorithm is executed by client $\mathcal{C}$. It takes as input a security parameter τ, and outputs a *secret key* K for $\mathcal{C}$.
- $\mathsf{Encode}(K, F) \rightarrow (\mathsf{fid}, \hat{F})$: It takes the key K and the file $F = \{S_1, S_2, ..., S_n\}$ as inputs, and returns the file $\hat{F} = \{\hat{S}_1, \hat{S}_2, ..., \hat{S}_n\}$ and F's *unique* identifier fid. Cloud server $\mathcal{S}$ is required to store $\hat{F}$ together with fid. $\hat{F}$ is obtained by first applying to F an *error-correcting code* (ECC) which allows client $\mathcal{C}$ to recover the file from minor corruptions that may go undetected by the POR scheme, and further by adding some *verifiable redundancy* that enables client $\mathcal{C}$ to check whether cloud server $\mathcal{S}$ still stores a *retrievable* version of F or not.
  Note that the Encode algorithm is invertible. Namely, there exists an algorithm Decode that allows the client $\mathcal{C}$ to recover its original file F from the file $\hat{F}$.
- $\mathsf{Challenge}(K, \mathsf{fid}) \rightarrow \mathsf{chal}$: The verifier $\mathcal{V}$ calls this *probabilistic* algorithm to generate a challenge chal for an execution of the POR protocol for some file F. This algorithm takes as inputs the secret key K and the file identifier fid, and returns the challenge chal that will be sent to cloud server $\mathcal{S}$.
- $\mathsf{ProofGen}(\mathsf{fid}, \mathsf{chal}) \rightarrow \mathcal{P}$: On receiving the challenge chal and the file identifier fid, cloud server $\mathcal{S}$ executes ProofGen to generate a proof of retrievability $\mathcal{P}$ for the file $\hat{F}$ whose identifier is fid. The proof $\mathcal{P}$ is then transmitted to verifier $\mathcal{V}$.
- $\mathsf{ProofVerif}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) \rightarrow b \in \{0, 1\}$: Verifier $\mathcal{V}$ runs this algorithm to check the validity of the proofs of retrievability sent by cloud server $\mathcal{S}$. On input of the key K, the file identifier fid, the challenge chal, and the proof $\mathcal{P}$, the ProofVerif algorithm outputs bit $b = 1$ if the proof $\mathcal{P}$ is a valid proof, and $b = 0$ otherwise.

3 Adversary models

A POR scheme should ensure that if cloud server $\mathcal{S}$ is storing the outsourced files, then the ProofVerif algorithm should always output 1, meaning that ProofVerif does not yield any false negatives. This corresponds to the *completeness* property of the POR scheme. PORs should also guarantee that if $\mathcal{S}$ provides a number (to be determined) of valid proofs of retrievability for some file F, then verifier $\mathcal{V}$ can deduce that server $\mathcal{S}$ is storing a retrievable version of F. This matches the *soundness* property of POR. These two properties are formally defined in the following sections.

3.1 Completeness

If cloud server $\mathcal{S}$ and verifier $\mathcal{V}$ are both honest, then on input of a challenge chal and some file identifier fid sent by verifier $\mathcal{V}$, the ProofGen algorithm generates a proof of retrievability $\mathcal{P}$ that will be accepted by verifier $\mathcal{V}$ with probability 1.

Definition 1 (Completeness). *A POR scheme is* complete *if for any honest pair of cloud server $\mathcal{S}$ and verifier $\mathcal{V}$, and for any challenge* $\mathsf{chal} \leftarrow \mathsf{Challenge}(K, \mathsf{fid})$*:*

$$\mathsf{Pr}(\mathsf{ProofVerif}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) \rightarrow 1 \mid \mathcal{P} \leftarrow \mathsf{ProofGen}(\mathsf{fid}, \mathsf{chal})) = 1$$

3.2 Soundness

A proof of retrievability is deemed sound, if for any malicious cloud server $\mathcal{S}$, the only way to convince verifier $\mathcal{V}$ that it is storing a file F is by actually keeping a retrievable version of that file. This implies that any cloud server $\mathcal{S}$ that generates (a polynomial number of) valid proofs of retrievability for some file F, must possess a version of that file that can be used later by client $\mathcal{C}$ to recover F. To reflect the intuition behind this definition of soundness, Juels and Kaliski [13] suggested the use of a file extractor algorithm $\mathcal{E}$ that is able to retrieve the file F by interacting with cloud server $\mathcal{S}$ using the *sound* POR protocol. Along these lines, we present a new and a more generic soundness definition that refines the formalization of Shacham and Waters [17] which in turn builds upon the work of Juels and Kaliski [13]. Although the definition of Shacham and Waters [17] captures the soundness of POR schemes that empower the verifier with unlimited (i.e. exponential) number of "possible" POR challenges [3, 17, 23], it does not define properly the soundness of POR schemes with limited number of "possible" POR challenges such as in [13, 19] and in **StealthGuard**[3]. We recall that the formalization in [17] considers a POR to be sound, if a file can be recovered whenever the cloud server generates a valid POR response for that file with a *non-negligible* probability. While this definition is accurate in the case where the verifier is endowed with unlimited number of POR challenges, it cannot be employed to evaluate the soundness of the mechanisms introduced in [13, 19] or the solution we will present in this paper. For example, if we take the POR scheme in [19] and if we consider a scenario where the

[3] Note that having a bounded number of POR challenges does not negate the fact that the verifier can perform unlimited number of POR queries with these same challenges, cf. [19].

cloud server corrupts randomly half of the outsourced files, then the cloud server will be able to correctly answer half (which is non-negligible) of the POR challenges that the verifier issues, yet the files are irretrievable. This implies that this POR mechanism is not secure in the model of Shacham and Waters [17], still it is arguably sound.

The discrepancy between the soundness definition in [17] and the work of [13, 19] springs from the fact that in practice to check whether a file is correctly stored at the cloud server, the verifier issues a polynomial number of POR queries to which the server has to respond correctly; otherwise, the verifier detects a corruption attack (the corruption attack could either be malicious or accidental) and flags the server as malicious. This is actually what the PORs of [13, 19] and **StealthGuard** aim to capture. In order to remedy this shortcoming, we propose augmenting the definition of Shacham and Waters [17] (as will be shown in Algorithm 2) with an additional parameter γ that quantifies the number of POR queries that verifier should issue to either be sure that a file is retrievable or to detect a corruption attack on that file.

Now in accordance with [17], we first formalize *soundness* using a game that describes the capabilities of an adversary $\mathcal{A}$ (i.e., malicious cloud server) which can deviate arbitrarily from the POR protocol, and then we define the extractor algorithm $\mathcal{E}$.

To formally capture the capabilities of adversary $\mathcal{A}$, we assume that it has access to the following oracles:

- $\mathcal{O}_{\mathsf{Encode}}$: This oracle takes as inputs a file F and the client's key K, and returns a file identifier fid and a verifiable version $\hat{F}$ of F that will be outsourced to $\mathcal{A}$. Note that adversary $\mathcal{A}$ can corrupt the outsourced file $\hat{F}$ either by modifying or deleting $\hat{F}$'s blocks.
- $\mathcal{O}_{\mathsf{Challenge}}$: On input of a file identifier fid and client's key K, the oracle $\mathcal{O}_{\mathsf{Challenge}}$ returns a POR challenge chal to adversary $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Verify}}$: When queried with client's key K, a file identifier fid, a challenge chal and a proof of retrievability $\mathcal{P}$, the oracle $\mathcal{O}_{\mathsf{Verify}}$ returns bit b such that: $b = 1$ if $\mathcal{P}$ is a valid proof of retrievability, and $b = 0$ otherwise.

Adversary $\mathcal{A}$ accesses the aforementioned oracles in two phases: a learning phase and a challenge phase. In the learning phase, adversary $\mathcal{A}$ can call oracles $\mathcal{O}_{\mathsf{Encode}}$, $\mathcal{O}_{\mathsf{Challenge}}$, and $\mathcal{O}_{\mathsf{Verify}}$ for a polynomial number of times in any interleaved order as depicted in Algorithm 1. Then, at the end of the learning phase, the adversary $\mathcal{A}$ specifies a file identifier fid^* that was already output by oracle $\mathcal{O}_{\mathsf{Encode}}$.

We note that the goal of adversary $\mathcal{A}$ in the challenge phase (cf. Algorithm 2) is to generate γ valid proofs of retrievability $\mathcal{P}_\backslash{}^*$ for file F^* associated with file identifier fid^*. To this end, adversary $\mathcal{A}$ first calls the oracle $\mathcal{O}_{\mathsf{Challenge}}$ that supplies $\mathcal{A}$ with γ challenges chal_i^*, then it responds to these challenges by outputting γ proofs $\mathcal{P}_i^*$. Now, on input of client's key K, file identifier fid^*challenges chal_i^* and proofs $\mathcal{P}_i^*$, oracle $\mathcal{O}_{\mathsf{Verify}}$ outputs γ bits b_i^*. Adversary $\mathcal{A}$ is said to be successful if $b^* = \bigwedge_{i=1}^{\gamma} b_i^* = 1$. That is, if $\mathcal{A}$ is able to generate γ proofs of retrievability $\mathcal{P}^*$ for file F^* that are accepted by oracle $\mathcal{O}_{\mathsf{Verify}}$.

Given the game described above and in line with [13, 17], we formalize the soundness of POR schemes through the definition of an extractor algorithm $\mathcal{E}$ that uses adversary $\mathcal{A}$ to recover/retrieve the file F^* by processing as follows:

– $\mathcal{E}$ takes as inputs the client's key K and the file identifier fid^*;
– $\mathcal{E}$ is allowed to initiate a polynomial number of POR executions with adversary $\mathcal{A}$ for the file F^*;
– $\mathcal{E}$ is also allowed to rewind adversary $\mathcal{A}$. This suggests in particular that extractor $\mathcal{E}$ can execute the challenge phase of the soundness game a polynomial number of times, while the state of adversary $\mathcal{A}$ remains unchanged.

Intuitively, a POR scheme is sound, if for any adversary $\mathcal{A}$ that wins the soundness game with a non-negligible probability δ, there exists an extractor algorithm $\mathcal{E}$ that succeeds in retrieving the challenge file F^* with an overwhelming probability. A probability is overwhelming if it is equal to $1 - \varepsilon$, where ε is negligible.

Algorithm 1: Learning phase of the soundness game

// $\mathcal{A}$ executes the following in any interleaved
// order for a polynomial number of times
$(\mathsf{fid}, \hat{F}) \leftarrow \mathcal{O}_{\mathsf{Encode}}(F, K)$;
$\mathsf{chal} \leftarrow \mathcal{O}_{\mathsf{Challenge}}(K, \mathsf{fid})$;
$\mathcal{P} \leftarrow \mathcal{A}$;
$b \leftarrow \mathcal{O}_{\mathsf{Verify}}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P})$;
// $\mathcal{A}$ outputs a file identifier fid^*
$\mathsf{fid}^* \leftarrow \mathcal{A}$;

Algorithm 2: Challenge phase of the soundness game

for $i = 1$ **to** γ **do**
  $\mathsf{chal}_i^* \leftarrow \mathcal{O}_{\mathsf{Challenge}}(K, \mathsf{fid}^*)$;
  $\mathcal{P}_i^* \leftarrow \mathcal{A}$;
  $b_i^* \leftarrow$
  $\mathcal{O}_{\mathsf{Verify}}(K, \mathsf{fid}_i^*, \mathsf{chal}_i^*, \mathcal{P}_i^*)$;
end
$b^* = \bigwedge_{i=1}^{\gamma} b_i^*$

Definition 2 (Soundness). *A POR scheme is said to be* (δ, γ)-sound*, if for every adversary $\mathcal{A}$ that provides γ valid proofs of retrievability in a row (i.e., succeeds in the soundness game described above) with a non-negligible probability δ, there exists an extractor algorithm $\mathcal{E}$ such that:*

$$Pr(\mathcal{E}(K, \mathsf{fid}^*) \rightarrow F^* \mid \mathcal{E}(K, \mathsf{fid}^*) \overset{\mathsf{interact}}{\longleftrightarrow} \mathcal{A}) \geq 1 - \varepsilon$$

Where ε is a negligible function in the security parameter τ.

The definition above could be interpreted as follows: if verifier $\mathcal{V}$ issues a sufficient number of queries ($\geq \gamma$) to which cloud server $\mathcal{S}$ responds correctly, then $\mathcal{V}$ can ascertain that $\mathcal{S}$ is still storing a retrievable version of file F^* with high probability. It should be noted that while γ characterizes the number of *valid* proofs of retrievability that $\mathcal{E}$ has to receive (successfully or in a row) to assert that file F^* is still retrievable, δ quantifies the number of operations that the extractor $\mathcal{E}$ has to execute and the amount of data that it has to download to first declare F^* as retrievable and then to extract it. Actually, the computation and the communication complexity of extractor $\mathcal{E}$ will be of order $O(\frac{\gamma}{\delta})$.

4 Overview

4.1 Idea

In **StealthGuard**, client $\mathcal{C}$ first injects some pseudo-randomly generated *watchdogs* into random positions in the encrypted data. Once data is outsourced, $\mathcal{C}$ launches lookup

queries to check whether the watchdogs are stored as expected by the cloud. By relying on a privacy-preserving word search (WS), we ensure that neither the cloud server $\mathcal{S}$ nor eavesdropping intruders can discover which watchdog was targeted by search queries. As a result, $\mathcal{C}$ can launch an unbounded number of POR queries (even for the same watchdog) without the need of updating the data with new watchdogs in the future. The responses are also obfuscated thanks to the underlying WS scheme. This ensures that the only case in which $\mathcal{S}$ returns a valid set of responses for the POR scheme is when it stores the entire file and executes the WS algorithm correctly (soundness property).

Besides, as in [13], in order to protect the data from small corruptions, **StealthGuard** applies an ECC that enables the recovery of the corrupted data. Substantial damage to the data is detected via the watchdog search.

4.2 StealthGuard phases

A client $\mathcal{C}$ uploads to the cloud server $\mathcal{S}$ a file F which consists of n splits $\{S_1, ..., S_n\}$. Thereafter a verifier $\mathcal{V}$ checks the retrievability of F using **StealthGuard**.

The protocol is divided into three phases:

- *Setup*: During this phase, client $\mathcal{C}$ performs some transformations over the file and inserts a certain number of watchdogs in each split. The resulting file is sent to cloud server $\mathcal{S}$.
- *WDSearch*: This phase consists in searching for some watchdog w in a privacy-preserving manner. Hence, verifier $\mathcal{V}$ prepares and sends a lookup query for w; the cloud $\mathcal{S}$ in turn processes the relevant split to generate a correct response to the search and returns the output to $\mathcal{V}$.
- *Verification*: Verifier $\mathcal{V}$ checks the validity of the received response and makes the decision about the existence of the watchdog in the outsourced file.
  We note that if $\mathcal{V}$ receives at least γ (γ is a threshold determined in Section 6.2) correct responses from the cloud, then it can for sure decide that F is retrievable. On the other hand, if $\mathcal{V}$ receives one response that is not valid, then it is convinced either the file is corrupted or even lost.

5 StealthGuard

This section details the phases of the protocol. Table 1 sums up the notation used in the description. We also designed a dynamic version of **StealthGuard** that allows efficient POR even when data is updated. Due to space limitations, we only present in Section 5.4 an overview of dynamic **StealthGuard**.

5.1 Setup

This phase prepares a verifiable version $\hat{F}$ of file $F = \{S_1, S_2, ..., S_n\}$. Client $\mathcal{C}$ first runs the KeyGen algorithm to generate the master secret key K. It derives $n + 3$ additional keys, used for further operations in the protocol: $K_{enc} = H_{enc}(K)$, $K_{wdog} = H_{wdog}(K)$, $K_{permF} = H_{permF}(K)$ and for $i \in [\![1, n]\!]$, $K_{permS,i} = H_{permS}(K, i)$

Index	Description
n	number of splits S_i in F
m	number of blocks in a split S_i
D	number of blocks in an encoded split $\tilde{S}_i$
v	number of watchdogs in one split
C	number of blocks in a split $\hat{S}_i$ with watchdogs
i	index of a split $\in [\![1, n]\!]$
k	index of a block in $\hat{S}_i \in [\![1, C]\!]$
j	index of a watchdog $\in [\![1, v]\!]$
l	size of a block
p	index of a block in $\tilde{F} \in [\![1, n \cdot D]\!]$
q	number of cloud's matrices
κ	index of a cloud's matrix $\in [\![1, q]\!]$
(s, t)	size of cloud's matrices
(x, y)	coordinates in a cloud's matrix $\in [\![1, s]\!] \times [\![1, t]\!]$

Table 1: Notation used in the description of StealthGuard

with H_{enc}, H_{wdog}, H_{permF} and H_{permS} being four cryptographic hash functions. K is the single information stored at the client.

Once all keying material is generated, $\mathcal{C}$ runs the Encode algorithm which first generates a pseudo-random and unique file identifier fid for file F, and then processes F as depicted in Figure 1.

1. **Error correcting**: The error-correcting code (ECC) assures the protection of the file against small corruptions. This step applies to each split S_i an ECC that operates over l-bit symbols. It uses an efficient $[m+d-1, m, d]$-ECC, such as Reed-Solomon codes [16], that has the ability to correct up to $\frac{d}{2}$ errors[4]. Each split is expanded with $d-1$ blocks of redundancy. Thus, the new splits are made of $D = m+d-1$ blocks.
2. **File block permutation**: **StealthGuard** applies a pseudo-random permutation to permute all the blocks in the file. This operation conceals the dependencies between the original data blocks and the corresponding redundancy blocks within a split. Without this permutation, the corresponding redundancy blocks are just appended to this split. An attacker could for instance delete all the redundancy blocks and a single data block from this split and thus render the file irretrievable. Such an attack would not easily be detected since the malicious server could still be able to respond with valid proofs to a given POR query targeting other splits in the file. The permutation prevents this attack since data blocks and redundancy blocks are mixed up among all splits. Let $\Pi_F : \{0,1\}^\tau \times [\![1, n \cdot D]\!] \rightarrow [\![1, n \cdot D]\!]$ be a pseudo-random permutation: for each $p \in [\![1, n \cdot D]\!]$, the block at current position p will be at position $\Pi_F(K_{permF}, p)$ in the permuted file that we denote $\tilde{F}$. $\tilde{F}$ is then divided into n splits $\{\tilde{S}_1, \tilde{S}_2, ..., \tilde{S}_n\}$ of equal size D.
3. **Encryption**: **StealthGuard** uses a semantically secure encryption E that operates over l-bit blocks[5] to encrypt the data. An encryption scheme like AES in counter mode [10] can be used. The encryption E is applied to each block of $\tilde{F}$ using K_{enc}.

[4] d is even

[5] Practically, l will be 128 or 256 bits.

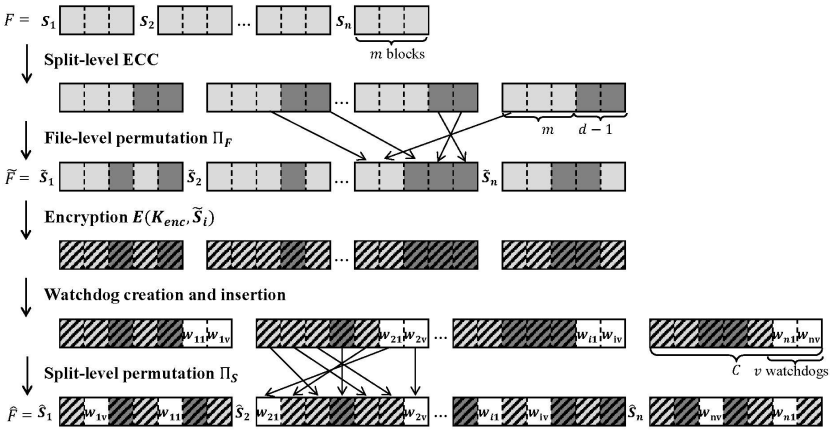

Fig. 1: Setup phase in StealthGuard

4. **Watchdog creation**: For each encrypted split, v l-bit watchdogs are generated using a pseudo-random function $\Phi : \{0,1\}^\tau \times [\![1,n]\!] \times [\![1,v]\!] \times \{0,1\}^* \rightarrow \{0,1\}^l$. Hence, for $j \in [\![1,v]\!]$, $w_{i,j} = \Phi(K_{wdog}, i, j, \mathsf{fid})$. The use of fid guarantees that two different files belonging to the same client have different watchdogs. Since the watchdogs are pseudo-randomly generated and the blocks in the split are encrypted, a malicious cloud cannot distinguish watchdogs from data blocks.
5. **Watchdog insertion**: The v watchdogs are appended to each split. Let $C = D + v$ be the size of the new splits. A split-level pseudo-random permutation $\Pi_S : \{0,1\}^\tau \times [\![1,C]\!] \rightarrow [\![1,C]\!]$ is then applied to the blocks within the same split in order to randomize the location of the watchdogs: for $i \in [\![1,n]\!]$, the block at current position k will be at position $\Pi_S(K_{permS,i}, k)$ in the permuted split. Note that in practice, the permutation is only applied to the last v blocks: for $k \in [\![D,C]\!]$, this step swaps block at current position k for block at position $\Pi_S(K_{permS,i}, k)$. We denote $\hat{S}_i$, $i \in [\![1,n]\!]$, the permuted split and $\hat{b}_{i,k}$, $k \in [\![1,C]\!]$ its blocks.

These operations yield file $\hat{F}$. The client uploads the splits $\{\hat{S}_i\}_{i=1}^n$ and fid to the cloud.

5.2 WDSearch

Verifier $\mathcal{V}$ wants to check the retrievability of F. Hence, it issues lookup queries for randomly selected watchdog, one watchdog for one split in one query. Cloud server $\mathcal{S}$ processes these queries without knowing what the values of the watchdogs are and where they are located in the splits. We propose WDSearch, a privacy-preserving WS solution derived from PRISM in [6]. Our proposal is a simpler version of PRISM and improves its performance in the particular context of **StealthGuard**. Note that this proposed building block is only an example and any existing privacy-preserving WS mechanism assuring the confidentiality of both the query and the result can be used in **StealthGuard**. PRISM and thus WDSearch are based on Private Information Retrieval (PIR). To process a query, $\mathcal{S}$ constructs q (s,t)-binary matrices such that $s \cdot t = C$. Each element in the matrices is filled with the witness (a very short information) of the corresponding block in the split. Based on the PIR query sent by the verifier, the server

retrieves in the matrices the witnesses corresponding to the requested watchdogs. We insist on the fact that WDSearch is not a PIR solution: the server does not retrieve the watchdog itself but only the witness.

WDSearch consists of two steps:

- **WDQuery**: Verifier $\mathcal{V}$ executes the Challenge algorithm to generate a challenge chal that is transmitted to cloud server $\mathcal{S}$. Challenge takes as input master key K and file identifier fid and it is executed in three phases. In the first phase, Challenge randomly selects a split index i and a watchdog index j ($i \in [\![1, n]\!]$ and $j \in [\![1, v]\!]$), and computes the position $\mathsf{pos_j}$ of the watchdog $w_{i,j}$ in the split $\hat{S}_i$ by applying the permutation performed during the watchdog insertion step: $\mathsf{pos_j} = \Pi_S(K_{permS,i}, D + j)$. Then, Challenge maps the position $\mathsf{pos_j}$ to a unique position (x_j, y_j) in an (s, t)-matrix:

  $$x_j = \lceil \frac{\mathsf{pos_j}}{t} \rceil \qquad\qquad y_j = \mathsf{pos_j} - \lceil \frac{\mathsf{pos_j}}{t} \rceil \times t + t$$

  In the second phase, given (x_j, y_j) and using any efficient PIR algorithm, Challenge computes a PIR query, denoted WitnessQuery, to retrieve the witness (and not the watchdog) at position (x_j, y_j) in the matrix. In the last phase, Challenge generates a random number r (this nonce will be used by the cloud when filling the binary matrices to guarantee freshness), and outputs the challenge $\mathsf{chal} = (\mathsf{WitnessQuery}, r, i)$. Eventually, verifier $\mathcal{V}$ sends the challenge chal and file identifier fid to cloud server $\mathcal{S}$.
- **WDResponse**: Upon receiving the challenge $\mathsf{chal} = (\mathsf{WitnessQuery}, r, i)$ and file identifier fid, cloud server $\mathcal{S}$ runs ProofGen to process the query. The cloud creates q binary matrices of size (s, t). For each block $\hat{b}_{i,k}$ in $\hat{S}_i$, the cloud computes $h_{i,k} = H(\hat{b}_{i,k}, r)$, where $k \in [\![1, C]\!]$. Here, H denotes a cryptographic hash function. The use of r forces the cloud to store the actual data block. Otherwise it could drop the block, only store the hash and respond to the query using that hash. Let $h_{i,k}|_q$ be the first q bits of $h_{i,k}$. For $\kappa \in [\![1, q]\!]$, let $\mathcal{M}_\kappa$ be one of the matrices created by the cloud. It fills the κ^{th} matrix with the κ^{th} bit of $h_{i,k}|_q$ as Algorithm 3 shows. It should be noted that according to the assignment process described in Algorithm 3, the witness at position (x_j, y_j) in $\mathcal{M}_\kappa$ is associated with watchdog $w_{i,j}$: it is the κ^{th} bit of $H(w_{i,j}, r)$.
  Once all the q binary matrices are filled, the cloud processes WitnessQuery by executing a PIR operation that retrieves one bit from each matrix $\mathcal{M}_\kappa$, $\kappa \in [\![1, q]\!]$. We denote $\mathsf{WitnessResponse}_\kappa$ the result of the PIR on matrix $\mathcal{M}_\kappa$. The ProofGen algorithm outputs $\mathcal{P}$, i.e. the proof of retrievability that consists in the set $\mathcal{P} = \{\mathsf{WitnessResponse}_1, ..., \mathsf{WitnessResponse}_q\}$. Cloud server $\mathcal{S}$ sends the proof $\mathcal{P}$ to verifier $\mathcal{V}$.

5.3 Verification

Verifier $\mathcal{V}$ runs ProofVerif to analyze the received proof $\mathcal{P}$. This algorithm takes as input master key K, proof $\mathcal{P}$, split index i, watchdog index j, and file identifier fid. ProofVerif outputs a bit equal to 1 if the proof is valid or 0 otherwise.

Algorithm 3: Filling the cloud matrices

```
// For a given (s, t)-matrix M_κ, a given split Ŝ_i and a given random number r
// k is the index of a block in split Ŝ_i
k = 1;
for x = 1 to s do
    for y = 1 to t do
        M_κ[x, y] ← κ^th bit of H(b̂_{i,k}, r);
        k = k + 1;
    end
end
```

$\mathcal{V}$ processes the q $\mathsf{WitnessResponse}_\kappa$ in order to retrieve the q bits ϵ_κ at position (x_j, y_j) in the matrix $\mathcal{M}_\kappa$, for $\kappa \in [\![1, q]\!]$. Let h denote $\epsilon_1\epsilon_2...\epsilon_q$.

We recall that verifier $\mathcal{V}$ queried watchdog $w_{i,j}$ for split $\hat{S}_i$ and that by having access to the master key K, $\mathcal{V}$ can recompute the value of $w_{i,j} = \Phi(K_{wdog}, i, j, \mathsf{fid})$ and its position in the split $\hat{S}_i$, $\mathsf{pos_j} = \Pi_S(K_{permS,i}, D + j)$. Thereafter, $\mathcal{V}$ computes the hash of the watchdog $h_{i,\mathsf{pos_j}} = H(w_{i,j}, r)$, with the same r chosen during the challenge and considers the q first bits of $h_{i,\mathsf{pos_j}}$. Based on the value of $h = \epsilon_1\epsilon_2...\epsilon_q$ and $h_{i,\mathsf{pos_j}}$, $\mathcal{V}$ checks whether $h = h_{i,\mathsf{pos_j}}|_q$. If it is the case, then $\mathcal{V}$ judges the proof valid and returns 1, otherwise it interprets the invalid proof as the occurrence of an attack and outputs 0.

As mentioned in section 4.2, in order to acknowledge the retrievability of F, verifier $\mathcal{V}$ needs to initiate at least γ POR queries[6] from randomly selected splits in order to either ascertain that F is retrievable or detect a corruption attack: if $\mathcal{V}$ receives γ valid POR responses, then it can conclude that cloud server $\mathcal{S}$ stores a retrievable version of F, otherwise, it concludes that $\mathcal{S}$ has corrupted part of the file.

5.4 Dynamic StealthGuard

The previously described protocol does not consider update operations that the client can perform over its data. Similarly to the work in [2, 8, 9, 11, 15, 18, 19, 21, 22, 24], we propose a scheme that handles these updates. Due to space limitations we present only an idea of how dynamic **StealthGuard** operates. Any update in the data impacts the security of our protocol. For example, if the client modifies the same block several times then the cloud can discover that this particular block is not a watchdog. Therefore, dynamic **StealthGuard** updates the watchdogs in a split each time an update occurs on that split. Besides, the verifier must be ensured that the file stored at the server is actually the latest version. Dynamic **StealthGuard** offers a versioning solution to assure that the cloud always correctly applies the required update operations and that it always stores the latest version of the file. Our proposal uses Counting Bloom Filters [12] and Message Authentication Codes (MAC) [5]. Each time a split is updated, some information regarding the split number and the version number is added into the counting Bloom filter which is authenticated using a MAC that can only be computed by the client and the verifier. Additionally, to guarantee the freshness of the response at each

[6] The value of γ will be determined in Section 6.2.

update query, a new MAC key is generated. This protocol does not imply any additional cost at the verifier except of storing an additional MAC symmetric key.

Another challenging issue is that updating a data block requires to update the corresponding redundancy blocks, resulting in the disclosure to the cloud server of the dependencies between the data blocks and the redundancy blocks. Therefore, the file permutation in the *Setup* phase becomes ineffective. Some techniques are available to conceal these dependencies such as batch updates [19] or oblivious RAM [8]. However, these approaches are expensive in terms of computation and communication costs. Hence, we choose to trade off between POR security and update efficiency by omitting the file permutation.

6 Security Analysis

In this section, we state the security theorems of **StealthGuard**.

6.1 Completeness

Theorem 1. ***StealthGuard*** *is complete.*

Proof. Without loss of generality, we assume that the honest verifier $\mathcal{V}$ runs a POR for a file F. To this end, verifier $\mathcal{V}$ sends a challenge $\mathsf{chal} = (\mathsf{WitnessQuery}, r, i)$ for watchdog $w_{i,j}$, and the file identifier fid of F. Upon receiving challenge chal and file identifier fid, the cloud server generates a proof of retrievability $\mathcal{P}$ for F.

According to **StealthGuard**, the verification of POR consists of first retrieving the first q bits of a hash h_{i,pos_j}, then verifying whether $h_{i,\mathsf{pos}_j}|q$ corresponds to the first q-bits of the hash $H(w_{i,j}, r)$. Since the cloud server $\mathcal{S}$ is honest, then this entails that it stores $w_{i,j}$, and therewith, can always compute $h_{i,\mathsf{pos}_j} = H(w_{i,j}, r)$.

Consequently, $\mathsf{ProofVerif}(K, \mathsf{fid}, \mathsf{chal}, \mathcal{P}) = 1$.

6.2 Soundness

As in Section 5, we assume that each split S_i in a file F is composed of m blocks, and that the Encode algorithm employs a $[D, m, d]$-ECC that corrects up to $\frac{d}{2}$ errors per split (i.e., $D = m + d - 1$). We also assume that at the end of its execution, the Encode algorithm outputs the encoded file $\hat{F}$ which consists of a set of splits $\hat{S}_i$ each comprising $C = (D + v)$ blocks (we recall that v is the number of watchdogs per split).

In the following, we state the main security theorem for **StealthGuard**.

Theorem 2. *Let τ be the security parameter of* **StealthGuard** *and let ρ denote $\frac{d}{2D}$.*

StealthGuard is (δ, γ)-sound in the random oracle model, for any $\delta > \delta_{\mathsf{neg}}$ and $\gamma \geq \gamma_{\mathsf{neg}}$, where

$$\delta_{\mathsf{neg}} = \frac{1}{2^{\tau}}$$

$$\gamma_{\mathsf{neg}} = \lceil \frac{\mathsf{ln}(2)\tau}{\rho_{\mathsf{neg}}} \rceil$$

$$(1 - \frac{\rho}{\rho_{\mathsf{neg}}})^2 \rho_{\mathsf{neg}} = \frac{3\mathsf{ln}(2)\tau}{D} \text{ and } \rho_{\mathsf{neg}} \leq \rho$$

Actually if $\gamma \geq \gamma_{\mathsf{neg}}$, then there exists an extractor $\mathcal{E}$ that recovers a file F with a probability $1-\frac{n}{2^\tau}$, such that n is the number of splits in F, by interacting with an adversary $\mathcal{A}$ against StealthGuard who succeeds in the soundness game with a probability $\delta > \frac{1}{2^\tau}$.

Due to space limitations, a proof sketch of this theorem is provided in our long report [4]. We note that the results derived above can be interpreted as follows: if verifier $\mathcal{V}$ issues $\gamma \geq \gamma_{\mathsf{neg}}$ POR queries for some file F to which the cloud server $\mathcal{S}$ responds correctly, then $\mathcal{V}$ can declare F as retrievable with probability $1 - \frac{n}{2^\tau}$. Also, we recall that a POR execution for a file F in **StealthGuard** consists of fetching (obliviously) a witness of a watchdog from the encoding $\hat{F}$ of that file. Consequently, to ensure a security level of $\frac{1}{2^\tau}$, the client $\mathcal{C}$ must insert at least γ_{neg} watchdogs in F. That is, if file F comprises n splits, then $nv \geq \gamma_{\mathsf{neg}}$ (v is the number of watchdogs per split).

7 Discussion

StealthGuard requires the client to generate $v > \frac{\gamma_{\mathsf{neg}}}{n}$ watchdogs per split where n is the number of splits and γ_{neg} is the threshold of the number of queries that verifier $\mathcal{V}$ should issue to check the retrievability of the outsourced data. As shown in Theorem 2, this threshold does not depend on the size of data (in bytes). Instead, γ_{neg} is defined solely by the security parameter τ, the number $D = m + d - 1$ of data blocks and redundancy block per split and the rate $\rho = \frac{d}{2D}$ of errors that the underlying ECC can correct. Namely, γ_{neg} is inversely proportional to both D and ρ. This means that by increasing the number of blocks D per split or the *correctable* error rate ρ, the number of queries that the client should issue decreases. However, having a large ρ would increase the size of data that client $\mathcal{C}$ has to outsource to cloud server $\mathcal{S}$, which can be inconvenient for the client. Besides, increasing D leads to an increase of the number of blocks $C = s \cdot t$ per split $\hat{S}_i$ which has a direct impact on the communication cost and the computation load *per query* at both the verifier $\mathcal{V}$ and the cloud server $\mathcal{S}$. It follows that when defining the parameters of **StealthGuard**, one should consider the tradeoff between the affordable storage cost and the computation and communication complexity per POR query.

To enhance the computation performances of **StealthGuard**, we suggest to use the **Trapdoor Group Private Information Retrieval** which was proposed in [20] to implement the PIR instance in WDSearch. This PIR enables the verifier in **StealthGuard** to fetch a row from an (s, t) matrix (representing a split) without revealing to the cloud which row the verifier is querying. One important feature of this PIR is that it only involves random number generations, additions and multiplications in $\mathbb{Z}_p$ (where p is a prime of size $|p| = 200$ bits) which are not computationally intensive and could be performed by a lightweight verifier. In addition, we emphasize that PIR in **StealthGuard** is not employed to retrieve a watchdog, but rather to retrieve a q-bit hash of the watchdog (typically $q = 80$), and that it is not performed on the entire file, but it is instead executed over a split. Finally, we indicate that when employing **Trapdoor Group Private Information Retrieval**, the communication cost of **StealthGuard** is minimal when $s \simeq \sqrt{Cq}$ and $t \simeq \sqrt{\frac{C}{q}}$. This results in a computation and a communication complexity (per query) at the verifier of $O(\sqrt{Cq})$ and a computation and communication complexity at the server of $O(C)$ and $O(\sqrt{Cq})$ respectively.

Example. A file F of 4GB is divided into $n = 32768$ splits $F = \{S_1, S_2, ..., S_n\}$, and each split S_i is composed of 4096 blocks of size 256 bits. **StealthGuard** inserts 8 watchdogs per split and applies an ECC that corrects up to 228 corrupted blocks (i.e., $\rho = 5\%$). We obtain thus $\hat{F} = \{\hat{S}_1, \hat{S}_2, ..., \hat{S}_n\}$, where $\hat{S}_i$ is composed of 4560 blocks of size 256 bits. This results in a redundancy of $\simeq 11.3\%$, where 11.1% redundancy is due to the use of ECC, and 0.20% redundancy is caused by the use of watchdogs.

If $(s, t) = (570, 8)$, $q = 80$ and **StealthGuard** implements the Trapdoor Group PIR [20] where $|p| = 200$ bits, then the verifier's query will be of size $\simeq 13.9$ KB, whereas the cloud server's response will be of size $\simeq 15.6$KB. In addition, if the cloud server still stores the file $\hat{F}$, then the verifier will declare the file as retrievable with probability $1 - \frac{n}{2^{60}} \simeq 1 - \frac{1}{2^{45}}$ by executing the POR protocol 1719 times. That is, by downloading 26.2MB which corresponds to 0.64% of the size of the original file F.

8 Related Work

The approach that is the closest to **StealthGuard** is the sentinel-based POR introduced by Juels and Kaliski [13]. As in **StealthGuard**, before outsourcing the file to the server, the client applies an ECC and inserts in the encrypted file special blocks, *sentinels*, that are indistinguishable from encrypted blocks. However, during the challenge, the verifier asks the prover for randomly-chosen sentinels, disclosing their positions and values to the prover. Thus, this scheme suggests a limited number of POR queries. Therefore, the client may need to download the file in order to insert new sentinels and upload it again to the cloud. [13] mentions, without giving any further details, a PIR-based POR scheme that would allow an unlimited number of challenges by keeping the positions of sentinels private, at the price of high computational cost equivalent in practice to downloading the entire file. In comparison, **StealthGuard** uses a PIR within the WS technique to retrieve a witness of the watchdog (a certain number of bits instead of the entire watchdog), and does not limit the number of POR verifications.

Ateniese et al. [1] define the concept of Provable Data Possession (PDP), which is weaker than POR in that it assures that the server possesses parts of the file but does not guarantee its full recovery. PDP uses RSA-based homomorphic tags as check-values for each file block. To verify possession, the verifier asks the server for tags for randomly chosen blocks. The server generates a proof based on the selected blocks and their respective tags. This scheme provides public verifiability meaning that any third party can verify the retrievability of a client's file. However, this proposal suffers from an initial expensive tag generation leading to high computational cost at the client. The same authors later propose in [3] a *robust auditing* protocol by incorporating erasure codes in their initial PDP scheme [1] to recover from small data corruption. To prevent an adversary from distinguishing redundancy blocks from original blocks, the latter are further permuted and encrypted. Another permutation and encryption are performed on the redundancy blocks only which are then concatenated to the file. This solution suffers from the fact that a malicious cloud can selectively delete redundant blocks and still generate valid proofs. Even though these proofs are valid, they do not guarantee that the file is retrievable.

Shacham and Waters in [17] introduce the concept of Compact POR. The client applies

Scheme	Parameter	Setup cost	Storage overhead	Server cost	Verifier cost	Communication cost
Robust PDP [3]	block size: 2 KB tag size: 128 B	4.4×10^6 exp 2.2×10^6 mul	tags: 267 MB	764 PRP 764 PRF 765 exp 1528 mul	challenge: 1 exp verif: 766 exp 764 PRP	challenge: 168 B response: 148 B
JK POR [13]	block size: 128 bits number of sentinels: 2×106	2×10^6 PRF	sentinels: 30.6 MB	$\perp$	challenge: 1719 PRP verif: $\perp$	challenge: 6 KB response:26.9 MB
Compact POR [17]	block size: 80 bits number of blocks in one split: 160 tag size: 80 bits	1 enc 5.4×10^6 PRF 1.1×10^9 mul	tags: 51 MB	7245 mul	challenge: 1 enc, 1 MAC verif: 45 PRF, 160 + 205 mul	challenge: 1.9 KB response: 1.6 KB
Efficient POR [23]	block size: 160 bits number of blocks in one split: 160	2.2×10^8 mul 1.4×10^6 PRF	tags: 26 MB	160 exp $2.6 * 10^5$ mul	challenge:$\perp$ verif: 2 exp, 1639 PRF, 1639 mul	challenge: 36 KB response: 60 B
StealthGuard	block size: 256 bits number of blocks in one split: 4096	2.6×10^5 PRF 2.6×10^5 PRP	watchdogs: 8 MB	6.2×10^8 mul	challenge: 2.0×10^6 mul verif: 1.4×10^5 mul	challenge: 23.3 MB response: 26.2 MB

Table 2: Comparison of relevant related work with **StealthGuard**.

an erasure code and for each file block, it generates *authenticators* (similar to tags in [1]), with BLS signatures [7], for public verifiability, or with Message Authentication Codes (MAC) [5], for private verifiability. The generation of these values are computationally expensive. Moreover, the number of authenticators stored at the server is linear to the number of data blocks, leading to an important storage overhead. Xu and Chang [23] propose to enhance the scheme in [17] using the technique of polynomial commitment [14] which leads to light communication costs. These two schemes employ erasure codes in conjunction with authentication tags, which induces high costs at the time of retrieving the file. Indeed, erasure coding does not inform the verifier about the position of the corrupted blocks. Thus, the verifier has to check each tag individually to determine whether it is correct or not. When a tag is detected as invalid, meaning that the corresponding block is corrupted, the verifier applies the decoding to recover the original data block.

A recent work of Stefanov et al. [19], Iris, proposes a POR protocol over authenticated file systems subject to frequent changes. Each block of a file is authenticated using a MAC to provide file-block integrity which makes the tag generation very expensive.

Compared to all these schemes, **StealthGuard** performs computationally lightweight operations at the client, since the generation of watchdogs is less expensive than the generation of tags like in [1, 17]. In addition, the storage overhead induced by the storage of watchdogs is less important than in the previous work. At the cost of more bits transmitted during the POR challenge-response, **StealthGuard** ensures a better probability of detecting adversarial corruption.

Table 2 depicts the performance results of **StealthGuard** and compares it with previous work. We analyze our proposal compared to other schemes [3, 13, 17, 23] with respect to a file of size 4 GB. The comparison is made on the basis of the POR assur-

ance of $1 - \frac{1}{2^{45}}$ computed in Section 7. We assume that all the compared schemes have three initial operations in the *Setup* phase: the application of an ECC, the encryption and the file-level permutation of data and redundancy blocks. Since these three initial operations have comparable costs for all the schemes, we omit them in the table. Computation costs are represented with `exp` for exponentiation, `mul` for multiplication, `PRF` for pseudo-random function or `PRP` for pseudo-random permutation. For **StealthGuard**, we compute the different costs according to the values provided in Section 7. For the other schemes, all initial parameters derive from the respective papers. In [17] since the information on the number of blocks in a split is missing, we choose the same one as in [23]

Setup. In our scheme, the client computes $32768 \times 8 \approx 2.6 \times 10^5$ PRF and 2.6×10^5 PRP for the generation and the insertion of watchdogs. One of the advantages of **StealthGuard** is having a more lightweight setup phase when the client preprocesses large files. Indeed, the setup phase in most of previous work [3, 17, 19, 23] requires the client to compute an authentication tag for each block of data in the file which is computationally demanding in the case of large files.

Storage Overhead. The insertion of watchdogs in **StealthGuard** induces a smaller storage overhead compared to other schemes that employ authentication tags.

Proof Generation and Verification. For **StealthGuard**, we consider the PIR operations as multiplications of elements in $\mathbb{Z}_p$ where $|p| = 200$ bits. To get the server and verifier computational costs of existing work, based on the parameters and the bounds given in their respective papers, we compute the number of requested blocks in one challenge to obtain a probability of $1 - \frac{1}{2^{45}}$ to declare the file as irretrievable: 764 blocks in [3], 1719 sentinels in [13], 45 blocks in [17] and 1639 blocks in [23]. **StealthGuard** induces high cost compared to existing work but is still acceptable.

Communication. Even if its communication cost is relatively low compared to **StealthGuard**, JK POR [13] suffers from the limited number of challenges, that causes the client to download the whole file to regenerate new sentinels. Although we realize that **StealthGuard**'s communication cost is much higher than [3, 17, 23], such schemes would induce additional cost at the file retrieval step, as mentioned earlier.

To summarize, **StealthGuard** trades off between light computation at the client, small storage overhead at the cloud and significant but still acceptable communication cost. Nevertheless, we believe that **StealthGuard**'s advantages pay off when processing large files. The difference between the costs induced by existing schemes and those induced by **StealthGuard** may become negligible if the size of the outsourced file increases.

9 Conclusion

StealthGuard is a new POR scheme which combines the use of randomly generated watchdogs with a lightweight privacy-preserving word search mechanism to achieve high retrievability assurance. As a result, a verifier can generate an unbounded number of queries without decreasing the security of the protocol and thus without the need for updating the watchdogs. **StealthGuard** has been proved to be complete and sound.

As future work, we plan to implement **StealthGuard** in order to not only evaluate its efficiency in a real-world cloud computing environment but also to define optimal values for system parameters.

10 Acknowledgment

This work was partially funded by the Cloud Accountability project - A4Cloud (grant EC 317550).

References

[1] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 598–609. ACM, 2007. ISBN 978-1-59593-703-2. URL `http://dblp.uni-trier.de/db/conf/ccs/ccs2007.html`.

[2] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication networks*, SecureComm '08, pages 9:1–9:10, New York, NY, USA, 2008. ACM.

[3] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12, 2011.

[4] Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva, and Melek Önen. Stealthguard: Proofs of retrievability with hidden watchdogs. Technical report, EURECOM, June 2014. URL `http://www.eurecom.fr/publication/4338`.

[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Proceedings of the 16th Annual International Cryptology conference on Advances in Cryptology, CRYPTO'96*, pages 1–15. LNCS, August 1996.

[6] Erik-Oliver Blass, Roberto di Pietro, Refik Molva, and Melek Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. LNCS, July 2012.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *J. Cryptology*, 17(4):297–319, September 2004.

[8] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious ram. In *EUROCRYPT*, pages 279–295, 2013.

[9] Bo Chen and Reza Curtmola. Robust dynamic provable data possession. In *ICDCS Workshops*, pages 515–525, 2012.

[10] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. National Institute of Standards and Technology. Special Publication 800-38A, 2001.

[11] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-894-0. doi: 10.1145/1653662.1653688. URL `http://doi.acm.org/10.1145/1653662.1653688`.

[12] Li Fan, Pei Cao, Jurassa Almeida, and Andrei Z. Broder. Summary Cache: a Scalable Wide-Area Web Cache Sharing Protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000. ISSN 1063-6692.

[13] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007. ISBN 978-1-59593-703-2. URL http://dblp.uni-trier.de/db/conf/ccs/ccs2007.html.

[14] Aniket Kate, Gregory Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. *Advances in Cryptology-ASIACRYPT 2010*, pages 177–194, 2010.

[15] Zhen Mo, Yian Zhou, and Shigang Chen. A dynamic proof of retrievability (por) scheme with o(logn) complexity. In *ICC*, pages 912–916, 2012.

[16] Irving S. Reed and Gustave Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society of Industrial and Applied Mathematics*, 8(2):300–304, 06/1960 1960.

[17] Shacham, Hovav and Waters, Brent. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89254-0. doi: 10.1007/978-3-540-89255-7_7. URL http://dx.doi.org/10.1007/978-3-540-89255-7_7.

[18] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *ACM Conference on Computer and Communications Security*, pages 325–336, 2013.

[19] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: a scalable cloud file system with efficient integrity checks. In *ACSAC*, pages 229–238, 2012.

[20] Jonathan Trostle and Andy Parrish. Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups. In *Proceedings of Conference on Information Security*, pages 114–128, Boca Raton, USA, 2010.

[21] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 355–370, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(5):847–859, 2011.

[23] Jia Xu and Ee-Chien Chang. Towards efficient proofs of retrievability. In *ASIACCS*, pages 79–80, 2012.

[24] Qingji Zheng and Shouhuai Xu. Fair and dynamic proofs of retrievability. In *CODASPY*, pages 237–248, 2011.

PerfectDedup: Secure Data Deduplication

Pasquale Puzio[1,2], Refik Molva[2], Melek Önen[2], Sergio Loureiro[1]

[1] SecludIT, Sophia Antipolis, FRANCE
{pasquale,sergio}@secludit.com,
http://secludit.com
[2] EURECOM, Sophia Antipolis, FRANCE
{puzio,molva,onen}@eurecom.fr,
http://www.eurecom.fr

Abstract. With the continuous increase of cloud storage adopters, data deduplication has become a necessity for cloud providers. By storing a unique copy of duplicate data, cloud providers greatly reduce their storage and data transfer costs. Unfortunately, deduplication introduces a number of new security challenges. We propose PerfectDedup, a novel scheme for secure data deduplication, which takes into account the popularity of the data segments and leverages the properties of Perfect Hashing in order to assure block-level deduplication and data confidentiality at the same time. We show that the client-side overhead is minimal and the main computational load is outsourced to the cloud storage provider.

Keywords: cloud,storage,deduplication,confidentiality,encryption,security, perfect hashing

1 Introduction

Cloud storage providers constantly look for techniques aimed to minimize redundant data and maximize space savings. We focus on deduplication, which is one of the most popular techniques and has been adopted by many major providers such as Dropbox[3]. The idea behind deduplication is to store duplicate data only once. Thanks to such a mechanism, space savings can reach 70% [7] and even more in backup applications. On the other hand, along with low costs, users also require the confidentiality of their data through encryption. Unfortunately, deduplication and encryption are two conflicting techniques. A solution which has been proposed to meet these two conflicting requirements is Convergent Encryption (CE) [4] whereby the encryption key is the result of the hash of the data segment. However, CE unfortunately suffers from various

[2] Partially supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

[3] https://www.dropbox.com

well-known weaknesses [9] including dictionary attacks. We propose to counter the weaknesses due to CE by taking into account the popularity [10] of the data segments. Data segments stored by several users, that is, popular ones, are only protected under the weak CE mechanism whereas unpopular data segments that are unique in storage are protected under semantically-secure encryption. This declination of encryption mechanisms lends itself perfectly to efficient deduplication since popular data segments that are encrypted under CE are also the ones that need to be deduplicated. This scheme also assures proper security of stored data since sensitive thus unpopular data segments enjoy the strong protection thanks to the semantically-secure encryption whereas the popular data segments do not actually suffer from the weaknesses of CE since the former are much less sensitive because they are shared by several users. Nevertheless, this approach raises a new challenge: the users need to decide about the popularity of each data segment before storing it and the mechanism through which the decision is taken paves the way for a series of exposures very similar to the ones with CE. The focus of schemes based on popularity then becomes the design of a secure mechanism to detect the popularity of data segments.

In this paper we suggest a new scheme for the secure deduplication of encrypted data, based on the aforementioned popularity principle. The main building block of this scheme is an original mechanism for detecting the popularity of data segments in a perfectly secure way. Users can lookup for data segments in a list of popular segments stored by the Cloud Storage Provider (CSP) based on data segment identifiers computed with a Perfect Hash Function (PHF). Thanks to this technique, there is no information leakage about unpopular data segments and popular data segments are very efficiently identified. Based on this new popularity detection technique, our scheme achieves deduplication of encrypted data at block level in a perfectly secure manner. The advantages of our scheme can be summarized as follows:

- our scheme allows for storage size reduction by deduplication of popular data;
- our scheme relies on symmetric encryption algorithms, which are known to be very efficient even when dealing with large data;
- our scheme achieves deduplication at the level of blocks, which leads to higher storage space savings compared to file-level deduplication [7];
- our scheme does not require any coordination or initialization among users;
- our scheme does not incur any storage overhead for unpopular data blocks;

2 Secure Deduplication Based on Popularity

Given the inherent incompatibility between encryption and deduplication, existing solutions suffer from different drawbacks. CE was considered to be the most convenient solution for secure deduplication but it has been proved that is is vulnerable to various types of attacks [9]. Hence, CE cannot be employed to protect data confidentiality and thus stronger encryption mechanisms are required.

We point out that data may need different levels of protection depending on its popularity [10] a data segment becomes "popular" whenever it belongs to more than t users (where t is the popularity threshold). The "popularity" of a block is viewed as a trigger for its deduplication. Similarly, a data segment is considered to be unpopular if it belongs to less than t users. This is the case for all highly sensitive data, which are likely to be unique and thus unlikely to be duplicated.

Given this simple distinction, we observe that popular data do not require the same level of protection as unpopular data and therefore propose different forms of encryption for popular and unpopular data. For instance, if a file is easily accessible by anyone on the Internet, then it is reasonable to consider a less secure protection. On the other hand, a confidential file containing sensitive information, such as a list of usernames and passwords, needs much stronger protection. Popular data can be protected with CE in order to enable source-based deduplication, whereas unpopular data must be protected with a stronger encryption. Whenever an unpopular data segment becomes popular, that is, the threshold t is reached, the encrypted data segment is converted to its convergent encrypted form in order to enable deduplication.

We propose to encrypt unique and thus unpopular data blocks (which cannot be deduplicated) with a symmetric encryption scheme using a random key, which provides the highest level of protection while improving the computational cost at the client. Whenever a client wishes to upload a data segment, we propose that she should first discover its popularity degree in order to perform the appropriate encryption operation. The client may first lookup for a convergent encrypted version of the data stored at the CSP. If such data segment already exists, then the client discovers that this data segment is popular and hence can be deduplicated. If such data segment does not exist, the client will encrypt it with a symmetric encryption scheme. Such a solution would greatly optimize the encryption cost and the upload cost at the client. However, a standard lookup solution for the convergent encrypted data segment would reveal the convergent encrypted data segment ID, that is the digest of the data computed under an unkeyed hash function like SHA-3, which would be a serious breach. Secure lookup for a data segment is thus a delicate problem since the ID used as the input to the lookup query can lead to severe data leakage as explained in [17] and [9]. Therefore, in such a scenario the main challenge becomes how to enable the client to securely determine the popularity of a data segment without leaking any exploitable information to the CSP. Also, the client needs to securely handle the "popularity transition", that is the phase triggered by a data segment that has just reached the popularity threshold t. More formally, the popularity detection problem can be defined as follows: given a data segment D and its ID ID_D, the client wants to determine whether ID_D belongs to the set P of popular data segment IDs stored at an untrusted CSP. It is crucial that if $ID_D \notin P$, no information must be leaked to the CSP. More generally, this problem can be seen as an instance of the Private Set Intersection (PSI) problem [26]. However, existing solutions are known to be costly in terms of computation and communication,

especially when dealing with very large sets. Private Information Retrieval (PIR) [25] may also be a solution to this problem. However, using PIR raises two main issues: first, it would incur a significant communication overhead; second, PIR is designed to retrieve a single element per query, whereas an efficient protocol for the popularity check should allow to check the existence of multiple data segment IDs at once. Hence instead of complex cryptographic primitives like PSI and PIR we suggest a secure mechanism for popularity detection based on a lightweight building block called Perfect Hashing [11]. We aim at solving this problem by designing a novel secure lookup protocol, which is defined in next section, based on Perfect Hashing [11].

3 Basic Idea: Popularity Detection Based on Perfect Hashing

The popularity detection solution we propose makes use of the Perfect Hashing process which, given an input set of n data segments, finds a collision-free hash function, called the perfect hash function (PHF), that maps the input set to a set of m integers (m being larger than n by a given load factor). The CSP can run this process in order to generate the PHF matching the IDs of the convergent encrypted popular blocks that are currently stored at the CSP. The resulting PHF can be efficiently encoded into a file and sent to the client. Using the PHF received from the CSP, the client can lookup for new blocks in the set of encrypted popular block IDs stored at the CSP, as illustrated in Figure 1. For each new block D, the client first encrypts the block to get $CE(D)$, he then computes the ID thereof using an unkeyed hash function h like SHA-3. Finally, by evaluating the PHF over ID, the client gets the lookup index i for the new block. The integer i will be the input of the lookup query issued by the client. Once the CSP has received the lookup query containing i, he will return to the client the convergent encrypted popular block ID stored under i. At this point, the client can easily detect the popularity of his data segment by comparing the ID he computed with the one received from the CSP: if the two IDs match, then D is popular. As mentioned above, it is a crucial requirement to prevent the CSP from discovering the content of the block D when it is yet unpopular. We achieve so by introducing an enhanced and secure version of Perfect Hashing, which makes the generated PHF one-way, meaning that the CSP cannot efficiently derive the input of the PHF from its output i. This also implies that the PHF must yield well-distributed collisions for unpopular blocks.

However, even though the client is now able to securely detect the popularity of a block, he still needs to handle the popularity transition, that is the phase in which a block reaches the threshold t and the convergent encrypted block needs to be uploaded to the CSP. Since the client cannot be aware of other copies of the same block previously uploaded by other users, a mechanism to keep track of the unpopular data blocks is needed. Clearly, the client cannot rely on the CSP for this task, as the CSP is not a trusted component. Therefore, we propose to introduce a semi-trusted component called Index Service (IS), which is

responsible for keeping track of unpopular blocks. If the result of a popularity check is negative, then the client updates the IS accordingly by sending the popular convergent encrypted block ID and the ID of the symmetrically encrypted block. As soon as a block becomes popular, that is reaches the threshold t, the popularity transition is triggered and the client is notified in order to let him upload the convergent encrypted block, which from now on will be deduplicated by the CSP. Upon a popularity transition, the IS will delete from its storage any information related to the newly popular block. Regarding the popularity threshold, we point out that users do not have to be aware of its value, since the popularity transition is entirely managed by the IS, that is responsible for determining the current value for t. For instance, the value of t may be either static or dynamic, as proposed in [15]. Indeed, our scheme is completely independent of the strategy used for determining the value of the popularity threshold.

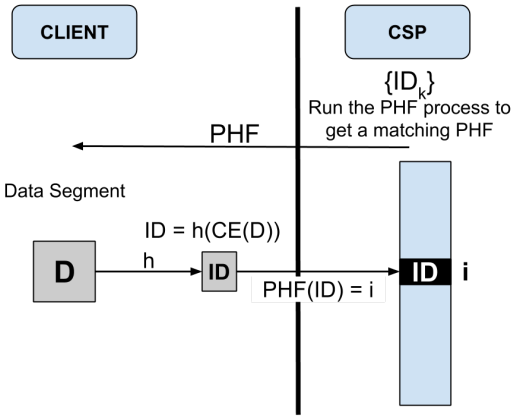

Fig. 1. The secure PHF allows users to detect popular blocks while preventing the CSP from discovering unpopular blocks

4 Background

4.1 Convergent Encryption

The idea of convergent encryption (CE) [4] is to derive the encryption key from the hash of the plaintext. A basic implementation of convergent encryption can be defined as follows: a user computes the encryption key using the message by applying a secure hash function H over M: $K = H(M)$; the message can then be encrypted with this key using a block cipher E: hence, $C = E(K, M) = E(H(M), M)$. Thanks to this technique, two users with two identical plaintexts will obtain two identical ciphertexts since the encryption key is the same and the

encryption algorithm is deterministic. Despite its practicality, CE is known to be vulnerable to several weaknesses which undermine its capability of protecting confidential data and allow an attacker who has access to the storage server to perform offline dictionary attacks and discover predictable files. As shown in [9], CE is unfortunately exposed to the two following attacks: confirmation-of-a-file (COF) and learn-the-remaining-information (LRI). These attacks exploit the deterministic relationship between the plaintext and the encryption key and therefore can be successful in the verification whether a given plaintext has already been stored.

4.2 Perfect Hashing

A Perfect Hash Function (PHF) maps a set of arbitrary entries into a set of integers without collisions. Authors in [11] proposed a new algorithm that allows finding a perfect mapping for very large sets in a very efficient way. This algorithm, which is called CHD (Compress, Hash and Displace), achieves linear space and computational complexities (with respect to the size of the set). The main idea behind this algorithm is to split the input set into several buckets (subsets) with a few elements and find a collision-free mapping for each of these buckets separately. This approach has proved to be much more scalable than previous approaches. The mean number of elements per bucket is a parameter that can be tuned upon executing the generation algorithm. CHD also allows choosing a load factor, which is the fraction of non-empty positions in the hash table.

Although perfect hashing is widely adopted for efficient indexing in the field of relational databases [19], it has some desirable properties which make it an appropriate building block for our scheme. First, the computational complexity to build the PHF is linear and the PHF can be evaluated in constant time. Thanks to these properties, the system is scalable since the PHF generation remains feasible when dealing with very large datasets. In addition to that, the main computational load is outsourced to the CSP, while the client only has to perform very simple and lightweight operations such as evaluating the PHF on block IDs and symmetrically encrypting data blocks. Second, thanks to a special encoding and compression mechanism, the size of the PHF file is small and therefore it can easily be transferred to the client. Therefore, the performance impact is minimal and this approach can easily scale up to sets of millions of elements. Third, the resulting hash table is collision-free with respect to the elements of the input set (popular block IDs), meaning that any index is associated to at most one element of the input set. On the other hand, if the PHF is evaluated over the rest of the domain (unpopular block IDs) then collisions are well-distributed. This property is an important starting point to build our secure lookup protocol which must guarantee that an attacker is not able to determine on what input the PHF has been evaluated. Indeed, while an index in the hash table corresponds to a unique popular block ID, many unpopular block IDs are mapped to the same index. Therefore, given an index in the hash table, the CSP cannot determine the corresponding block ID. In our solution we

propose to extend the existing PHF by replacing the underlying hash function with a one-way secure hash function such as SHA-3 [24]. Indeed, for the security of the scheme, it is crucial that the hash function used by the algorithm is one-way, meaning that it is easy to compute on a given input, but hard to invert given the image of a random input.

5 Our solution

5.1 Overview

We consider a scenario where users want to store their data (files) on a potentially untrusted Cloud Storage Provider (CSP) while taking advantage of source-based block-level deduplication and protecting the confidentiality of their data at the same time. Users run a client C which is a lightweight component with respect to both storage and computational capacity. CSP is assumed to be honest-but-curious and thus correctly stores users' data while trying to disclose the content thereof. Prior to uploading its data, C runs a secure lookup protocol to check whether the data are popular. The CSP is responsible for the generation of the PHF over the popular blocks and the storage of the resulting collision-free hash table. The proposed protocol introduces a trusted third party called Index Service (IS) which helps the client to discover the actual number of copies of a yet unpopular block. We stress the fact that IS only stores information on unpopular blocks and once a block becomes popular, all corresponding information are removed from its database, hence this component does not need to have a significant storage capacity.

The proposed solution is described under three different scenarios:

- Unpopular data upload (Scenario 1): if C finds out that the data is yet unpopular, it performs the upload to the CSP and updates the IS;
- Popularity transition (Scenario 2): if C finds out that the popularity degree of the data is $t-1$ (where t is the popularity threshold), then it performs the appropriate operations to upload the newly popular data. IS removes all information with respect to this specific data and CSP deletes all the encrypted copies previously stored;
- Popular data upload (Scenario 3): C only uploads metadata since it has detected that the requested data is popular, therefore deduplication can take place.

CSP stores a hash table for popular block IDs which is constructed with the previously introduced PHF. Each element of the hash table is defined by the couple $(PHF(h(CE(b_i))), h(CE(b_i)))$ where $h(CE(b_i))$ is the unkeyed secure hash of the convergent encrypted block. Before any operation, given the current set of popular blocks, CSP creates a corresponding secure PHF. This PHF is updated only when CSP needs to store new popular blocks. In the next sections, we first present the popularity check phase which is common to all three scenarios and then explain the following phases.

5.2 Popularity Check (Scenarios 1, 2 and 3)

Before uploading a file F, C splits F into blocks $F = \{b_i\}$, encrypts each of them with CE and computes their IDs. We point out that our scheme is completely independent of the underlying data-chunking strategy used for determining block boundaries, which is a problem that is out of the scope of this paper. The client fetches the PHF from the CSP and evaluates it over $\{h(CE(b_i))\}$. The result of this operation is a set of indices $I = \{PHF(h(CE(b_i)))\}$, where each index represents the position of the potentially popular block ID in the hash table stored at the CSP. These indices can be used to perform the popularity check without revealing the content of the blocks to the CSP. Indeed, given a set of indices obtained as above, the client can retrieve the corresponding block IDs stored in the hash table and then compare them with his own block IDs. Any block b_i such that $h(CE(b_i))$ is equal to the popular block ID retrieved from the CSP, is considered as popular, hence will be deduplicated. The index does not reveal any exploitable information on the block.

5.3 Popularity Transition (Scenarios 1 and 2)

If the popularity check reveals that a block is not popular, C needs to check whether it is going to trigger a popularity transition. A block becomes popular as soon as it has been uploaded by t users. In order to enable C to be aware of the change of the popularity status and perform the transition, C sends an update to the IS whenever the popularity check has returned a negative result for a given block ID. IS stores a list of block IDs and owners corresponding to each encrypted copy of the yet unpopular block. When the number of data owners for a particular block reaches t, the popularity transition protocol is triggered and IS returns to C the list of block IDs. In order to complete this transition phase, CSP stores the convergent-encrypted copy, removes the corresponding encrypted copies and updates the PHF. From now on, the block will be considered popular, therefore it will be deduplicated. We point out that this operation is totally transparent to the other users who uploaded the same block as unpopular. Indeed, during their upload phase, users also keep encrypted information about the convergent encryption key. This allows them decrypting the block when it becomes popular.

5.4 Data Upload (Scenarios 1, 2 and 3)

Once the client has determined the popularity of each block, he can send the actual upload request. The content of the request varies depending on the block status. If the block is unpopular, C uploads the block symmetrically encrypted with a random key. If the block is popular, C only uploads the block ID, so that the CSP can update his data structures. Optionally, in order to avoid to manage the storage of the encryption keys, C may rely on the CSP for the storage of the random encryption key and the convergent encryption key, both encrypted with a secret key known only by the client.

6 Security Analysis

In this section, we analyze the security of the proposed scheme, the CSP being considered the main adversary. The CSP is "honest-but-curious", meaning that it correctly performs all operations but it may try to discover the original content of unpopular data. We do not consider scenarios where the CSP behaves in a byzantine way. We assume that CSP cannot collude with the IS since this component is trusted. Since the goal of the malicious CSP is to discover the content of unpopular blocks, we analyze in detail whether (and how) confidentiality is guaranteed for unpopular data in all phases of the protocol. However, if the user wants to keep a file confidential even when it becomes popular, he may encrypt the file with a standard encryption solution and upload it to the cloud without following the protocol steps. Finally, we also analyze some attacks that may be perpetrated by users themselves and propose simple countermeasures against them.

Security of blocks stored at the CSP By definition, an unpopular block is encrypted using a semantically-secure symmetric encryption. The confidentiality of unpopular data segments thus is guaranteed thanks to the security of the underlying encryption mechanism.

Security during Popularity Check The information exchanged during the Popularity Check must not reveal any information that may leak the identity of an unpopular block owned by the user. The identity of an unpopular block is protected thanks to the one-wayness of the secure PHF: the query generated by the client does not include the actual unpopular block ID but an integer i that is calculated by evaluating the secure PHF on the block ID. Simple guessing by exploring the results of the secure hash function embedded in the PHF is not feasible thanks to the one-wayness of the underlying secure hash function (SHA-3 [24]). In addition to that, when the PHF is evaluated over an unpopular block ID, there is definitely a collision between the ID of the unpopular block and the ID of a popular block stored at the CSP. These collisions serve as the main countermeasure to the disclosure of the unpopular block ID sent to the CSP during the lookup. With a reasonable assumption, we can also consider that the output of the underlying secure hash function (SHA-3) is random. In case of a collision between an unpopular block ID and the ID of a popular block stored at the CSP, thanks to the randomness of the underlying secure hash function, the output of a PHF based on such a hash function is uniformly distributed between 0 and m. In the case of such a collision, the probability that the CSP guesses the unpopular block ID used as input to the PHF by the client thus is:

$$\frac{m}{\left|\bar{P}\right|} = \frac{|P|}{\left|\bar{P}\right| * \alpha} \tag{1}$$

where P is the set of popular block IDs stored at the CSP, $\bar{P}$ is the rest of the block ID domain including all possible unpopular block IDs, α is the load factor of the PHF such that $m = \frac{|P|}{\alpha}$.

Assuming that the cardinality of the entire domain is much larger than the cardinality of the set of popular block IDs (which is the case if popular block IDs are the result of a secure hash function), we can state that the number of collisions per index is large enough to prevent a malicious CSP from inferring the actual block ID used as input to the PHF. In a typical scenario using a PHF based on a secure hash function like SHA-3, whereby the complexity of a collision attack would be 2^{256}, and a popular block ID set with 10^9 elements, this probability will be ($\alpha = 0.81$):

$$\frac{10^9}{(2^{256} - 10^9) * 0.81} \approx 1.06 * 10^{-68} \tag{2}$$

Hence collisions can effectively hide the identity of unpopular blocks from an untrusted cloud provider while keeping the lookup protocol extremely efficient and lightweight for the users.

Security against potential protocol vulnerabilities We now consider a few additional attacks that may be perpetrated by the CSP. For each of them, we propose simple but effective countermeasures, which are easy to implement and do not significantly increase the computational and network overhead. First, we consider that the CSP may pre-build a PHF based on some specific data (derived for example from a dictionary) which have not been yet uploaded by users. Within such a scenario, clients would detect their requested block to be popular although it has never actually been uploaded by any user; such a block will then be stored with a lower level of protection. As a countermeasure to such an attack, we propose that the IS attaches a signature to each popular block ID upon the Popularity Transition. Therefore, the IS will sign popular block IDs before being stored at the CSP, enabling clients to verify the authenticity of these blocks when running the popularity check. Such a countermeasure would have a minimal impact on the performance of the system. Another attack we consider is related to the confirmation-of-file attack to which convergent encryption is also vulnerable [9]. Indeed, upon a Popularity Check, the CSP may compare the sequence of indices sent by the client with the sequence produced by a given popular file F. If the two sequences match, then there is a chance that the client is actually uploading F. In order to hide this information from the CSP, the client may add a number of random indices to the list of indices being sent upon the Popularity Check. Thanks to the resulting noise included in the index list, the identification of the target file by the CSP will be prevented. This countermeasure also prevents the CSP from running the learn-the-remaining-information attack. Moreover, the overhead due to this countermeasure is negligible both in terms of bandwidth and computation.

Security against users Users may force a popularity transition by repeatedly uploading random or targeted blocks. As a countermeasure, the popularity threshold may be set to a value $t' = t + u$, where u is the expectation of the maximum number of malicious users. As opposed to the proposal of [10], the threshold can be dynamically updated at any time of the system life. Indeed, this parameter is transparent to both users and the CSP, hence the Index Ser-

vice can update it depending on the security needs. Users may also perpetrate a DoS attack by deleting random blocks stored at the cloud. This may happen upon a popularity transition: the client is asked to attach a list of block IDs that may not be the actual encrypted copies of the block being uploaded. We suggest making the Index Service sign the list of block IDs to be deleted so that the cloud can verify whether the request is authentic. This signature does not significantly increase the overhead since several schemes for short signatures [22] have been proposed in the literature.

7 Performance Evaluation

7.1 Prototype Implementation

In order to prove the feasibility of our approach, we implemented a proof-of-concept prototype consisting of the three main components, namely, the Client, the IS and the CSP. All components have been implemented in Python. Cryptographic functions have been implemented using the pycrypto library[4]. Both the Client and the IS run on an Ubuntu VM hosted on our OpenStack platform, while the CSP runs on an Ubuntu VM hosted on Amazon EC2 (EU Region). The IS uses REDIS[5] in order to store the information on unpopular blocks, which are encoded as lists. Metadata (block IDs, file IDs, files structures, encrypted keys) are stored in a MySQL database. Perfect Hashing has been implemented using the CMPH library[6] at both the Client and the CSP. In order to achieve one-wayness, we customized CMPH by replacing the internal hash function with SHA256 [20]. We stress the fact that this is a proof-of-concept implementation, therefore for the sake of simplicity the CSP has been deployed on a VM where data blocks are stored locally. In a production environment, the CSP service may be deployed on a larger scale and any storage provider such as Amazon S3[7] may be employed to physically store blocks.

We consider a scenario where the client uploads a 10MB file to the CSP prefilled with 10^6 random blocks. We propose to first evaluate the computational overhead of each single component and measure the total time a client needs to wait during each phase until the data upload has been completed. We then analyze the network overhead of the proposed solution. Our analysis considers the three previously described scenarios:

- Scenario 1 (Unpopular File): the file to be uploaded is still unpopular;
- Scenario 2 (Popularity Transition): the file has triggered a popularity transition hence is going to become popular;
- Scenario 3 (Popular File): the file to be uploaded is already popular.

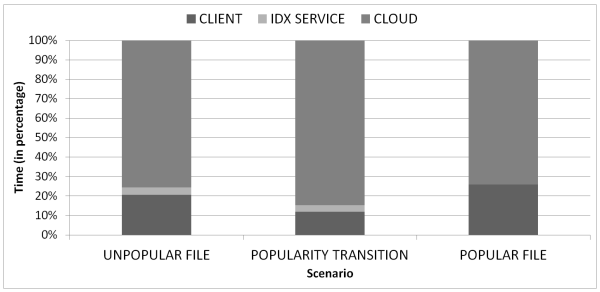

Fig. 2. Portion of the total computation time spent at each component in each scenario

7.2 Computational Overhead

In this section we present our measurements of the computational overhead at each component and then show the total time a client takes to upload a file. Figure 2 shows an aggregate measure of all computation-intensive operations each component performs. The results prove that, as expected, the computational overhead introduced in the CSP is much higher than the one affecting the client. Also, since the operations performed by the IS are extremely simple, its computational overhead is negligible.

Figure 3 shows more detailed results by highlighting which operations introduce a higher computational overhead. The results prove that:

- Symmetric encryption introduces a negligible computational overhead, hence it does not affect the system performance;
- The client-side Popularity Check is extremely lightweight and thus introduces a negligible computational overhead;
- The most computation-intensive operations (PHF generation, hash table storage, upload processing) are performed by the CSP, hence a big fraction of the computational overhead is outsourced to the CSP.

Figures 4 and 5 show the results of an in-depth study on the performance of the Perfect Hashing algorithm, both in terms of storage space and computation time for the generation of the PHF. The generation time also includes the time needed to store the hash table. We measured these quantities on a dataset of 10^6 random block IDs while varying the load factor and the bucket size. The former is a coefficient indicating the fraction of non-empty positions in the final collision-free hash table; the latter is the mean number of elements in each subset of the input set (see [11] for further details). As we can observe from Figures 4 and 5, the optimal bucket size is between 3 and 4 and the load factor should not

[4] https://pypi.python.org/pypi/pycrypto
[5] http://redis.io
[6] http://cmph.sourceforge.net/
[7] https://aws.amazon.com/s3

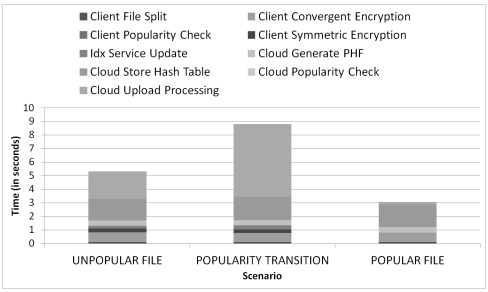

Fig. 3. Total time spent during each phase of the protocol in each scenario

be greater than 0.8. These parameters can be tuned depending on the scenario (e.g. bandwidth) in order to achieve the best performance.

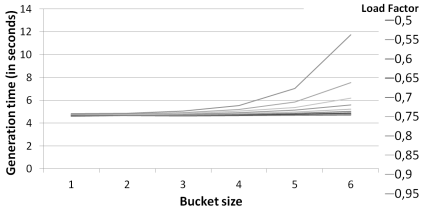

Fig. 4. Analysis of PHF generation time with varying parameters for a set containing 10^6 elements

Furthermore, as mentioned earlier, in order to improve the security of our novel lookup protocol, we replaced the default hash function employed by the CMPH library (Jenkins [21]) with SHA-3. This improvement is required for the following reason: using a non-secure hash function would allow an adversary such as the CSP to easily enumerate all block IDs mapped to a given index of the hash table. Such a threat may compromise the security of the whole system and make the popularity check protocol insecure.

Conclusion Figure 6 summarizes all measurements by showing the total time spent during each phase of the upload protocol within the three scenarios. These results show that despite the delay introduced by the Popularity Check phase, the user achieves a throughput of approximately 1MB per second even when a file does not contain any popular block.

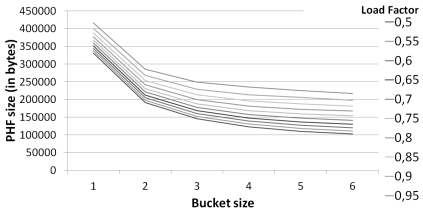

Fig. 5. Analysis of PHF size with varying parameters for a set containing 10^6 elements

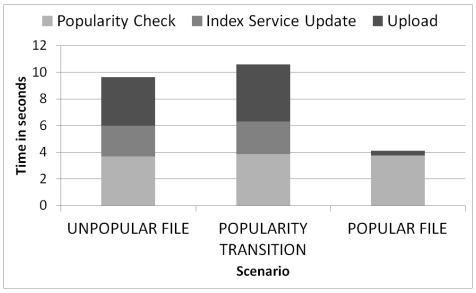

Fig. 6. Total time spent by all components when uploading a file (including Popularity Check) in each scenario

7.3 Communication Overhead

In this section we analyze the communication overhead of our scheme considering the same scenarios. The upload has been split into multiple sub-operations: PHF Download, Popularity Check, Index Service Update (not performed in Scenario 2) and the Upload. For each of these operations we analyze the size of all messages exchanged (both requests and responses). Table 1 regroups all the results expressed in MB. The PHF Download response size is linear with respect to the set of popular block IDs. The larger the set, the larger the response will be. However, as shown in [11], the size of PHF file is about 1.4 bits per popular block ID; hence this operation does not introduce a significant delay even when dealing with very large datasets. We point out that the PHF file does not have to be downloaded at every request, since the user can cache it. Furthermore, the size of the Popularity Check request and response is linear with respect to the number of blocks in the file that is being uploaded. The Popularity Check request contains a list of indices (one integer per block), while the response contains a list of block IDs (one per index) of 32 bytes each. The Index Service Update request is only sent for unpopular blocks. The request consists of two block IDs (32 bytes each) per block. The response size varies depending on whether the popularity transition occurs. If the file has triggered a popularity transition, then the response includes a list of block IDs, otherwise it is empty. As we can see from Table 1, requests and responses of the Popularity Check and the Index

Service Update operations have a negligible size with respect to the file size. Finally, the size of the Upload request varies depending on the block status. If a block is popular, the request only consists of the block ID and one key (32 bytes). If a block is not popular, the request contains the encrypted data, two keys (32 bytes each) and a few fields: the file ID (32 bytes), the user ID and the block status (1 byte). As shown in Table 1, the overhead introduced by the Upload is minimal and mainly depends on the encoding method used to transfer the encrypted binary data. For simplicity, we used JSON objects to pack encrypted blocks and keys and Base64 to encode binary data, which increases the size of the data by 1/3. To summarize, the preliminary operations performed in our scheme before the Upload introduce a negligible communication overhead. In addition, the scheme does not affect the gains in terms of storage space and bandwidth achieved thanks to deduplication.

	SCENARIO 1	**SCENARIO 2**	**SCENARIO 3**
PHF DOWNLOAD IN	0.67	0.67	0.67
POPULARITY CHECK REQUEST	0.004	0.004	0.004
POPULARITY CHECK RESPONSE	0.02	0.02	0.02
INDEX SERVICE UPDATE REQUEST	0.1	0.1	-
INDEX SERVICE UPDATE RESPONSE	0.009	0.04	-
UPLOAD REQUEST	13.51	13.47	0.09

Table 1. Communication overhead (in MB) introduced by each operation

8 Related Work

Secure deduplication for cloud storage has been widely investigated both in the literature and in the industry. Convergent encryption, has been proposed as a simple but effective solution to achieve both confidentiality and deduplication [1, 3, 4]. However, it is vulnerable to well-known attacks which put data confidentiality at risk [3, 4]. A relevant work on this topic is DupLESS [8], which is based on a privacy-preserving protocol running between the user and a trusted key server. If an attacker learns the secret stored at the key server, confidentiality can no longer be guaranteed. Recently, a system called ClouDedup [9] has been proposed, which achieves secure and efficient block-level deduplication while providing transparency for end users. However, the system relies on a complex architecture in which users have to trust an encryption gateway which takes care of encrypting/decrypting data. Similarly to DupLESS, the leakage of the secret key compromises confidentiality.Another relevant work is iMLE [2], which proposes an elegant scheme for secure data deduplication. However, the scheme is purely theoretical, hence cannot be adopted in real scenarios. In fact, it makes an extensive use of fully homomorphic encryption [23]. To the best of our knowledge, one of the most recent and relevant works in the field of secure data

deduplication is [10], which is based on the idea of differentiating data protection depending on its popularity and makes use of a mixed cryptosystem combining convergent encryption and a threshold encryption scheme. However, this work suffers from a few drawbacks which we aim to solve. First, the system suffers from a significant storage and bandwidth overhead. Indeed, for each unpopular file the user uploads two encrypted copies, one encrypted with a random symmetric key and one encrypted with the mixed encryption scheme. In scenarios with a high percentage of unpopular files, the storage overhead will be significant and nullify the savings achieved thanks to deduplication. We propose to eliminate the storage overhead by storing one single copy for each data segment at a time, encrypted with either a random symmetric key or a convergent key. Second, the system proposed in [10] relies on a trusted component which provides an indexing service for all data, both popular and unpopular. We propose to limit the usage of this trusted component to unpopular data. In our scheme, popular data can be detected thanks to the secure lookup protocol, whereby [10] relies on the trusted component. Third, the effectiveness of the system proposed in [10] is limited to file-level deduplication, which is known to achieve lower space savings than block-level deduplication. Fourth, both the client and the CSP have to perform complex cryptographic operations based on threshold cryptography on potentially very large data. As opposed to this, our proposed scheme has been designed to perform only simple and lightweight cryptographic operations, which significantly lowers the cost for the client. Fifth, our scheme does not require any coordination or initialization among users as opposed to [10]'s requirement to setup and distribute key shares among users.

9 Conclusion and Future Work

We designed a system which guarantees full confidentiality for confidential files while enabling source-based block-level deduplication for popular files. The main building block of our system is our novel secure lookup protocol built on top of an enhanced version of Perfect Hashing. To the best of our knowledge, this is the first work that uses Perfect Hashing for a different purpose other than database indexing. Our system is not based on any key-management protocol, hence it does not require users to agree on a shared secret or trust a third party for storing encryption keys. A semi-trusted component is employed for the purpose of storing metadata concerning unpopular data and providing a support for detecting popularity transitions, meaning that a data block has just reached the popularity threshold. We also implemented a prototype of the proposed solution. Our measurements show that the storage, network and computational overhead is affordable and does not affect the advantage of deduplication. Also, we showed that the computational overhead is moved to the CSP, while the client has to perform very lightweight operations. As part of future work, PerfectDedup may be optimized in order to reduce the overhead due to the PHF generation and transmission.

References

1. Xu, Jia, Ee-Chien Chang, and Jianying Zhou. "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage." In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pp. 195-206. ACM, 2013.
2. Bellare, Mihir, and Sriram Keelveedhi. "Interactive message-locked encryption and secure deduplication." (2015).
3. Adya, Atul, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment." ACM SIGOPS Operating Systems Review 36, no. SI (2002): 1-14.
4. Douceur, John R., Atul Adya, William J. Bolosky, P. Simon, and Marvin Theimer. "Reclaiming space from duplicate files in a serverless distributed file system." In Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, pp. 617-624. IEEE, 2002.
5. Perttula. Attacks on convergent encryption. `http://bit.ly/yQxyvl`.
6. Liu, Chuanyi, Xiaojian Liu, and Lei Wan. "Policy-based de-duplication in secure cloud storage." In Trustworthy Computing and Services, pp. 250-262. Springer Berlin Heidelberg, 2013.
7. Meyer, Dutch T., and William J. Bolosky. "A study of practical deduplication." ACM Transactions on Storage (TOS) 7, no. 4 (2012): 14.
8. Bellare, Mihir, Sriram Keelveedhi, and Thomas Ristenpart. "DupLESS: server-aided encryption for deduplicated storage." In Proceedings of the 22nd USENIX conference on Security, pp. 179-194. USENIX Association, 2013.
9. Puzio, Pasquale, Refik Molva, Melek Önen, and Sergio Loureiro. "ClouDedup: secure deduplication with encrypted data for cloud storage." In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on, vol. 1, pp. 363-370. IEEE, 2013.
10. Stanek, Jan, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. "A secure data deduplication scheme for cloud storage." In Financial Cryptography and Data Security, pp. 99-118. Springer Berlin Heidelberg, 2014.
11. Belazzougui, Djamal, Fabiano C. Botelho, and Martin Dietzfelbinger. "Hash, displace, and compress." In Algorithms-ESA 2009, pp. 682-693. Springer Berlin Heidelberg, 2009.
12. Cox, Landon P., Christopher D. Murray, and Brian D. Noble. "Pastiche: Making backup cheap and easy." ACM SIGOPS Operating Systems Review 36, no. SI (2002): 285-298.
13. Rabin, Michael O. Fingerprinting by random polynomials. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.
14. Wilcox-O'Hearn, Zooko, and Brian Warner. "Tahoe: the least-authority filesystem." In Proceedings of the 4th ACM international workshop on Storage security and survivability, pp. 21-26. ACM, 2008.
15. Harnik, Danny, Benny Pinkas, and Alexandra Shulman-Peleg. "Side channels in cloud services, the case of deduplication in cloud storage." IEEE Security & Privacy 8, no. 6 (2010): 40-47.
16. Is Convergent Encryption really secure? `http://bit.ly/Uf63yH`
17. Bellare, Mihir, Sriram Keelveedhi, and Thomas Ristenpart. "Message-locked encryption and secure deduplication." In Advances in CryptologyEUROCRYPT 2013, pp. 296-312. Springer Berlin Heidelberg, 2013.

18. Storer, Mark W., Kevin Greenan, Darrell DE Long, and Ethan L. Miller. "Secure data deduplication." In Proceedings of the 4th ACM international workshop on Storage security and survivability, pp. 1-10. ACM, 2008.
19. Olumofin, Femi, and Ian Goldberg. "Privacy-preserving queries over relational databases." In Privacy enhancing technologies, pp. 75-92. Springer Berlin Heidelberg, 2010.
20. Description of SHA256 http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf
21. Jenkins hash function http://www.burtleburtle.net/bob/c/lookup3.c
22. Boneh, Dan, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing." In Advances in CryptologyASIACRYPT 2001, pp. 514-532. Springer Berlin Heidelberg, 2001.
23. Gentry, Craig. "A fully homomorphic encryption scheme." PhD diss., Stanford University, 2009.
24. SHA-3. http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf
25. Chor, Benny, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private information retrieval." Journal of the ACM (JACM) 45, no. 6 (1998): 965-981.
26. Freedman, Michael J., Kobbi Nissim, and Benny Pinkas. "Efficient private matching and set intersection." Advances in Cryptology-EUROCRYPT 2004. Springer Berlin Heidelberg, 2004.

Multi-User Searchable Encryption in the Cloud

Cédric Van Rompay, Refik Molva, and Melek Önen

EURECOM, Sophia Antipolis, France,
{vanrompa,molva,onen}@eurecom.fr

Abstract. While Searchable Encryption (SE) has been widely studied, adapting it to the multi-user setting whereby many users can upload secret files or documents and delegate search operations to multiple other users still remains an interesting problem. In this paper we show that the adversarial models used in existing multi-user searchable encryption solutions are not realistic as they implicitly require that the cloud service provider cannot collude with some users. We then propose a stronger adversarial model, and propose a construction which is both practical and provably secure in this new model. The new solution combines the use of bilinear pairings with private information retrieval and introduces a new, non trusted entity called "proxy" to transform each user's search query into one instance per targeted file or document.

1 Introduction

Cloud computing nowadays appears to be the most prominent approach for outsourcing storage and computation. Despite well known advantages in terms of cost reduction and efficiency, cloud computing also raises various security and privacy issues. Apart from classical exposures due to third party intruders one of the new requirements akin to outsourcing is the privacy of outsourced data in the face of a potentially malicious or careless Cloud Service Provider (CSP).

While data encryption seems to be the right countermeasure to prevent privacy violations, classical encryption mechanisms fall short of meeting the privacy requirements in the cloud setting. Typical cloud storage systems also provide basic operations on stored data such as statistical data analysis, logging and searching and these operations would not be feasible if the data were encrypted using classical encryption algorithms.

Among various solutions aiming at designing operations that would be compatible with data encryption, Searchable Encryption (SE) schemes allow a potentially curious party to perform searches on encrypted data without having to decrypt it. SE seems a suitable approach to solve the data privacy problem in the cloud setting.

A further challenge is raised by SE in the multi-user setting, whereby each user may have access to a set of encrypted data segments stored by a number of different users. Multi-user searchable encryption schemes allow a user to search through several data segments based on some search rights granted by the owners of those segments. Privacy requirements in this setting are manifold, not only the

confidentiality of the data segments but also the privacy of the queries should be assured against intruders and potentially malicious CSP. Recently, few research efforts [5, 8, 12, 15] came up with multi-user keyword search schemes meeting these privacy requirements, either through some key sharing among users or based on a Trusted Third Party (TTP).

In this paper, we first investigate the new privacy challenges for keyword search raised by the multi-user setting beyond the basic privacy concerns about data, queries and responses by focusing on the relationship among multiple queries and responses. We realize that while as analyzed in [7], the protection of the *access pattern privacy* (privacy of the responses) is optional for single-user searchable encryption mechanisms, this requirement becomes mandatory in the multi-user setting. Unfortunately all existing Multi-User Searchable Encryption (MUSE) schemes [5,8,12,15] suffer from the lack of such protection. We further come up with a new adversary model for MUSE that takes into account new security exposures introduced by the possible collusion of some users with the CSP.

After showing that all existing MUSE schemes fail at meeting the privacy requirements in our new adversarial model, we suggest a new solution for MUSE for which it is not the case, i.e., all users who have not been explicitly authorized to search a document can collude with the adversary without threatening the privacy of that document. Our solution for MUSE inherently ensures access pattern privacy through the use of Private Information Retrieval (PIR). While the PIR protocol together with the multi-user setting may add a significant complexity overhead, this overhead is outsourced from the users to a third party our scheme introduces, the *proxy*, that is in charge of multiplexing a user query into several PIR queries. Moreover the overhead of PIR is further lowered by querying binary matrices representing the keyword indices instead of querying the bulky keyword lists themselves. As opposed to most existing solutions based on a TTP [3,5,8,15], the proxy in our scheme does not need to be trusted. With the sole assumptions that the CSP and the proxy are honest-but-curious and that they do not collude with one another, we prove that our solution meets the privacy requirements defined for MUSE.

Section 2 states the problem addressed by MUSE. Section 3 describes our solution for MUSE and Section 4 defines the security properties for MUSE. Section 5 proves that our solution achieves the security properties we defined and Section 6 studies the algorithmic complexity of our solution. Section 7 reviews the state of the art and, finally, Section 8 concludes the paper.

2 Multi-User Searchable Encryption (MUSE)

A MUSE mechanism extends existing keyword search solutions into a multi-writer multi-reader [6] architecture involving a large number of users, each of which having two roles:

- as a *writer*, the user uploads documents to the server and delegates keyword search rights to other users.

– as a *reader*, the user performs keyword search operations on the documents for which she received delegated rights.

As any SE solution, MUSE raises two privacy requirements:

– *index privacy*: unauthorized parties should not discover information about the content of uploaded documents.
– *query privacy*: no one should get information about the targeted word and the result of a search operation apart from the reader who sent the corresponding query.

In addition to the CSP, any user that has not been explicitly given search rights on an index should be considered as potentially colluding with the CSP in order to violate index or query privacy. This assumption leads to a model in which the adversary is composed of a coalition of the CSP and some non-delegated users. This new adversary model extends the one used in other existing MUSE schemes [5,7,12,15], which although secure in their own adversary model do not achieve index and query privacy any more if non-delegated users collude with the CSP.

Figure 1 illustrates one example of the impact of a collusion between a CSP and a user on privacy by taking advantage of the lack of access pattern privacy. Assuming that R_1 is authorized to query both indices I_1 and I_2, by observing the access pattern of R_1's queries, the CSP can discover similarities between I_a and I_b. In a second phase, the CSP corrupts reader R_2 who is authorized to query I_b only. By exploiting the similarities between I_a and I_b and discovering the content of I_b through R_2, the CSP can easily discover the content of I_a. The index privacy is thus violated for I_a since the CSP partially learns the content of I_a although R_1, the only reader having delegated search rights for I_a, was not corrupted. Furthermore, once the CSP obtains information about the content of an index, observing the access pattern of the queries targeting this index enables the CSP to violate the privacy of these queries. This attack allows to violate both query and index privacy in all existing MUSE schemes since they all let the CSP discover the access pattern of the queries. The new adversary model we introduce not only prevents such an attack but also prevents any attack that would require the corruption of a non-delegated user.

3 Our Solution

3.1 Idea

Our solution introduces a third party called the *proxy* that performs an algorithm called *QueryTransform* to transform a single reader query into one query per targeted document [1]. For each of these queries, the proxy sends to the CSP a specific

[1] Note that the set of targeted document can reveal the authorized set of documents for this particular user. However, such an additional information does not have a serious impact on index or query privacy as access pattern leakage has.

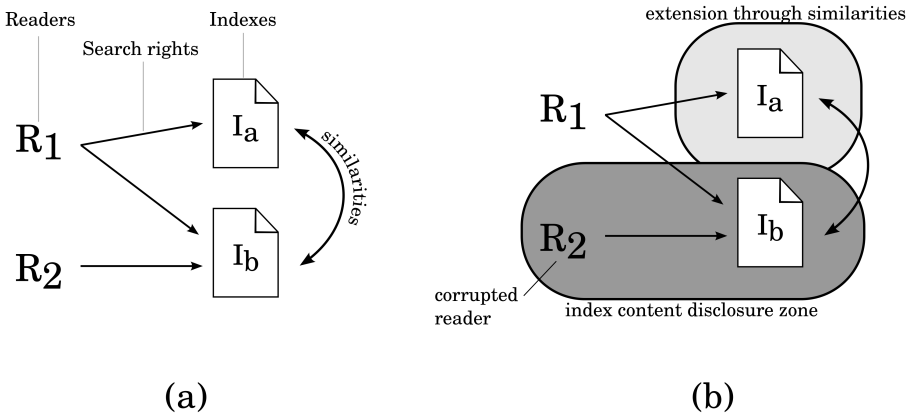

Fig. 1. In (a), discovery of similarities through the access pattern. In (b), use of the similarities to extend index privacy violation.

PIR request. Thanks to the PIR protocol the CSP does not have access neither to the content of the query nor to its result, which makes our scheme achieving query privacy (including access pattern privacy) against the CSP. While the use of PIR provides privacy against the CSP, a new privacy exposure raises with respect to the proxy. Indeed through the execution of *QueryTransform*, the proxy is able to discover the relationship between a query and the different ciphertexts in the targeted indices which are the encryption of the same keyword. However with the assumption that the proxy does not collude with the CSP, the proxy cannot realize whether these ciphertexts are present in their respective indices or not; thus, our scheme achieves index privacy against the proxy. Moreover thanks to some randomization of the queries and the encryption of the responses by the CSP with the reader's key, the proposed solution also ensures query privacy against the proxy. Consequently, while our solution does introduce a third party (the proxy), **this third party does not need to be trusted** and is considered as an adversary. Both the CSP and the proxy are then considered as potentially malicious in our scheme, and are only assumed *honest-but-curious* and non colluding with each other.

Another advantage of introducing the proxy into this new MUSE solution is scalability: Indeed, thanks to the $QueryTransform$ algorithm executed by the proxy a user does not need to generate several PIR queries (one per index) for the same keyword.

3.2 Preliminaries

Bilinear Pairings Let G_1, G_2 and G_T be three groups of prime order q and g_1, g_2 generators of G_1 and G_2 respectively. $e : G_1 \times G_2 \rightarrow G_T$ is a bilinear map if e is:

– efficiently computable
– non-degenerate: if x_1 generates G_1 and x_2 generates G_2, then $e(x_1, x_2)$ generates G_T
– bilinear: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}\ \forall (a, b) \in \mathbb{Z}^2$

We assume that the widely used eXternal Diffie-Hellman (XDH) assumption [4] holds.

Definition 1 (External Diffie Hellman assumption). *Given three groups G_1, G_2 and G_T and a bilinear map $e : G_1 \times G_2 \to G_T$, the Decisional Diffie-Hellman (DDH) problem is hard in G_1, i.e., given $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta) \in G_1^4$, it is computationally hard to tell if $\delta = \alpha\beta$.*

Private Information Retrieval (PIR) A PIR protocol allows a user to retrieve data from a database without revealing any information about the retrieved data.

PIR consists of five algorithms:

– **PIR.Setup**() $\to PIRParams$
– **PIR.KeyGen**() $\to (PirKey)$: this algorithm outputs the keying material for PIR.
– **PIR.Query**$(PirKey, size, target) \to Query$: given PIR parameters, the size of the targeted database and a target position, this algorithm outputs a PIR query targeting the given position in a database of the given size.
– **PIR.Process**$(Query, DataBase) \to R$: this algorithm applies the query $Query$ on the database $DataBase$ and outputs a response R.
– **PIR.Retrieve**$(R, PirKey) \to Cell$: given a PIR response R and the PIR key used in corresponding query, this algorithm outputs the value of the retrieved database cell.

Single-database computational PIR has already been widely studied [1, 2, 10, 11], and the results presented in [1] show that solutions with practical performances already exist. Our solution uses the technique of recursive PIR which allows to reduce communication complexity as explained in [1]: The database is viewed as a matrix each row of which is considered as a sub-database. To query the whole database a single query is sent and this query is further applied on each row, resulting in the generation of many PIR responses.

3.3 Protocol Description

Figure 2 illustrates the structure and the various flows of our solution. We define two phases in the protocol, the *upload phase* and the *search phase*: During the upload phase, a writer A uploads a secure index to the CSP by encrypting each keyword with the $Index$ algorithm. A then delegates search rights to reader B using the $Delegate$ algorithm which computes an authorization token using B's public key and A's private key. The authorization token is sent to the proxy. During the search phase, B can further search all the indices for which she has

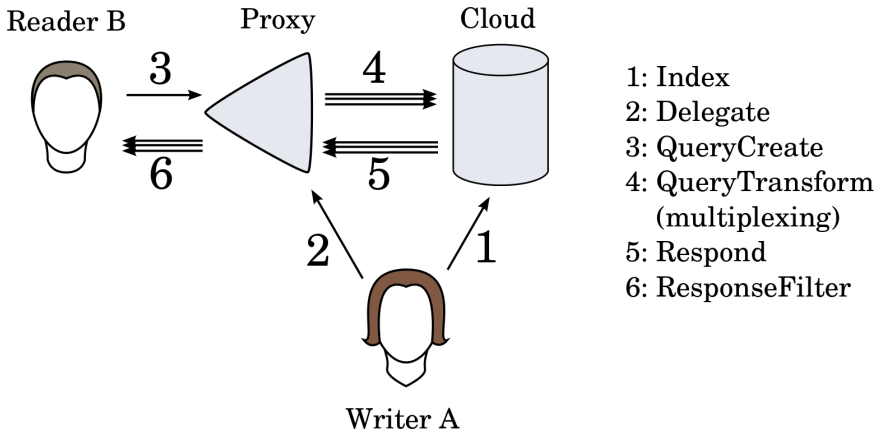

Fig. 2. Overview of our solution.

been given search rights, by creating a single query through the execution of $QueryCreate$. Whenever the proxy receives the B's query it uses the authorization tokens attributed to B to transform this query into one PIR query per authorized index through the execution of $QueryTransform$. Upon reception of a PIR query, the CSP through the execution of $Respond$ builds a binary matrix using the corresponding encrypted index, applies the query to the matrix and encrypts the resulting PIR answers. The responses are then pre-processed by the proxy through the execution of $ResponseFilter$. Finally B obtains the result of her search query by executing $ResponseProcess$.

Revocation in our solution only consists in the deletion of the appropriate authorizations by the proxy upon a writer's request.

The set of users is denoted by $u_{i1\leq i\leq N}$. For the sake of clarity, each user u_i is assumed to own only one index I_i.

- **Setup**$(\kappa) \rightarrow params$: given the security parameter κ, this algorithm outputs the parameters $param$ consisting in:
  - a description of the bilinear map that will be used: the three groups G_1, G_2, G_T of prime order q, the two generators g_1 and g_2 and the map itself e.
  - a cryptographically secure hash function $h: \{0,1\}^* \rightarrow G_1$
  - the size n of the matrices for PIR, and a hash function $H: G_T \rightarrow [\![0, n-1]\!]$ to transform encrypted keywords into positions in the matrices. Without loss of generality, n is assumed to be a perfect square.
  - the PIR parameters $PIRParams$ from the execution of $PIR.Setup$
  - a symmetric cipher Enc and the corresponding decipher algorithm Dec.

  All these parameters are considered implicit for each further algorithm.
- **KeyGen**$(\kappa) \rightarrow (\gamma, \rho, P, K)$: given the security parameter κ, a user u_i generates the following keys:

  - a *secret writer key* $\gamma_i \xleftarrow{\$} \mathbb{Z}_q^*$
  - a *private reader key* $\rho_i \xleftarrow{\$} \mathbb{Z}_q^*$
  - a *public reader key* $P_i = g_2^{\frac{1}{\rho}}$
  - a *transmission key* K_i used for Enc/Dec. This key is shared with the CSP.
- **Index**$(w, \gamma_i) \rightarrow \tilde{w}$: Writer u_i executes this algorithm to encrypt keyword w with his key γ_i. The algorithm outputs $\tilde{w} = e(h(w)^{\gamma_i}, g_2)$.
- **Delegate**$(\gamma_i, P_j) \rightarrow \Delta_{i,j}$: Provided with the public key P_j of reader u_j, writer u_i executes this algorithm using its secret key γ_i to generate $\Delta_{i,j} = P_j^{\gamma_i}$ the authorization token that authorizes u_j to search the index I_i. The output $\Delta_{i,j}$ is sent to the proxy which adds it to the set D_j. Note that this token can only be created by the legitimate data owner and cannot be forged by any other party including the CSP and the proxy.
- **QueryCreate**$(w, \rho_j) \rightarrow Q_j$: This algorithm is run by an authorized reader to generate a query for keyword w using its private reader key ρ_j. The algorithm draws a randomization factor $\xi \xleftarrow{\$} \mathbb{Z}_q^*$ and outputs $Q_j = h(w)^{\xi\rho_j}$.
- **QueryTransform**$(Q_j, D_j) \rightarrow < Q'_{i,j} >$: Whenever the proxy receives a reader's query Q, it calls this algorithm together with the set D_j.
  For each authorization token $\Delta_{i,j}$ in D, the algorithm creates a PIR query Q'_j as follow:
  - compute $\tilde{Q}_{i,j} \leftarrow e(Q_j, \Delta_{i,j})$
  - compute $x'||y' \leftarrow H(\tilde{Q}_{i,j})$
  - some PIR keying material is generated: $PirKey \leftarrow PIR.KeyGen()$
  - a $\sqrt{n}$-size PIR query is created that targets position y': $Q'_{i,j} \leftarrow PIR.Query(PirKey, \sqrt{n}, y')$

  The algorithm outputs $< Q'_{i,j} >$ which are forwarded to the CSP together with the corresponding identifiers i of the indices. The proxy additionally stores each generated $PIRKey$ and x' in a table in order to use them upon reception of the corresponding response.
- **Respond**$(Q', I, \xi) \rightarrow R$: Whenever the CSP receives an individual PIR query Q', it executes this algorithm using the corresponding index I and the randomization factor ξ corresponding to this query.
  The algorithm initializes a $\sqrt{n} \times \sqrt{n}$ matrix M with "0". Then for each encrypted word $\tilde{w} \in I$, the cell $M_{x,y}$ is set to "1" where $x||y \leftarrow H(\tilde{w}^{\xi})$ (recall that $\tilde{w} \in G_T$). The response is the tuple of the outputs from the application of the PIR query Q' on each row of the binary matrix M: $\tilde{R} \leftarrow (PIR.Process(Q', M_x) \mid M_x \text{ a row of } M)$. Each component of $\tilde{R}$ is then encrypted with algorithm Enc using the transmission key K of the querying reader to obtain R which the algorithm outputs. This layer of encryption prevents the proxy from reading the result of the query.
- **ResponseFilter**$(R, x', PirKey) \rightarrow (R', PirKey)$: Whenever the proxy receives a response R it calls this algorithm together with the x' and $PirKey$ associated to the corresponding query. The purpose of this algorithm is to reduce the communication cost for the reader. Indeed the algorithm extracts

the x'-th component of R and outputs it together with the value for $PirKey$. This results in a filtered response which is much smaller than the original response.
- **ResponseProcess**$(R', PirKey, K) \rightarrow b \in \{0,1\}$: On receiving the filtered response R' with the corresponding $PirKey$, the reader executes this algorithm using her transmission key K. The algorithm further outputs the value of $PIR.Retrieve(Dec_K(R'), PirKey)$ which corresponds to the content of the retrieved matrix cell. An output of 1 means that the searched keyword is present in the index, and a 0 means that it is absent.

3.4 Correctness

We now show that a query correctly retrieves a particular cell which content corresponds to whether the queried keyword has been uploaded or not.

Let γ be the encryption key of a given index. If keyword w has been uploaded to that index, then the cell $M_{x,y}$ of the corresponding matrix is equal to 1 with $x||y = H(e(h(w), g_2^{\gamma}))$. Conversely if a given cell $M_{x,y}$ is equal to 1 then with high probability the corresponding keyword w where $x||y = H(e(h(w), g_2^{\gamma}))$ has been uploaded. A false positive implies a collision in either H or h. Thus the content of $M_{x,y}$ corresponds to the upload of w.

Secondly, a query for keyword w in that index will retrieve cell $M_{x',y'}$ with:

$$x'||y' = H(e(h(w)^{\rho}, g_2^{\frac{\gamma}{\rho}})) = H(e(h(w), g_2^{\gamma})) = x||y \ . \tag{1}$$

Thus a response to a query will decrypt to the content of the proper cell and our scheme is correct.

4 Security Model

Our security definitions are game-based definitions, where the games are represented by algorithms. Since the CSP and the proxy are considered as two non-colluding adversaries, security will be defined for each of them independently. The consequence of the non-collusion assumption is that each adversary will see the other one as an oracle. For each adversary type we define one game for index privacy and one game for query privacy. For each definition, the corresponding game consists of seven phases: a setup phase, a learning phase, a challenge phase, a restriction phase, second learning and restriction phases identical to the previous ones, and finally a response phase. The adversary is denoted by $\mathcal{A}$.

4.1 Security with the CSP as Adversary

We now formally define *Index Privacy* and *Query Privacy* considering the CSP as the adversary. In the following two definitions the setup and the learning phases are the same and are described in Alg. 1. The challenge and restriction phases for index privacy are further described in Alg. 2 and the ones for query privacy are described in Alg. 3. Finally during the *response phase*, $\mathcal{A}$ outputs a bit b^* representing its guess for the challenge bit b.

```
/* Setup phase                                                    */
A ← Setup();
for i = 1 to N do
    (γ_i, ρ_i, P_i, K_i) ← KeyGen(κ) ;
    A ← (i, P_i, K_i) ;
end
/* First learning phase                                           */
for j = 1 to a polynomial number l_1 do
    A → query ;
    switch query do
        case Index for word w and user u_i
            A ← Index(w, u_i);
        case Corrupt user u_i
            A ← (ρ_i, γ_i, K_i)
        case Delegation of user u_i by user u_j
            /* A does not receive any value, but the delegation will
               modify the set D_i used in QueryTransform            */
        end
        case Queries for word w from user u_i
            /* D_i comes from the Delegations queried by A          */
            A ← QueryTransform(QueryCreate(w, ρ_i), D_i);
            /* A also receives the randomization factor ξ           */
            A ← ξ;
        case Queries for user query Q from corrupted user u_i
            A ← QueryTransform(Q, D_i);
        case Filtered response for response R from corrupted user u_i
            A ← ResponseFilter(R)
    endsw
end
```

Algorithm 1: Setup and learning phases of both index privacy and query privacy games, whereby $\mathcal{A}$ is the CSP

```
/* Challenge phase                                                */
A → (u_chall, w_0^*, w_1^*);
b ←$ {0, 1};
A ← Index(w_b^*, u_chall);
/* Restriction phase                                              */
if u_chall is corrupted OR Index for w_0^* or w_1^* for user u_chall has been previously
queried OR a corrupted user has been delegated by u_chall then
    HALT;
end
```

Algorithm 2: Challenge and restriction phases of the index privacy game whereby $\mathcal{A}$ is the CSP

```
/* Challenge phase                                                    */
A → (u_chall, w_0^*, w_1^*);
b ←$ {0,1};
A ← QueryTransform(QueryCreate(w_b^*, ρ_chall), D_chall);
/* Restriction phase                                                  */
if u_chall is corrupted then
  | HALT;
end
```

Algorithm 3: Challenge and restriction phases of the query privacy game whereby $\mathcal{A}$ is the CSP

Definition 2 (Index Privacy Against the CSP). *We say that a MUSE scheme achieves index privacy against the CSP when the following holds for the index privacy game (Alg. 1 and Alg. 2):* $|Pr[b = b^*] - \frac{1}{2}| \leq \epsilon$*, with* ϵ *a negligible function in the security parameter* κ*.*

Definition 3 (Query Privacy Against the CSP). *We say that a MUSE scheme achieves query privacy against the CSP when the following holds for the query privacy game (Alg. 1 and Alg. 3):* $|Pr[b = b^*] - \frac{1}{2}| \leq \epsilon$*, with* ϵ *a negligible function in the security parameter* κ*.*

4.2 Security with the Proxy as Adversary

Due to space limitations we do not provide the detailed description of index and query privacy games whereby the proxy is considered as the adversary. In a nutshell, the main differences with the previous games are the following:

- during the learning phase the proxy can query for the *Respond* algorithm executed by the CSP, but does not query for the $QueryTransform$ and $ResponseFilter$ algorithms. Moreover the proxy receives the output of the *Delegate* algorithm, but does not get the transmission key and the randomization factors of the users.
- during the challenge phase, the proxy does not receive the output of the $Index$ algorithm for index privacy, and receives the output of $QueryCreate$ for query privacy.

5 Security Analysis

Inspired by the methodology in [14], in order to prove each security property we define a sequence of games $(game_i)_{i=0..n}$, the first game being the original security definition. For each game $game_i$ a "success event" S_i is defined as the event when the adversary $\mathcal{A}_i$ correctly guesses the challenge bit b used as part of the challenge. For every two consecutive games $game_i$ and $game_{i+1}$, it is shown that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible. Then it is shown that the probability of

success $Pr[S_n]$ of the last game is the target probability, namely 0.5. Hence the probability of success of the first game is negligibly close to the target probability, which ends the proof.

Due to space limitations we provide a detailed proof for index privacy against the CSP only.

5.1 Index Privacy with the CSP as the Adversary

Theorem 1. *Our construction achieves index privacy against the CSP.*

***game*$_0$** Let $game_0$ be the game of Definition 2 (Alg. 1 and Alg. 2). The success event S_0 is "$b = b^*$".

***game*$_1$** The only difference between $game_0$ and $game_1$ is that in $game_1$, the adversary $\mathcal{A}_1$ can no longer send queries requesting the corruption of a user. Consequently $\mathcal{A}_1$ can neither send queries related to corrupted users, namely queries for $QueryTransform$ and $ResponseFilter$.

Lemma 1. *If* $\Pr[S_1]$ *is negligibly close to* 0.5*, then* $\Pr[S_1]$ *and* $\Pr[S_0]$ *are negligibly close.*

Proof. This Lemma is proved by introducing an adversary $\mathcal{A}_1$ executing the algorithm depicted in Alg. 4.

$\mathcal{A}_1$ plays $game_1$ using adversary $\mathcal{A}_0$ playing $game_0$. To that effect, $\mathcal{A}_1$ simulates an instance of $game_0$ with respect to $\mathcal{A}_0$ and responds at $game_1$ using the response of $\mathcal{A}_0$. Since, as opposed to $\mathcal{A}_0$, $\mathcal{A}_1$ cannot corrupt any user, $\mathcal{A}_1$ has to fabricate responses to $\mathcal{A}_0$'s corruption queries as part of the simulated instance of $game_0$. To do so, $\mathcal{A}_1$ simulates corrupted users by locally generating keys which are sent to $\mathcal{A}_0$ as a response to the corruption query. These same generated keys must be used in all responses related to this corrupted user in order for $\mathcal{A}_1$ to simulate a consistent instance of $game_0$. However $\mathcal{A}_0$ may have sent queries related to this user before the corruption query. A way for $\mathcal{A}_1$ to ensure the required consistency is to choose a set of users that will be simulated from the beginning. If $\mathcal{A}_0$ sends a request to corrupt a user that $\mathcal{A}_1$ chose not to simulate, $\mathcal{A}_1$ cannot simulate a proper instance of $game_0$ any more. Simulation also fails if a user that was simulated by $\mathcal{A}_1$ is chosen by $\mathcal{A}_0$ to be the challenge user or a delegate of the challenge user. We define the event C as when none of the previous cases occur, i.e., C is "$\mathcal{A}_0$ does not corrupt any non-simulated user and $\mathcal{A}_0$ does not chose any simulated user as either the challenge user or a delegate of the challenge user". We also define the event C' as "all users but the challenge user and her delegates are simulated". Since C' implies C we have $Pr[C] \geq Pr[C']$, and actually $Pr[C]$ is expected to be much greater than $Pr[C']$. Whenever the event C occurs, $\mathcal{A}_0$ received a valid instance of $game_0$ with the challenge value from the instance of $game_1$, and thus the probability for $\mathcal{A}_1$ to succeed at $game_1$ is the probability of $\mathcal{A}_0$ to succeed at $game_0$:

$$Pr[S_1|C] = Pr[S_0] \; . \tag{2}$$

$\mathcal{A}_1$ receives data from $game_1$ setup phase;
/* $\mathcal{A}_1$ simulates some users to $\mathcal{A}_0$ */
$Sim \xleftarrow{\$} \mathcal{P}([1..N])$;
for $i \in Sim$ **do**
  $(\gamma'_i, \rho'_i, P'_i, K'_i) \leftarrow KeyGen(\kappa)$
end
for i *from* 1 *to* N **do**
  if $i \in Sim$ **then**
    $\mathcal{A}_0 \leftarrow (i, P'_i, K'_i)$;
  else
    $\mathcal{A}_0 \leftarrow (i, P_i, K_i)$
  end
end
/* Learning phase 1 */
for *a polynomial number* l_1 *of times* **do**
  $\mathcal{A}_0 \rightarrow$ query;
  if $\mathcal{A}_1$ *knows all the input values for the corresponding algorithm* **then**
    $\mathcal{A}_1$ runs the algorithm locally and sends back the answer;
  else
    if *query was for corruption* **then**
      /* exit with random guess */
      $b^* \xleftarrow{\$} 0, 1$;
      $\mathcal{A}_1 \rightarrow b^*$;
      HALT;
    else
      $\mathcal{A}_1$ forwards the call to $game_1$ and forwards the answer to $\mathcal{A}_0$;
    end
  end
end
/* Challenge phase */
$\mathcal{A}_1$ forwards everything from $\mathcal{A}_0$ to $game_1$ and back.
/* Learning phase 2 */
Same as learning phase 1;
/* Response phase */
$\mathcal{A}_1$ forwards the bit b^* outputted by $\mathcal{A}_0$;

Algorithm 4: Algorithm run by $\mathcal{A}_1$ the transition adversary from $game_0$ to $game_1$. Restrictions phases are omitted.

If the simulation of $game_0$ fails, $\mathcal{A}_1$ can still give a random answer to $game_1$ which implies:

$$Pr[S_1|\neg C] = 0.5 \ . \tag{3}$$

Finally we define the event C'_i as "user u_i is either simulated or challenge-or-delegate, but not both". We have $Pr[C'_i] = 0.5$ and $Pr[C'] = \prod_{i=1..N} Pr[C'_i]$ thus $Pr[C'] = 2^{-N}$ and it follows that $Pr[C] \geq 2^{-N}$. It seems reasonable to assume that the number N of users grows at most polylogarithmically with the

security parameter κ, which implies that $Pr[C]$ is non-negligible:

$$\exists p \text{ polynomial in } \kappa, \ \frac{1}{Pr[C]} \leq p \ . \tag{4}$$

Then the following holds:

$$\begin{aligned} Pr[S_1] &= Pr[S_1|C].Pr[C] + Pr[S_1|\neg C].Pr[\neg C] \\ Pr[S_1] &= Pr[S_0].Pr[C] + 0.5(1 - Pr[C]) \\ Pr[S_1] &= Pr[C].\left(Pr[S_0] - 0.5\right) + 0.5 \\ Pr[S_0] &= 0.5 + \frac{1}{Pr[C]}\left(Pr[S_1] - 0.5\right) \\ Pr[S_0] - Pr[S_1] &= (0.5 - Pr[S_1])\left(1 - \frac{1}{Pr[C]}\right) \ . \end{aligned}$$

Then from (4) we have that if $(0.5 - Pr[S_1])$ is negligible then $|Pr[S_0] - Pr[S_1]|$ is negligible also. This conclude the proof of Lemma 1.

***game*$_2$** In $game_2$, calls to *QueryCreate* are replaced by the generation of random bits.

Lemma 2. $\Pr[S_2]$ *is negligibly close to* $\Pr[S_1]$.

Proof. Distinguishing between $game_1$ and $game_2$ is equivalent to breaking the security of the encryption scheme used in the PIR construction. This holds because corruption is not allowed in $game_1$, and hence the adversary cannot obtain the required PIR parameters to open the PIR query. It follows that Lemma 2 is true.

***game*$_3$** In $game_3$, the call to $Index$ in the challenge phase is replaced by picking a random element in G_T.

Lemma 3. $\Pr[S_3]$ *is negligibly close to* $\Pr[S_2]$.

Proof. To prove this Lemma we build a distinguishing algorithm $\mathcal{D}_{DDH}$, described in Alg. 5, which uses a $game_2$ adversary $\mathcal{A}_2$ and which advantage at the DDH game is :

$$\epsilon_{DDH} = O(\frac{1}{Nl})|\Pr[S_3] - \Pr[S_2]| \ . \tag{5}$$

Given the DDH problem instance $(g_1, g_1^{\alpha}, g_1^{\beta}, g_1^{\delta}) \in G_1^4$, the intuition behind algorithm $\mathcal{D}_{DDH}$ is to "put" β in the challenge word, α in the challenge user key, and δ in the value given to $\mathcal{A}_2$ during the challenge phase. $\mathcal{D}_{DDH}$ makes some predictions on the queries of $\mathcal{A}_2$, namely on the user $\mathcal{A}_2$ will choose as the challenge user and on the moment $\mathcal{A}_2$ will call the hash function h on the

$\mathcal{D}_{DDH} \leftarrow (g_1, g_1^{\alpha}, g_1^{\beta}, g_1^{\delta})$;
$\mathcal{A} \leftarrow Setup()$;
$predict \xleftarrow{\$} [1..N]$;
$I \xleftarrow{\$} [0, .., l]$;
for *i from* 1 *to* N **do**
  $(\gamma_i, \rho_i, P_i, K_i) \leftarrow KeyGen(\kappa)$;
  $\mathcal{A} \leftarrow (i, P_i, K_i)$
end
for *a polynomial number* l *of times* **do**
  $\mathcal{A} \rightarrow$ query;
  switch *query* **do**
    case *hash of word* w *through* h
      if *this is the* I*-th call to* $\mathcal{O}$ **then**
        $\mathcal{A} \leftarrow g_1^{\beta}$
      else
        $\mathcal{A} \leftarrow g_1^{\mathcal{O}[w]}$
      end
    case *Index for word* w *and user* $u_{predict}$
      $\mathcal{A} \leftarrow e((g_1^{\alpha})^{\mathcal{O}[w]}, g_2)$;
    otherwise
      normal handling of the query;
    end
  endsw
end
$\mathcal{A} \rightarrow (u_{chall}, w_0^*, w_1^*)$;
$b \xleftarrow{\$} \{0, 1\}$;
if *chall* $\neq$ *predict OR* $I = 0$ *and* $\mathcal{O}$ *has been called with input* w_b^* *OR* $I \neq 0$ *and* w_b^* *does not correspond to the* I*-th call to* $\mathcal{O}$ **then**
  $b_{DDH} \xleftarrow{\$} 0, 1$;
  $\mathcal{D}_{DDH} \rightarrow b_{DDH}$;
  `HALT`;
end
$\mathcal{A} \leftarrow e(g_1^{\delta}, g_2)$;
$\mathcal{A} \rightarrow b^*$;
if $b^* = b$ **then**
  $\mathcal{D}_{DDH} \rightarrow 1$;
else
  $\mathcal{D}_{DDH} \rightarrow 0$;
end

Algorithm 5: Listing for the distinguishing algorithm $\mathcal{D}_{DDH}$ from $game_2$ to $game_3$.

challenge word. If these predictions prove false $\mathcal{D}_{DDH}$ sends a random answer to the DDH problem. Otherwise if the predictions prove true $\mathcal{D}_{DDH}$ outputs 1 if $\mathcal{A}_2$ wins the game and 0 if not. If the correct answer to the DDH game was 1 then

$\mathcal{A}_2$ was playing $game_2$ and $\mathcal{D}_{DDH}$ outputs 1 with probability $Pr[S_2]$. Similarly if the answer to DDH was 0 $\mathcal{D}_{DDH}$ outputs 1 with probability $Pr[S_3]$ During the whole game $\mathcal{D}_{DDH}$ simulates the hash function h as a random oracle, using $\mathcal{O}$ which behaves the following way: if $\mathcal{O}$ has stored a value for keyword w, $\mathcal{O}[w]$ returns this value; else it returns a random value and stores it for future queries.

The following variable change shows that if the predictions prove right, the adversary received a proper game instance:

$$\alpha \leftrightarrow \gamma_{chall},\ g_1^{\beta} \leftrightarrow h(w_b^*)\ . \tag{6}$$

The probability that the predictions were correct is clearly non-negligible: $\Pr[u_{predict} = u_{chall}] = 1/N$ and the probability that predicted I is correct is $O(1/l)$, N and l being at most polynomial in the security parameter κ.

Finally from the XDH assumption, DDH is a hard problem in G_1. Thus ϵ_{DDH} is negligible in κ. Given that N and l are at most polynomial in κ and from (5), we have that $|\Pr[S_3] - \Pr[S_2]|$ is negligible which concludes the proof of Lemma 3.

Proof of Theorem 1. In $game_3$ the adversary does not receive any value which depends on the challenge bit, so $\Pr[S_3] = 0.5$. Then Lemma 3 implies that $Pr[S_2]$ is negligibly close to 0.5, Lemma 2 implies that $Pr[S_1]$ is negligibly close to 0.5 and finally Lemma 1 implies that $Pr[S_0]$ is negligibly close to 0.5. This concludes the proof of Theorem 1.

6 Performance Analysis

During the upload phase, the cost for a user of running the $Index$ algorithm over the entire index is naturally linear towards the number of keywords in the index. The most costly operation within the $Index$ algorithm is one pairing computation; however since inside a same index the second argument of the pairing remains the same between two executions of $Index$, pairing becomes much more efficient than in the case with independent pairings [13].

Furthermore, the $Delegate$ algorithm only consists in one exponentiation.

As the search phase involves three parties, namely the reader, the proxy and the CSP, we evaluate the computational cost for each of them.

The $QueryCreate$ algorithm executed by the reader is not costly since it only consists of one hashing and one exponentiation. This algorithm outputs a unique query for a given keyword to be searched in several indices. Therefore, the cost of this algorithm does not depend on the number of searched indices. On the other hand, the reader will receive one response per targeted index and will have to execute $ResponseProcess$ over each received response. The cost for one response consists in one decryption through Dec and one $PIR.Retrieve$ operation. Note that the retrieved value for each index is a single bit, and based on [1] the computational overhead can be considered as reasonable for a lightweight user.

The cost for the proxy of multiplexing the queries with $QueryTransform$ and filtering the responses with $ResponseFilter$ is linear towards the number of

indices the querying reader is authorized to search, and for each queried index the proxy performs a pairing and one execution of $PIR.Query$. The $ResponseFilter$ algorithm can be considered negligible as it only extracts the relevant part of the response.

For a given keyword search query, the CSP builds one matrix per queried index, and executes $PIR.Process$ on each matrix. The building of one matrix requires one exponentiation in G_T per keyword. The operations performed by the CSP being similar to the ones in [9], the workload of the CSP is considered affordable for a cloud server.

To conclude our scheme achieves a very low cost at the reader side, which usually is the main requirement for a cloud computing scenario, and a reasonable cost at the CSP and the proxy. Figure 3 summarizes the cost of each algorithm considering a scenario where one writer uploads several indices and one reader send one query.

Algorithm	**Cost**	**number of executions**
Index	$h + e + exp_{G_1}$	$\mathfrak{i}.\mathfrak{k}$
Delegate	exp_{G_2}	$\mathfrak{i}.\mathfrak{d}$
QueryCreate	$h + mult_{\mathbb{Z}_q} + exp_{G_1}$	
QueryTransform	$\mathfrak{a}(e + H + PIR.KeyGen + PIR.Query)$	
Respond	$\mathfrak{k}(exp_{G_T} + h) + \sqrt{n}(PIR.Process + Enc)$	$\mathfrak{a}$
ResponseFilter	negligible (data forwarding)	$\mathfrak{a}$
ResponseProcess	$Dec + PIR.Retrieve$	$\mathfrak{a}$

Key:

- exp_X: cost of an exponentiation in X
- $mult_X$: cost of a multiplication in X
- $\mathfrak{k}$: number of keyword per index
- $\mathfrak{i}$: number of index owned by a writer
- $\mathfrak{d}$: number of reader with delegated search rights per index
- $\mathfrak{a}$: number of indices the reader is authorized to search
- name of a function: execution cost of this function

Fig. 3. Computational cost of each algorithm.

7 Related Work

Our review of the related work focuses on fully multi-user SE schemes. For a detailed survey on SE in general, we refer the reader to [6].

While solutions in [7,9] seem very efficient in the case where there is a single writer authorizing multiple readers, they become unpractical for the multi writer-multi reader case. Indeed each reader should at least store one key per writer and send one query (even if the same) per writer.

Among the few existing MUSE solutions [3, 5, 8, 12, 15], all of them except the one described in [12] require the existence of a TTP, which is an unpractical assumption that our solution does not make. Finally, all the solutions share a common pitfall as they do not ensure access pattern privacy. As already discussed in this paper, this leads to a serious privacy exposure in the case where users collude with the CSP. Furthermore the execution of $PIR.Process$ in our solution is less costly compared to the search operation at the CSP in all existing MUSE schemes, since in these schemes the trapdoor must be tested with each encrypted keyword in the index either until the test shows that the keyword is present, or until all the keywords in the index have been tested.

8 Conclusion

We have presented a new multi-user searchable encryption scheme that is provably secure under the newly proposed adversarial model witch considers the case where some users can collude with the CSP. All existing schemes become insecure under this new model. The proposed solution is very efficient for the user as it introduces a new party, the proxy, which bears most of the overhead. At the same time this overhead remains reasonable for both the CSP and the proxy.

Future work on this scheme will include implementation and benchmark results of the presented scheme with realistic datasets.

Acknowledgements

The authors thank the anonymous reviewers for their suggestions for improving this paper.

This work was partially funded by the FP7-USERCENTRICNETWORKING european ICT project (grant 611001).

References

1. Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.O.: Xpir: Private information retrieval for everyone. Cryptology ePrint Archive, Report 2014/1025 (2014), `http://eprint.iacr.org/`
2. Aguilar-Melchor, C., Gaborit, P.: A lattice-based computationally-efficient private information retrieval protocol. In: In WEWORC 2007 (2007)
3. Asghar, M.R., Russello, G., Crispo, B., Ion, M.: Supporting complex queries and access policies for multi-user encrypted databases. In: Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop. pp. 77–88. CCSW '13, ACM, New York, NY, USA (2013)
4. Ballard, L., Green, M., de Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, Report 2005/417 (2005), `http://eprint.iacr.org/`
5. Bao, F., Deng, R.H., Ding, X., Yang, Y.: Private query on encrypted data in multi-user settings. In: Information Security Practice and Experience, pp. 71–85. Springer (2008)

6. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A Survey of Provably Secure Searchable Encryption. ACM Computing Surveys 47(2), 1–51 (Aug 2014), http://dl.acm.org/citation.cfm?doid=2658850.2636328
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. Cryptology ePrint Archive, Report 2006/210 (2006), http://eprint.iacr.org/
8. Dong, C., Russello, G., Dulay, N.: Shared and Searchable Encrypted Data for Untrusted Servers. In: Atluri, V. (ed.) Data and Applications Security XXII, Lecture Notes in Computer Science, vol. 5094, pp. 127–143. Springer Berlin Heidelberg (2008)
9. Elkhiyaoui, K., Önen, M., Molva, R.: Privacy preserving delegated word search in the Cloud. In: SECRYPT 2014, 11th International conference on Security and Cryptography, 28-30 August, 2014, Vienna, Austria. Vienna, Austria (2014), http://www.eurecom.fr/publication/4345
10. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 3580, pp. 803–815. Springer Berlin Heidelberg (2005)
11. Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: Zhou, J., Lopez, J., Deng, R., Bao, F. (eds.) Information Security, Lecture Notes in Computer Science, vol. 3650, pp. 314–328. Springer Berlin Heidelberg (2005)
12. Popa, R.A., Zeldovich, N.: Multi-Key Searchable Encryption (2013), http://people.csail.mit.edu/nickolai/papers/popa-multikey-eprint.pdf
13. Scott, M.: On the efficient implementation of pairing-based protocols. In: Chen, L. (ed.) Cryptography and Coding, Lecture Notes in Computer Science, vol. 7089, pp. 296–308. Springer Berlin Heidelberg (2011)
14. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. IACR Cryptology ePrint Archive 2004, 332 (2004), http://www.shoup.net/papers/games.pdf
15. Yang, Y., Lu, H., Weng, J.: Multi-User Private Keyword Search for Cloud Computing. pp. 264–271. IEEE (Nov 2011)

Efficient Techniques for Publicly Verifiable Delegation of Computation

Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui, Refik Molva
EURECOM
Campus SophiaTech
Biot Sophia Antipolis, France
{elkhiyao, onen, azraoui, molva}@eurecom.fr

ABSTRACT

With the advent of cloud computing, individuals and companies alike are looking for opportunities to leverage cloud resources not only for storage but also for computation. Nevertheless, the reliance on the cloud to perform computation raises the unavoidable challenge of how to assure the correctness of the delegated computation. In this regard, we introduce two cryptographic protocols for publicly verifiable computation that allow a lightweight client to securely outsource to a cloud server the evaluation of high-degree univariate polynomials and the multiplication of large matrices. Similarly to existing work, our protocols follow the amortized verifiable computation approach. Furthermore, by exploiting the mathematical properties of polynomials and matrices, they are more efficient and give way to *public delegatability*. Finally, besides their efficiency, our protocols are provably secure under well-studied assumptions.

1. INTRODUCTION

Cloud computing is increasingly becoming an attractive option for SMEs interested in minimizing their expenditures by outsourcing their data and computations. However, the lack of security still deters the wide adoption of cloud technology. As a matter of fact, cloud clients lose control over their data once outsourced, and as such they can neither thwart nor detect cloud servers' misbehavior.

Recently, researchers [6, 11, 13, 17, 19] introduced solutions for verifiable outsourced computation whereby a client delegates the execution of computationally demanding operations to the cloud, and further receives the result with some *cryptographic proofs* asserting the correct execution of requested operations. By definition, these cryptographic proofs fulfill the classical security requirements of *correctness* and *soundness*: They neither yield a situation in which a server is *falsely accused* of misbehavior, nor make the client accept an *incorrect result*.

In addition to the previously mentioned security requirements, another key prerequisite that should be taken into account when designing solutions for verifiable computation is the *efficiency* of the proof verification at the client: For a solution to be viable, the computational and the storage complexity of the verification process should naturally be lower than the complexity of the outsourced function. This requirement thus seeks solutions that minimize the computational and the storage load at lightweight clients, in the aim of not offsetting the advantages of cloud computing.

In order to be able to check the proof of correct computation efficiently, the client generates a verification key: While some solutions [6, 13] keep this verification key secret, in which case only the client verifies the correctness of the outsourced computation, other proposals [11, 17–19] allow *public verifiability* which empowers any third party to verify the validity of the outsourced computation.

Besides public verifiability, several schemes achieve *public delegatability* [17–19]. As the name implies, public delegatability enables any third party to submit computation queries to the outsourced function and verify the returned results. Such a property comes in handy in scenarios where an organization outsources the computation of a function to a cloud server, and still wants its employees to delegate the evaluation of that function without exchanging or sharing any secret keys.

In this paper, we focus on the public verifiability and delegatability of two specific functions, namely, high-degree polynomial evaluation and matrix multiplication. Similarly to existing work, we adopt the *amortized model* [13]: In this model, the client is required to execute a one-time expensive pre-processing operation that is leveraged later for efficient verifications. Furthermore, we suitably tailor the algebraic properties of polynomials and matrices to devise cryptographic solutions that compare favorably to existing work, as they offer better performances and contrary to [11, 20] enable public delegatability.

Contributions:

- We first propose a publicly verifiable polynomial evaluation solution whose efficiency derives from the *Euclidean division* of the polynomial to be outsourced by some randomly generated small-degree polynomial. The basic idea of our solution is that the outsourced polynomial and the quotient polynomial are used to produce the proof of correct computation, whereas the divisor and the remainder polynomials are used together to verify the correctness of the evaluation. Thanks to the properties of Euclidean division, our proposal ensures public delegatability while enjoying better performances than existing work [17].
- Secondly, we propose a solution for publicly verifiable matrix multiplication that exploits the associative property of multiplication in the ring of matrices. As such our solution outperforms the schemes in [11, 20] while ensuring the additional feature of public delegatability.
- Both of our solutions are proved to be correct and sound. Their soundness is proved under the *t-strong Diffie Hellman*

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

 ISBN 978-1-4503-4233-9/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897845.2897910

(t-SDH) and *co-computational Diffie-Hellman (co-CDH)* assumptions.

The rest of the paper is organized as follows. Section II formally defines publicly verifiable computation and the underlying security model. The proposed publicly verifiable polynomial evaluation and matrix multiplication solutions are described and evaluated in Sections III and IV respectively. Finally, we review the state of the art in section V.

2. BACKGROUND

2.1 Publicly Verifiable Computation

According to [18], a publicly verifiable computation scheme empowers a client to outsource the evaluation of a function to a *potentially malicious* server while meeting the requirements of:

- *public delegatability:* Any querier (not necessarily the client) can submit inputs to evaluate the outsourced function;
- *public verifiability:* Any verifier (not necessarily the client or the querier) can assess the correctness of the server's results.

Thus, Parno et al. [18] formally define publicly verifiable computation schemes by the following algorithms:

$\mathsf{Setup}(1^\kappa, \mathfrak{f}) \rightarrow (\mathsf{param}, \mathsf{PK}_\mathfrak{f}, \mathsf{EK}_\mathfrak{f})$ It is a randomized algorithm executed by the client. It takes as input the security parameter 1^κ and a description of the function $\mathfrak{f}$ to be outsourced, and outputs a set of *public parameters* param that will be used by subsequent algorithms, a *public key* $\mathsf{PK}_\mathfrak{f}$, and an *evaluation key* $\mathsf{EK}_\mathfrak{f}$.

$\mathsf{ProbGen}(x, \mathsf{PK}_\mathfrak{f}) \rightarrow (\sigma_x, \mathsf{VK}_x)$ Given an input x in the domain $\mathcal{D}_\mathfrak{f}$ of the outsourced function $\mathfrak{f}$ and public key $\mathsf{PK}_\mathfrak{f}$, the querier calls this algorithm to produce an *encoding* σ_x of input x and a *public verification key* VK_x.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f}) \rightarrow \sigma_y$ On input of the encoding σ_x and the evaluation key $\mathsf{EK}_\mathfrak{f}$, the server runs this algorithm to compute an *encoding* σ_y of $\mathfrak{f}$'s output $y = \mathfrak{f}(x)$.

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \rightarrow \mathsf{out}_y$ A verifier operates this deterministic algorithm to check the correctness of the result σ_y supplied by the server on input σ_x. More precisely, this algorithm first decodes σ_y which yields a value y, and then uses the public verification key VK_x associated with the encoding σ_x to decide whether y is equal to the expected output $\mathfrak{f}(x)$. If so, Verify outputs $\mathsf{out}_y = y$ meaning that $\mathfrak{f}(x) = y$; otherwise it outputs an error $\mathsf{out}_y = \perp$.

Besides the properties of public delegatability and verifiability, a publicly verifiable computation scheme should also ensure the security properties of *correctness* and *soundness*.

2.2 Correctness

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is deemed to be correct, if whenever an *honest* server executes the algorithm $\mathsf{Compute}$ to evaluate a function $\mathfrak{f} \in \mathcal{F}$ on an input $x \in \mathcal{D}_\mathfrak{f}$, this algorithm *always* yields an encoding σ_y that will be accepted by algorithm Verify (i.e. $\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \rightarrow \mathfrak{f}(x)$).

Definition 1. *A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is correct,* **iff** *for any function $\mathfrak{f} \in \mathcal{F}$ and any input $x \in \mathcal{D}_\mathfrak{f}$:*

If $\mathsf{ProbGen}(x, \mathsf{PK}_\mathfrak{f}) \rightarrow (\sigma_x, \mathsf{VK}_x)$ *and* $\mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f}) \rightarrow \sigma_y$*, then:*

$$\Pr(\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \rightarrow \mathfrak{f}(x)) = 1$$

Algorithm 1: Soundness experiment of publicly verifiable computation

$(\mathsf{param}, \mathsf{PK}_\mathfrak{f}, \mathsf{EK}_\mathfrak{f}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\kappa, \mathfrak{f})$;
$\mathcal{A} \rightarrow x$;
$(\sigma_x, \mathsf{VK}_x) \leftarrow \mathcal{O}_{\mathsf{ProbGen}}(x, \mathsf{PK}_\mathfrak{f})$;
$\mathcal{A} \rightarrow \sigma_y$;
$\mathsf{out}_y \leftarrow \mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$;

2.3 Soundness

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is said to be sound, if for any $\mathfrak{f} \in \mathcal{F}$ and for any $x \in \mathcal{D}_\mathfrak{f}$, a server cannot convince a verifier to accept an incorrect result. Notably, a verifiable computation scheme is sound if it assures that the only way a server generates a result σ_y that will be accepted by a verifier as a valid encoding of the evaluation of some function $\mathfrak{f} \in \mathcal{F}$ on an input x, is by correctly computing σ_y (i.e. $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f})$).

Similarly to [18], we capture the adversarial capabilities of an adversary (i.e. malicious server) $\mathcal{A}$ against a publicly verifiable computation scheme for a family of functions $\mathcal{F}$ through a *soundness experiment* (cf. Algorithm 1).

In this experiment, adversary $\mathcal{A}$ first accesses the output of algorithm Setup by calling oracle $\mathcal{O}_{\mathsf{Setup}}$. When queried with a security parameter 1^κ and a description of a function $\mathfrak{f} \in \mathcal{F}$, oracle $\mathcal{O}_{\mathsf{Setup}}$ returns the set of public parameters param, public key $\mathsf{PK}_\mathfrak{f}$, and evaluation key $\mathsf{EK}_\mathfrak{f}$.

Afterwards, adversary $\mathcal{A}$ outputs a challenge input $x \in \mathcal{D}_\mathfrak{f}$ and submits the latter together with public key $\mathsf{PK}_\mathfrak{f}$ to oracle $\mathcal{O}_{\mathsf{ProbGen}}$. Oracle $\mathcal{O}_{\mathsf{ProbGen}}$ accordingly executes algorithm $\mathsf{ProbGen}$ and outputs a pair of matching encoding σ_x and public verification key VK_x.

Finally, adversary $\mathcal{A}$ generates an encoding σ_y and runs algorithm Verify on the pair $(\sigma_y, \mathsf{VK}_x)$.

Let out_y denote the output of algorithm Verify at the end of the experiment. We say that adversary $\mathcal{A}$ succeeds in the soundness experiment of publicly verifiable computation if $\mathsf{out}_y \neq \perp$ and $\mathsf{out}_y \neq \mathfrak{f}(x)$.

Definition 2. *Let $\Pi_{\mathcal{A},\mathfrak{f}}$ denote the probability that adversary $\mathcal{A}$ succeeds in the soundness experiment of publicly verifiable computation (i.e. $\Pr(\mathsf{out}_y \neq \perp \ \wedge \mathsf{out}_y \neq \mathfrak{f}(x))$).*

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is sound, **iff***: For any adversary $\mathcal{A}$ and for any $\mathfrak{f} \in \mathcal{F}$, $\Pi_{\mathcal{A},\mathfrak{f}} \leqslant \epsilon$ and ϵ is a negligible function in the security parameter κ.*

3. PUBLICLY VERIFIABLE POLYNOMIAL EVALUATION

3.1 Protocol Overview

The solution we propose for publicly verifiable evaluation of polynomials draws upon the basic properties of *Euclidean division* of polynomials. Specifically the fact that for any pair of polynomials A and $B \neq 0$ of degree d and 2 respectively, the Euclidean division of A by B yields a unique pair of polynomials Q and R such that: **i.)** $A = QB + R$ and **ii.)** the degree of *quotient* polynomial Q equals $d - 2$, whereas the *remainder* polynomial R has a degree $\leqslant 1$.

Now a client which would like to outsource the evaluation of a polynomial A of degree d, first defines a polynomial $B(X) = X^2 + b_0$ for a randomly chosen b_0, and divides A by B to get the

quotient polynomial $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and the remainder polynomial $R(X) = r_1 X + r_0$. Next, the client outsources polynomial A together with quotient polynomial Q to the server and publishes the public key $\mathsf{PK}_A = (g^{b_0}, g^{r_1}, g^{r_0})$. Consequently, whenever a querier wants to evaluate polynomial A at point x, it first computes and advertises the public verification key $\mathsf{VK}_x = (g^{B(x)}, g^{R(x)})$, and then transmits x to the server. The latter in turn computes $y = A(x)$ and generates the proof $\pi = Q(x)$. Given the server's output (y, π), a verifier checks whether $g^y = (g^{B(x)})^{\pi} g^{R(x)}$.

The efficiency of the verification in the solution sketched above stems from the fact that B and R are small-degree polynomials. Indeed, to verify the correctness of a result (y, π) provided by the server on an input x, the verifier performs a small and constant number of computations as opposed to carrying out $\mathcal{O}(d)$ operations to evaluate polynomial A.

It is clear that the soundness of such a protocol relies on the secrecy of polynomials B and R. However since B is a two-degree polynomial, the secrecy of these two polynomials can be easily compromised by disclosing the quotient polynomial Q. To remedy this shortcoming, the client encodes polynomial Q using an *additively homomorphic one-way* encoding. Namely, each coefficient q_i of polynomial Q is encoded as h^{q_i}. In this manner, we allow the server to compute the proof $\pi = h^{Q(x)}$ of correct execution while ensuring the confidentiality of polynomials B and R.

Finally, we use bilinear pairings to let verifiers assess the correctness of the server's results. Accordingly, we show that our solution is sound under the $\lfloor d/2 \rfloor$-*Strong Diffie-Hellman* ($\lfloor d/2 \rfloor$-SDH) assumption.

Before describing our protocol in full details, we recall the definitions of bilinear pairings and the SDH assumption.

3.2 Bilinear Pairings

Definition 3 (Bilinear Pairing)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite order p.*

A bilinear pairing is a map e: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with the following properties:

1. *e is bilinear: $\forall\ \alpha, \beta \in \mathbb{Z}_p$, $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, $e(g^\alpha, h^\beta) = e(g, h)^{\alpha\beta}$;*
2. *e is computable: There is an efficient algorithm to compute $e(g, h)$ for any $(g, h) \in \mathbb{G}_1 \times \mathbb{G}_2$;*
3. *e is non-degenerate: If g is a generator of $\mathbb{G}_1$ and h is a generator of $\mathbb{G}_2$, then $e(g, h)$ is a generator of $\mathbb{G}_T$.*

Definition 4 (t-SDH Assumption)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite prime order p such that there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*

*We say that the t-**Strong Diffie-Hellman assumption** (t-SDH) holds, if given the tuple $(g, g^\alpha, h, h^\alpha, ..., h^{\alpha^t}) \in \mathbb{G}_1^2 \times \mathbb{G}_2^{t+1}$ for some randomly chosen $\alpha \in \mathbb{F}_p^*$, the probability to produce a pair $(\beta, h^{1/(\beta+\alpha)}) \in \mathbb{F}_p \backslash \{-\alpha\} \times \mathbb{G}_2$ is negligible.*

3.3 Description

We assume here that the client wants to outsource the evaluation of a d-degree polynomial $A(X) = \sum_{i=0}^{d} a_i X^i$ with coefficients $a_i \in \mathbb{F}_p$ where p is a large prime.

$\mathsf{Setup}(1^\kappa, A)$ Given security parameter 1^κ and a description of polynomial A, algorithm Setup first selects two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Then it picks a generator g and a generator h of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and defines the set of public parameters as:

$$\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h).$$

Next, algorithm Setup selects randomly $b_0 \in \mathbb{F}_p^*$ such that polynomial $B(X) = X^2 + b_0$ does not divide polynomial A and performs the Euclidean division of polynomial A by polynomial B in $\mathbb{F}_p[X]$. We denote the resulting quotient polynomial by $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and the resulting remainder polynomial by $R(X) = r_1 X + r_0$[1].

Thereupon, algorithm Setup computes the public key

$$\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) = (g^{b_0}, h^{r_1}, h^{r_0}).$$

To compute evaluation key EK_A algorithm Setup computes $\mathbf{q}_i = h^{q_i} \in \mathbb{G}_2$ for all $0 \leqslant i \leqslant d-2$, and lets

$$\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2}).$$

Algorithm Setup concludes its execution by outputting the tuple $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$.

$\mathsf{ProbGen}(x, \mathsf{PK}_A)$ On input of a point $x \in \mathbb{F}_p$ and public key $\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0)$, algorithm ProbGen first computes

$$\mathsf{VK}_{(x,B)} = \mathbf{b_0} g^{x^2}$$
$$\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0$$

and then outputs the public encoding $\sigma_x = x$ and the public verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_A)$ Given $\sigma_x = x$ and evaluation key $\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2})$, algorithm Compute evaluates

$$y = A(x) = \sum_{i=0}^{d} a_i x^i \mod p,$$

generates the proof

$$\pi = \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i},$$

and outputs the encoding $\sigma_y = (y, \pi)$.

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$ Provided with encoding $\sigma_y = (y, \pi)$ and verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$, algorithm Verify checks whether the following equation holds:

$$e(g, h^y) = e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)}). \quad (1)$$

If so, then Verify outputs y meaning that $A(x) = y$; otherwise it outputs $\perp$.

3.4 Security Analysis

Here we state and prove the main security theorems pertaining to our protocol for publicly verifiable polynomial evaluation.

Theorem 1. *The scheme proposed above for publicly verifiable polynomial evaluation is correct.*

[1] R is a polynomial of degree at most 1, i.e. r_1 could be 0.

Proof. If on input $\sigma_x = x \in \mathbb{F}_p$, the server executes algorithm $\mathsf{Compute}$ correctly, then the latter's output will correspond to

$$\sigma_y = (y, \pi) = (A(x), h^{Q(x)}).$$

Indeed, we have:

$$\begin{aligned}\pi &= \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i} = \prod_{i=0}^{d-2} h^{q_i x^i} = h^{\sum_{i=0}^{d-2} q_i x^i} \\ &= h^{Q(x)}.\end{aligned}$$

Given that $A = QB + R$ in $\mathbb{F}_p[X]$ and that the order of $e(g,h)$ is equal to p, we get:

$$\begin{aligned}e(g,h)^{A(x)} &= e(g,h)^{Q(x)B(x)+R(x)} \\ &= e(g, h^{Q(x)})^{B(x)} e(g,h)^{R(x)}.\end{aligned}$$

As $y = A(x)$ and $\pi = h^{Q(x)}$ we have:

$$\begin{aligned}e(g,h)^{y} &= e(g,\pi)^{B(x)} e(g,h)^{R(x)} \\ &= e(g^{B(x)}, \pi) e(g, h^{R(x)}).\end{aligned}$$

Since

$$\mathsf{VK}_{(x,B)} = \mathbf{b}_0 g^{x^2} = g^{b_0 + x^2} = g^{B(x)}$$

and

$$\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0 = h^{r_1 x + r_0} = h^{R(x)},$$

we conclude that

$$e(g,h)^y = e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)})$$

and that Verify outputs $y = A(x)$. □

Theorem 2. *The scheme proposed above for publicly verifiable polynomial evaluation is sound under the $\lfloor d/2 \rfloor$-SDH assumption.*

Proof. Assume there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable polynomial evaluation with a non-negligible advantage ϵ. We demonstrate in what follows that there exists another adversary $\mathcal{B}$ that breaks the $\lfloor d/2 \rfloor$-SDH assumption with a non-negligible advantage $\geqslant \epsilon$.

Let $\mathcal{O}_{\mathsf{sdh}}$ be an oracle which when queried returns the pair (g, g^α) in $\mathbb{G}_1$ and the tuple $(h, h^\alpha, h^{\alpha^2}, ..., h^{\alpha^{\lfloor d/2 \rfloor}})$ in $\mathbb{G}_2$ for randomly generated α in $\mathbb{F}_p^*$.

In order to break $\lfloor d/2 \rfloor$-SDH, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{sdh}}$ to obtain a tuple $(g, g^\alpha, h, h^\alpha, ..., h^{\alpha^{\lfloor d/2 \rfloor}})$; then simulates the soundness experiment (see Algorithm 1) to adversary $\mathcal{A}$. Namely, when $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with polynomial $A(X) = \sum_{i=0}^{d} a_i X^i$ in $\mathbb{F}_p[X]$, adversary $\mathcal{B}$ simulates $\mathcal{O}_{\mathsf{Setup}}$'s response as follows:

1. It defines the public parameters

$$\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$$

2. To compute the evaluation key $\widehat{\mathsf{EK}}_A = (A, \hat{\mathbf{q}}_0, ..., \hat{\mathbf{q}}_{d-2})$, it proceeds as described below:

   - It lets $\hat{\mathbf{q}}_{d-2} = h^{a_d}$ and $\hat{\mathbf{q}}_{d-3} = h^{a_{d-1}}$;
   - For each $2 \leqslant k \leqslant d-2$, it computes

$$\hat{\mathbf{q}}_{d-2-k} = \prod_{i=0}^{\lfloor k/2 \rfloor} h^{a_{d-k+2i}(-1)^i \alpha^i}$$

3. It computes the public key $\widehat{\mathsf{PK}}_A = (\hat{\mathbf{b}}_0, \hat{\mathbf{r}}_1, \hat{\mathbf{r}}_0)$ as following:

$$\begin{aligned}\hat{\mathbf{b}}_0 &= g^\alpha \\ \hat{\mathbf{r}}_0 &= \prod_{i=0}^{\lfloor d/2 \rfloor} h^{a_{2i}(-1)^i \alpha^i} \\ \hat{\mathbf{r}}_1 &= \prod_{i=0}^{\lfloor (d-1)/2 \rfloor} h^{a_{2i+1}(-1)^i \alpha^i}.\end{aligned}$$

   If $(\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1) = (1,1)$, then adversary $\mathcal{B}$ stops the experiment.

4. Otherwise, it returns public parameters $\widehat{\mathsf{param}}$, evaluation key $\widehat{\mathsf{EK}}_A$ and public key $\widehat{\mathsf{PK}}_A$ to adversary $\mathcal{A}$.

It can easily be shown that if adversary $\mathcal{B}$ does not stop the experiment, then the distribution of the tuple $(\widehat{\mathsf{param}}, \widehat{\mathsf{PK}}_A, \widehat{\mathsf{EK}}_A)$ returned by adversary $\mathcal{B}$ is *statistically indistinguishable* from the distribution of $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$ in the soundness experiment. As a matter of fact, if we denote for all $0 \leqslant i \leqslant d-2$, $\hat{\mathbf{q}}_i = h^{q_i}$ and if we let $(\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1) = (h^{r_0}, h^{r_1})$, then we can easily verify that:

- $a_d = q_{d-2} \mod p$ and $a_{d-1} = q_{d-3} \mod p$;
- for all $2 \leqslant i \leqslant d-2$, $a_i = \alpha q_i + q_{i-2} \mod p$;
- $a_1 = \alpha q_1 + r_1 \mod p$ and $a_0 = \alpha q_0 + r_0 \mod p$;
- $(r_0, r_1) \neq (0,0)$.

This entails that the polynomials defined as $Q(X) = \sum_{i=0}^{d-2} q_i X^i$, $B(X) = X^2 + \alpha$ and $R(X) = r_1 X + r_0$ verify the following equality: $A = BQ + R$ with $R \neq 0$.

Therefore we can safely conclude (i) that polynomial B does not divide polynomial A; (ii) that each $\hat{\mathbf{q}}_i$ correctly encodes the i^{th} coefficient of the quotient polynomial Q that results from the Euclidean division of polynomial A by polynomial B; (iii) that the pair $(\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1)$ correctly encodes the corresponding remainder polynomial R.

Eventually, adversary $\mathcal{A}$ selects a challenge value $x \in \mathbb{F}_p$ and calls oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(x, \widehat{\mathsf{PK}}_A)$. Accordingly, adversary $\mathcal{B}$ computes the response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and returns verification key

$$\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)}) = (\hat{\mathbf{b}}_0 g^{x^2}, \hat{\mathbf{r}}_0 \hat{\mathbf{r}}_1^x).$$

Finally, adversary $\mathcal{A}$ returns a pair (y, π) such that $y \neq A(x)$ and (y, π) is accepted by algorithm Verify with a non-negligible advantage ϵ.

Consequently, adversary $\mathcal{B}$ breaks $\lfloor d/2 \rfloor$-SDH by first computing $A(x)$ and the proof

$$\pi^* = \prod_{i=0}^{d-2} \hat{\mathbf{q}}_i^{x^i}$$

and finally outputting:

$$(\beta, h^{1/(\beta+\alpha)}) = \left(x^2, \left(\frac{\pi}{\pi^*} \right)^{(y - A(x))^{-1}} \right).$$

Indeed, since the pair (y, π) passes the verification, it satisfies Equation 1, namely:

$$e(g,h)^y = e(\hat{\mathbf{b}}_0 g^{x^2}, \pi) e(g, \hat{\mathbf{r}}_0 \hat{\mathbf{r}}_1^x) = e(g^{x^2+\alpha}, \pi) e(g, \hat{\mathbf{r}}_0 \hat{\mathbf{r}}_1^x). \tag{2}$$

Furthermore, by construction:

$$e(g,h)^{A(x)} = e(g^{x^2+\alpha}, \pi^*) e(g, \hat{\mathbf{r}}_0 \hat{\mathbf{r}}_1^x). \tag{3}$$

Algorithm	Computation	Client's storage	Server's storage
Setup	1 prng and d mul in $\mathbb{F}_p$ 1 exp in $\mathbb{G}_1$ $d+1$ exp in $\mathbb{G}_2$	$\mathcal{O}(1)$	$\mathcal{O}(d)$
ProbGen	1 mul in $\mathbb{F}_p$ 1 exp and 1 mul in $\mathbb{G}_1$ 1 exp and 1 mul in $\mathbb{G}_2$	–	–
Compute	$2d-3$ mul in $\mathbb{F}_p$ $d-1$ exp and $d-2$ mul in $\mathbb{G}_2$	–	–
Verify	1 exp and 1 div in $\mathbb{G}_2$ 2 pairings	–	–

Table 1: Computation and storage requirements of our protocol for publicly verifiable polynomial evaluation

By dividing Equation 2 by 3, we obtain:

$$e(g,h)^{(y-A(x))} = e\left(g^{x^2+\alpha}, \frac{\pi}{\pi*}\right).$$

Since $y \neq A(x)$, the above equation implies:

$$e(g,h) = e\left(g^{x^2+\alpha}, \left(\frac{\pi}{\pi*}\right)^{(y-A(x))^{-1}}\right).$$

Hence if adversary $\mathcal{B}$ does not stop the experiment, then it will be able to break the $\lfloor d/2 \rfloor$-SDH assumption.

Now if adversary $\mathcal{B}$ aborts the experiment which occurs when $(\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1) = (1,1)$, then adversary $\mathcal{B}$ can conclude that B divides A. This means that by using a factorization algorithm in $\mathbb{F}_p[X]$ on polynomial A, adversary $\mathcal{B}$ will be able to find α, and therewith, break the $\lfloor d/2 \rfloor$-SDH assumption.

Thus, we deduce that if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable polynomial evaluation with a non-negligible advantage ϵ, then there is an adversary $\mathcal{B}$ that breaks the $\lfloor d/2 \rfloor$-SDH assumption with a non-negligible advantage $\geqslant \epsilon$. □

Remark 1. *Notice that if $B(X) = X^\delta + b_0$, then using a similar argument as the one above, we can easily show that our protocol for verifiable polynomial evaluation is secure under the t-SDH assumption for $t \geqslant \lfloor d/\delta \rfloor$.*

3.5 Performance Analysis

The reader may refer to Table 1 for a summary of the performances of our protocol for publicly verifiable polynomial evaluation.

Algorithm Setup first generates a random coefficient $b_0 \in \mathbb{F}_p^*$ to construct polynomial B and conducts an Euclidean division of polynomial A by polynomial B. The latter operation consists of d multiplications and additions, where d is the degree of polynomial A. Once the Euclidean division is performed, algorithm Setup performs one exponentiation in $\mathbb{G}_1$ to derive $\mathbf{b}_0$, and $d+1$ exponentiations in $\mathbb{G}_2$ to compute $\mathbf{r}_0$, $\mathbf{r}_1$ and $\mathbf{q}_i$. Although computationally expensive, algorithm Setup is executed only once by the client. Besides, its computational cost is *amortized* over the large number of verifications that third-party verifiers can carry out.

On the other hand, algorithm ProbGen computes the verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$ which demands a constant number of operations that does not depend on the degree of polynomial A. More precisely, ProbGen's work consists of computing x^2 in $\mathbb{F}_p$, performing one exponentiation and one multiplication in $\mathbb{G}_1$ to get $\mathsf{VK}_{(x,B)} = g^{B(x)}$, and running one exponentiation and one multiplication in $\mathbb{G}_2$ to obtain $\mathsf{VK}_{(x,R)} = h^{R(x)}$.

Furthermore, algorithm Compute runs in two steps: (i) the evaluation of polynomial A at point x which requires at most d additions and multiplications in $\mathbb{F}_p$ if the server uses *Horner's rule*; and (ii) the generation of the proof π which involves $d-3$ multiplications in $\mathbb{F}_p$ and $d-1$ exponentiations and $d-2$ multiplications in $\mathbb{G}_2$.

Finally, the work at third-party verifiers only consists of one exponentiation and one division in $\mathbb{G}_2$ and the computation of 2 bilinear pairings.

With respect to storage, the client is required to store and publish the public key $(\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) \in \mathbb{G}_1 \times \mathbb{G}_2^2$. The server however keeps the $d+1$ coefficients $a_i \in \mathbb{F}_p$ of polynomial A and the $d-1$ encodings $\mathbf{q}_i \in \mathbb{G}_2$.

The reader may refer to Table 1 for a summary of the performances of our protocol for publicly verifiable polynomial evaluation.

4. PUBLICLY VERIFIABLE MATRIX MULTIPLICATION

4.1 Protocol Overview

The protocol we introduce in this section relies on the intuition already expressed in [11], which states that in order to verify that a server correctly multiplies an (n,m)-matrix M of elements M_{ij} with some column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$, it suffices that the client randomly picks a *secret* (n,m)-matrix R of elements R_{ij}, and supplies a server with (n,m)-matrix M and an auxiliary (n,m)-matrix $\mathcal{N}$ such that $\mathcal{N}_{ij} = \tilde{g}^{M_{ij}} g^{R_{ij}}$ (where $\tilde{g} = g^\delta$ for some randomly generated δ). Consequently, when a client prompts the server to multiply matrix M with vector $\vec{x}$, the latter returns vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ and proof $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)^\intercal$, such that $\pi_i = \tilde{g}^{y_i} g^{\sum_{j=1}^{m} R_{ij} x_j}$ if the server is honest. If we denote $\pi_i = g^{\gamma_i}$ and $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n)^\intercal$, then loosely speaking, the verification process consists of checking whether $\vec{\gamma} = \delta\vec{y} + R\vec{x}$.

Now to transform this intuition into a viable solution, one must ensure that the verification process is much less computationally demanding than the matrix multiplication $M\vec{x}$ for all vectors $\vec{x}$. In [11], the authors speed up the verification process by generating the secret matrix R using dedicated *algebraic PRFs* that optimize the multiplication $R\vec{x}$. Although this solution gives way to an efficient verification process that takes $\mathcal{O}(n+m)$ time, it does not enable public delegatability: Only the client can submit multiplication queries to the server.

We tackle this issue by observing that for any vector $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$, the verification of whether $\vec{\lambda}\vec{\gamma} = \delta\vec{\lambda}\vec{y} + \vec{\lambda}(R\vec{x})$ takes $\mathcal{O}(n)$ time if the vector $\vec{\lambda}R$ is computed beforehand. Therefore, we define the public key by an exponent encoding of $\vec{\lambda}R$, and the verification key for vector $\vec{x}$ by an exponent encoding of $(\vec{\lambda}R)\vec{x}$.

More concretely, we generate the elements in the auxiliary matrix $\mathcal{N}$ as $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$ for $g_i = g^{\lambda_i}$, we let the public key PK_M be a vector of m components $\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$, and we compute the verification key for vector $\vec{x}$ as $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$. Therefore, the problem generation combined with the verification take $\mathcal{O}(n+m)$ time as opposed to performing $\mathcal{O}(nm)$ operations to compute the matrix multiplication $\vec{y} = M\vec{x}$.

As a result, the proposed solution does not only offer public delegatability, but also is sound under the assumption of co-computational Diffie-Hellman (*co-CDH*).

Definition 5 (co-CDH Assumption)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite prime order p such that there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.*

We say that the **co-computational Diffie-Hellman assumption** *(co-CDH) holds in $\mathbb{G}_1$, if given $g, g^{\alpha} \in \mathbb{G}_1$ and $h, h^{\beta} \in \mathbb{G}_2$ for random $\alpha, \beta \in \mathbb{F}_p^*$, the probability to compute $g^{\alpha\beta}$ is negligible.*

4.2 Protocol for Verifiable Matrix Multiplication

Without loss of generality, we assume that a client outsources to a server the multiplication operations involving an (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ ($1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$) with p being a large prime.

$\mathsf{Setup}(1^{\kappa}, M)$ Given security parameter 1^{κ} and matrix M, algorithm Setup chooses two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. It then selects a generator h of group $\mathbb{G}_2$ and computes $\tilde{h} = h^{\delta}$ for a randomly selected δ in $\mathbb{F}_p^*$. Thereafter, it randomly picks n generators[2] g_i of $\mathbb{G}_1$, for all $1 \leqslant i \leqslant n$. Subsequently, algorithm Setup defines the public parameters associated with matrix M as:

$$\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{g_i\}_{1 \leqslant i \leqslant n}, h, \tilde{h}).$$

Afterwards, algorithm Setup computes the evaluation key EK_M as follows:

- It selects an (n, m)-random matrix R of elements R_{ij} in $\mathbb{F}_p^*$.
- It derives another (n, m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} = g_i^{\delta M_{ij} + R_{ij}}$, $\forall\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m$.
- Finally, it sets the evaluation key to

$$\mathsf{EK}_M = (M, \mathcal{N}).$$

Next, algorithm Setup determines public key PK_M as depicted hereafter:

- It generates m keys $\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$, $1 \leqslant j \leqslant m$.
- Then, it lets $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$.

At the end of its execution, algorithm Setup outputs public parameters param, public key PK_M and evaluation key EK_M.

[2]Without loss of generality, we can assume that $g_i = g^{\lambda_i}$ for random λ_i in $\mathbb{F}_p^*$.

$\mathsf{ProbGen}(\vec{x}, \mathsf{PK}_M)$ On input of a column vector $\vec{x} = (x_1, x_2 ..., x_m)^{\intercal}$ in $\mathbb{F}_p^m$ and public key $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$ associated with matrix M, algorithm $\mathsf{ProbGen}$ derives

$$\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$$

and returns the encoding $\sigma_x = \vec{x}$ and the verification key VK_x.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_M)$ Provided with encoding $\sigma_x = \vec{x} = (x_1, x_2, ..., x_m)^{\intercal}$ and evaluation key $\mathsf{EK}_M = (M, \mathcal{N})$, algorithm $\mathsf{Compute}$ multiplies matrix M with vector $\vec{x}$ which yields a column vector $\vec{y} = (y_1, y_2, ..., y_n)^{\intercal}$, evaluates the product:

$$\Pi = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$$

and outputs the encoding $\sigma_y = (\vec{y}, \Pi)$.

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$ Given $\sigma_y = (\vec{y}, \Pi)$ and verification key VK_x, algorithm Verify checks whether the following equality holds:

$$e(\Pi, h) \stackrel{?}{=} e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x. \tag{4}$$

If so, algorithm Verify outputs $\vec{y}$ meaning that $M\vec{x} = \vec{y}$; otherwise it outputs $\perp$.

4.3 Security Analysis

In this section, we formally prove the security properties of our solution for publicly verifiable matrix multiplication.

Theorem 3. *The solution described above for publicly verifiable matrix multiplication is correct.*

Proof. If when queried with vector $\vec{x} = (x_1, x_2, ..., x_n)^{\intercal}$, the server correctly operates algorithm $\mathsf{Compute}$, then Equation 4 always holds.

Actually in that case, σ_y corresponds to the pair $(\vec{y}, \Pi)$ such that $\vec{y} = (y_1, y_2, ..., y_n)^{\intercal} = M\vec{x}$ and $\Pi = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$. This implies that for all $1 \leqslant i \leqslant n$: $y_i = \sum_{j=1}^{m} M_{ij} x_j \mod p$, and as the order of g_i is p, it also implies that:

$$\begin{aligned}
\Pi &= \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j} = \prod_{i=1}^{n} \prod_{j=1}^{m} \left(g_i^{\delta M_{ij} + R_{ij}}\right)^{x_j} \\
&= \prod_{i=1}^{n} \prod_{j=1}^{m} \left(g_i^{\delta M_{ij} x_j + R_{ij} x_j}\right) = \prod_{i=1}^{n} \prod_{j=1}^{m} g_i^{\delta M_{ij} x_j} g_i^{R_{ij} x_j} \\
&= \prod_{i=1}^{n} g_i^{\delta \sum_{j=1}^{m} M_{ij} x_j} \prod_{i=1}^{n} \prod_{j=1}^{m} g_i^{R_{ij} x_j} \\
&= \prod_{i=1}^{n} g_i^{\delta y_i} \prod_{i=1}^{n} \prod_{j=1}^{m} g_i^{R_{ij} x_j}
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}
e(\Pi, h) &= e(\prod_{i=1}^{n} g_i^{\delta y_i} \prod_{i=1}^{n} \prod_{j=1}^{m} g_i^{R_{ij} x_j}, h) \\
&= e(\prod_{i=1}^{n} g_i^{y_i}, h^{\delta}) e(\prod_{i=1}^{n} \prod_{j=1}^{m} g_i^{R_{ij} x_j}, h) \\
&= e(\prod_{i=1}^{n} g_i^{y_i}, h^{\delta}) \prod_{j=1}^{m} e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)^{x_j}
\end{aligned}$$

As $\tilde{h} = h^{\delta}$ and $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$, where

$$\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$$

we get:

$$e(\Pi, h) = e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x$$

and we conclude that Verify outputs $\vec{y} = M\vec{x}$. □

Theorem 4. *The solution described above for publicly verifiable matrix multiplication is sound under the co-CDH assumption in* $\mathbb{G}_1$.

Proof. Assume there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable delegation of matrix multiplication with a non-negligible advantage ϵ. We show in what follows how an adversary $\mathcal{B}$ can use adversary $\mathcal{A}$ to break the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \simeq \epsilon$.

To break the co-CDH assumption, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{co-cdh}}$ which in turn outputs the pair $(g, g^{\alpha}) \in \mathbb{G}_1^2$ and the pair $(h, h^{\beta}) \in \mathbb{G}_2^2$.

Later, adversary $\mathcal{B}$ simulates the soundness experiment (cf. Algorithm 1) to adversary $\mathcal{A}$ as following:

When adversary $\mathcal{A}$ calls the oracle $\mathcal{O}_{\mathsf{Setup}}$ with some matrix M of elements M_{ij} in $\mathbb{F}_p$, adversary $\mathcal{B}$ simulates the oracle $\mathcal{O}_{\mathsf{Setup}}$ of the soundness experiment by executing algorithm Setup as depicted in Section 4.2 except for the following:

1. It lets $\widehat{g} = g^{\alpha}$ and $\widehat{h} = (h^{\beta})^{\delta}$, computes for all $1 \leqslant i \leqslant n$, $\widehat{g}_i = \widehat{g}^{\lambda_i}$ for some randomly chosen $\lambda_i \in \mathbb{F}_p^*$, and sets the public parameters to
$$\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\widehat{g}_i\}_{1 \leqslant i \leqslant n}, h, \widehat{h}).$$
2. It generates an (n, m)-random matrix $\widehat{\mathcal{N}}$ of elements $\widehat{\mathcal{N}}_{ij} \in \mathbb{G}_1$;
3. It computes for all $1 \leqslant j \leqslant m$,
$$\widehat{\mathsf{PK}}_j = \frac{e(\prod_{i=1}^{n} \widehat{\mathcal{N}}_{ij}, h)}{e(\prod_{i=1}^{n} \widehat{g}_i^{M_{ij}}, \widehat{h})}; \tag{5}$$
4. It defines the public key associated with matrix M as $\widehat{\mathsf{PK}}_M = (\widehat{\mathsf{PK}}_1, ..., \widehat{\mathsf{PK}}_m)$;
5. Finally, it sets the corresponding evaluation key to $\widehat{\mathsf{EK}}_M = (M, \widehat{\mathcal{N}})$.

Adversary $\mathcal{B}$ concludes its simulation of the oracle $\mathcal{O}_{\mathsf{Setup}}$ by outputting public parameters $\widehat{\mathsf{param}}$, public key $\widehat{\mathsf{PK}}_M$ and evaluation key $\widehat{\mathsf{EK}}_M$.

Note here that the simulated output of oracle $\mathcal{O}_{\mathsf{Setup}}$ in the game is statistically indistinguishable from the distribution of the output of algorithm Setup in the soundness experiment. Namely, the following is true:

- The statistical distribution of matrix $\widehat{\mathcal{N}}$ is identical to the distribution of matrix $\mathcal{N}$ generated by algorithm Setup.
- For all vectors $\vec{x} = (x_1, ..., x_m)^{\intercal} \in \mathbb{F}_p^m$ and $\vec{y} = (y_1, ..., y_n)^{\intercal} = M\vec{x}$, the simulated public key $\widehat{\mathsf{PK}}_M = (\widehat{\mathsf{PK}}_1, ..., \widehat{\mathsf{PK}}_m)$ verifies this equation:
$$e(\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}, h) = e(\prod_{i=1}^{n} \widehat{g}_i^{y_i}, \widehat{h}) \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{x_j}.$$
Therefore, we conclude that the distribution of matrix $\widehat{\mathcal{N}}$ and public key $\widehat{\mathsf{PK}}_M$ is the same as the distribution of matrix $\mathcal{N}$ and $\mathsf{PK}_M = (\mathsf{PK}_1, ..., \mathsf{PK}_m)$ produced by algorithm Setup.

At the end of the experiment, adversary $\mathcal{A}$ picks a challenge vector $\vec{x} = (x_1, x_2, ..., x_m)^{\intercal}$ and queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(\vec{x}, \widehat{\mathsf{PK}}_M)$.

As a result, adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and outputs the pair $(\vec{x}, \widehat{\mathsf{VK}}_x)$ with $\widehat{\mathsf{VK}}_x = \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{x_j}$.

Afterwards, adversary $\mathcal{A}$ returns a response $\sigma_y = (\vec{y}, \Pi)$ such that $\vec{y} \neq M\vec{x}$.

In the remainder of this proof, we denote $\vec{y}^* = (y_1^*, y_2^*, ..., y_n^*)^{\intercal} = M\vec{x}$.

To break the co-CDH assumption in $\mathbb{G}_1$, adversary $\mathcal{B}$ first fetches the vector $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$ used to compute the powers $\widehat{g}_i = \widehat{g}^{\lambda_i}$ and verifies whether $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$. If so, adversary $\mathcal{B}$ aborts the game; otherwise it breaks co-CDH by returning:

$$g^{\alpha\beta} = \left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}\right)^{(\delta\vec{\lambda}(\vec{y}-\vec{y}^*))^{-1}}.$$

Indeed, if $\sigma_y = (\vec{y}, \Pi)$ passes the verification, then this implies that the following equation holds:

$$e(\Pi, h) = e(\prod_{i=1}^{n} \widehat{g}_i^{y_i}, \widehat{h})\widehat{\mathsf{VK}}_x. \tag{6}$$

Also given Equation 5, we have:

$$e(\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}, h) = e(\prod_{i=1}^{n} \widehat{g}_i^{y_i^*}, \widehat{h})\widehat{\mathsf{VK}}_x \tag{7}$$

By dividing Equation 6 with Equation 7, we obtain:

$$\begin{aligned} e\left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(\prod_{i=1}^{n} \widehat{g}_i^{y_i - y_i^*}, \widehat{h}\right) \\ &= e\left(\prod_{i=1}^{n} \widehat{g}^{\lambda_i(y_i - y_i^*)}, \widehat{h}\right) \\ &= e\left(\widehat{g}^{\sum_{i=1}^{n} \lambda_i(y_i - y_i^*)}, \widehat{h}\right) \\ &= e\left(\widehat{g}^{\vec{\lambda}(\vec{y}-\vec{y}^*)}, \widehat{h}\right) \end{aligned}$$

As $\widehat{g} = g^{\alpha}$ and $\widehat{h} = h^{\beta\delta}$, we deduce that

$$\begin{aligned} e\left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(g^{\alpha\vec{\lambda}(\vec{y}-\vec{y}^*)}, h^{\beta\delta}\right) \\ &= e\left(g^{\alpha\beta}, h\right)^{\delta\vec{\lambda}(\vec{y}-\vec{y}^*)} \end{aligned}$$

Therefore if $\vec{\lambda}(\vec{y}-\vec{y}^*) \neq 0 \mod p$, then $\delta\vec{\lambda}(\vec{y}-\vec{y}^*) \neq 0 \mod p$ ($\delta \in \mathbb{F}_p^*$) and we can compute:

$$g^{\alpha\beta} = \left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}\right)^{(\delta\vec{\lambda}(\vec{y}-\vec{y}^*))^{-1}}$$

Hence, adversary $\mathcal{B}$ breaks the co-CDH assumption in $\mathbb{G}_1$ as long

Algorithm	Computation cost	Client's storage	Server's storage
Setup	nm prng in $\mathbb{F}_p$ and nm mul in $\mathbb{F}_p$ $m(n-1)$ mul and $2nm$ exp in $\mathbb{G}_1$ m pairings	$\mathcal{O}(n+m)$	$\mathcal{O}(nm)$
ProbGen	$(m-1)$ mul and m exp in $\mathbb{G}_T$	–	–
Compute	nm mul in $\mathbb{F}_p$ $(n-1)(m-1)$ mul and nm exp in $\mathbb{G}_1$	–	–
Verify	$(n-1)$ mul and n exp in $\mathbb{G}_1$ 1 mul in $\mathbb{G}_T$ 2 pairings	–	–

Table 2: Computation and storage requirements of our protocol for publicly verifiable matrix multiplication

as $\vec{\lambda}\vec{y} \neq \vec{\lambda}\vec{y}^* \mod p$. Fortunately, under the hardness of discrete logarithm, the probability that adversary $\mathcal{B}$ finds $\vec{y}$ such that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$ is negligible.

Lemma 1. *If adversary $\mathcal{A}$ outputs $\vec{y}$ such that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$, then adversary $\mathcal{B}$ can break the discrete logarithm (DL) assumption in $\mathbb{G}_1$.*

Proof Sketch. Assume there is an adversary $\mathcal{A}$ that outputs a vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ verifying the property above with a non-negligible advantage ϵ. Here we show that there is another adversary $\mathcal{B}$ which uses adversary $\mathcal{A}$ to break the discrete logarithm assumption in $\mathbb{G}_1$ with a non-negligible advantage $\geqslant \epsilon/n$.

Assume that adversary $\mathcal{B}$ receives $\breve{g} \in \mathbb{G}_1$ and is required to output $\lambda \in \mathbb{F}_p$ such that $\breve{g} = g^\lambda$.

To this effect, adversary $\mathcal{B}$ simulates the soundness experiment as depicted in Algorithm 1. More precisely, upon receipt of an (n, m)-matrix M, it simulates the output of $\mathcal{O}_{\mathsf{Setup}}$ exactly as the soundness experiment except for the following:

- It selects k randomly in $\{1, 2, ..., n\}$ and lets $\breve{g}_k = \breve{g}$;
- for all $1 \leqslant i \leqslant n$, $i \neq k$, it randomly selects $\lambda_i \in \mathbb{F}_p^*$ and sets $\breve{g}_i = \breve{g}^{\lambda_i}$;
- it sets the public parameters to $\widetilde{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\breve{g}_i\}_{1\leqslant i\leqslant n}, h, \tilde{h})$.

Adversary $\mathcal{A}$ eventually returns a pair of vectors $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ that verify $\vec{y} \neq M\vec{x}$ and $\vec{\lambda}\vec{y} = \vec{\lambda}M\vec{x} \mod p$, whereby $\vec{\lambda} = (\lambda_1, ..., \lambda_{k-1}, \lambda, \lambda_{k+1}, ..., \lambda_n)$.

If we denote $\vec{y}^* = (y_1^*, y_2^*, ..., y_n^*)^\intercal = M\vec{x}$, then the above equality entails that

$$\lambda = \frac{\sum_{i=1, i\neq k}^{n} \lambda_i (y_i^* - y_i)}{y_k - y_k^*}$$

as long as $y_k \neq y_k^*$.

Since $\vec{y} \neq \vec{y}^*$, then there is at least one index $1 \leqslant j \leqslant n$ such that $y_j \neq y_j^*$. Since k is randomly chosen from $\{1, ..., n\}$, the probability that $y_k \neq y_k^*$ is at least $1/n$, and consequently, adversary $\mathcal{B}$ will be able to break the discrete logarithm assumption with advantage $\geqslant \epsilon/n$. □

To summarize, if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable matrix multiplication with a non-negligible advantage ϵ, then there exists an adversary $\mathcal{B}$ that breaks the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \simeq \epsilon$. □

4.4 Performance Analysis

Algorithm Setup generates the (n, m)-random matrix R which requires the generation of nm random numbers in $\mathbb{F}_p$. To compute the elements $\mathcal{N}_{ij}$ of matrix $\mathcal{N}$ as $g_i^{\delta M_{ij} + R_{ij}}$, algorithm Setup performs nm multiplications and nm additions in $\mathbb{F}_p$, and nm exponentiations in $\mathbb{G}_1$. Furthermore, the generation of public key PK_M demands $m(n-1)$ multiplications in $\mathbb{G}_1$, nm exponentiations in $\mathbb{G}_1$ and m pairings. It should be noted that while algorithm Setup involves expensive operations such as exponentiations and pairings, it is executed only once by the client, and consequently, its cost is *amortized* over the large number of verifications that a verifier can perform.

To multiply a vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ with matrix M, algorithm ProbGen computes $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$. This involves $m-1$ multiplications and m exponentiations in $\mathbb{G}_T$.

Moreover, algorithm Compute consists of two operations: (i) the matrix multiplication $\vec{y} = M\vec{x}$ which requires nm multiplications and additions in $\mathbb{F}_p$; and (ii) the generation of the proof Π which involves nm exponentiations and $(n-1)(m-1)$ multiplications in $\mathbb{G}_1$.

Finally, algorithm Verify evaluates two bilinear pairings, $(n-1)$ multiplications and n exponentiations in $\mathbb{G}_1$, and one multiplication in $\mathbb{G}_T$.

As for storage, the server is required to keep the (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ and the (n, m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} \in \mathbb{G}_1$. On the other hand, the client is required to store and publish the public parameters which are of size $\mathcal{O}(n)$ and the public key PK_M whose size is $\mathcal{O}(m)$. We highlight the fact that the public parameters' size can be made constant: Instead of advertising the set $\{g_i\}_{1\leqslant i\leqslant n}$, the client can select a hash function $\mathcal{H} : \mathbb{F}_p^* \to \mathbb{G}_1 \backslash \{1\}$ and compute the generators g_i as $\mathcal{H}(i)$, for all $1 \leqslant i \leqslant n$. On the downside, this optimization makes our scheme secure only in the random oracle model.

Table 2 summarizes the performance analysis of our scheme for publicly verifiable matrix multiplication.

5. RELATED WORK

Verifiable Polynomial Evaluation. Benabbas et al. [6] were the first to use algebraic PRFs for the problem of verifiable polynomial evaluation. Their solution only works in the symmetric-key setting, thus does not enable public verifiability as the schemes presented in this paper. In the same line of work, Fiore and Gennaro [11] devise new algebraic PRFs, also used by Zhang and Safavi-Naini [20], to develop publicly verifiable solutions. Compared to these two solutions, our protocol induces the same amount of computational costs but with the additional property of public delegatability. Another solution for public verification considers signatures for

	Setup	ProbGen	Compute	Verify	Hardness Assumptions	Public Delegatability
Fiore and Gennaro [11]	1 pairing $2(d+1)$ exp in $\mathbb{G}_1$ 1 exp in $\mathbb{G}_T$	1 pairing 1 exp in $\mathbb{G}_1$	$(d+1)$ exp in $\mathbb{G}_1$	1 pairing 1 exp in $\mathbb{G}_T$	co-CDH DLin	No
Papamanthou et al. [17]	*Polynomial preparation* $2d+1$ exp in $\mathbb{G}_1$		$d+1$ exp in $\mathbb{G}_1$	2 pairing 2 exp in $\mathbb{G}_1$	d-SBDH	Yes
Our scheme	$d+1$ exp in $\mathbb{G}_2$ 1 exp in $\mathbb{G}_1$	1 exp in $\mathbb{G}_1$ 1 exp in $\mathbb{G}_2$	$d-1$ exp in $\mathbb{G}_2$	2 pairings 1 exp in $\mathbb{G}_2$	$\lfloor d/2 \rfloor$-SDH	Yes

Table 3: Comparison of computation complexity with existing work for polynomial evaluation

	Setup	ProbGen	Compute	Verify	Hardness Assumptions	Public Delegatability
Fiore and Gennaro [11]	$3nm$ exp in $\mathbb{G}_1$	n pairings $2(n+m)$ exp in $\mathbb{G}_1$	nm exp in $\mathbb{G}_1$	n pairings n exp in $\mathbb{G}_T$	co-CDH DLin	No
Zhang and Blanton [21]	1 pairing	n exp in $\mathbb{G}_1$ m exp in $\mathbb{G}_2$ $(n+1)$ exp in $\mathbb{G}_T$	nm exp in $\mathbb{G}_2$	n pairings $(n+1)$ exp in $\mathbb{G}_T$	M-DDH XDH	Yes
Our scheme	$2nm$ exp in $\mathbb{G}_1$ m pairings	m exp in $\mathbb{G}_T$	nm exp in $\mathbb{G}_1$	2 pairings n exp in $\mathbb{G}_1$	co-CDH	Yes

Table 4: Comparison of computation complexity with existing work for matrix multiplication

correct computation [17], and uses polynomial commitments [16] to construct these signatures. Besides public verifiability, this solution implements public delegatability. However, the construction by [17] relies on the d-SBDH assumption, whereas our solution is secure under a weaker assumption that is the $\lfloor d/2 \rfloor$-SDH. It is worth mentioning that our protocol can be changed to rely on the $\lfloor d/\delta \rfloor$-SDH assumption, where δ is the degree of the divisor polynomial, as specified in Remark 1 of Section 3.3, and therefore our scheme can accommodate higher-degree polynomials.

Table[3] 3 compares the computational costs our solution for polynomial evaluation with the work described by Fiore and Gennaro [11] and Papamanthou et al. [17].

Verifiable Matrix Multiplication.

Fiore and Gennaro [11] and Zhang and Safavi-Naini [20] exploit algebraic PRFs for publicly verifiable matrix multiplications. However, only the client which outsourced the matrix can submit input vectors to the outsourced multiplication, hence their constructions do not meet the public delegatability requirement. Zhang and Blanton [21] present a construction for publicly delegatable and verifiable outsourcing of matrix multiplication that uses mathematical properties of matrices instead of algebraic PRFs. Unlike our work, the public verifiable scheme suggested in [21] does not transfer the matrix M to the server during Setup (whose purpose is reduced to generating the public parameters). Instead, the problem generation phase prepares the matrix and the input vector for the delegation. This construction is secure under the multiple decisional Diffie Hellman (M-DDH) and the eXternal Diffie-Hellman (XDH) assumptions, which are stronger than the co-CDH assumption we rely on in our solution.

Table[3] 4 depicts a comparison of our proposal for matrix multiplication with the solution proposed by Fiore and Gennaro [11] and Zhang and Blanton [21].

Arbitrary functions. A significant collection of work applies succinct non-interactive arguments of knowledge (SNARKs) to the problem of verifiable computation of arbitrary functions [4, 5, 7, 19]. One of the most relevant applications of the SNARK approach [7] appears in Pinocchio [19]. Pinocchio translates the outsourced function into an arithmetic circuit, which is then converted into a Quadratic Arithmetic Program [14]. As such, it enables public delegatability and public verifiability. However, the security of Pinocchio and other SNARK-based protocols relies on non-falsifiable assumptions as proved by Gentry and Wichs [15], whereas the security of our schemes only relies on falsifiable assumptions.

Parno et al. [18] propose a solution for public delegation and verification of computation using Attribute-Based Encryption (ABE). However, this scheme is limited to the computation of Boolean functions that output a single bit. For functions with more than one output bit, the client has to repeatedly (for each output bit) launch several instances of the protocol. Alderman et al. [1, 2] also propose an ABE-based protocol for Boolean functions. The authors adopt a scenario orthogonal to ours, in which queriers are identified according to access control policies. Furthermore, Alderman et al. [1, 2] introduce the concept of *blind verifiability*. In a nutshell, their protocols distinguish queriers from verifiers: The latter may only be authorized to verify an outsourced computation but not to learn its results. In the present paper, the blind verifiability property is out of scope.

Homomorphic MACs and signatures. Another type of solutions use homomorphic MACs [3, 9, 12] or homomorphic signatures [8, 10]. These solutions generally induce a verification as costly as the computation of the outsourced function itself. Homomorphic MACs proposed by Backes et al. [3] take advantage of algebraic PRFs to allow efficient verification, provided that the data is indexed. This solution however is suitable for quadratic functions only. Similarly, Catalano et al. [10] propose homomorphic signatures for polynomial functions with efficient verification and suggest that they can be used for a publicly verifiable computation scheme. Nevertheless, their construction uses expensive multilinear pairings.

6. CONCLUSION

In this paper, we introduced two protocols for publicly verifi-

[3]Table 3 and Table 4 compare the computational complexity of our solution with existing work only in terms of exponentiations and bilinear pairings which are the most computationally expensive operations.

able delegation of computation which enable a client to securely outsource the evaluation of arbitrary degree univariate polynomials and the multiplication of large matrices. We built our protocols upon the *algebraic* properties of polynomials and matrices. This paved the way for practical solutions that are provably secure against adaptive adversaries under the co-CDH and the SDH assumptions.

7. ACKNOWLEDGEMENTS

This work was partially supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

References

[1] James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. Revocation in Publicly Verifiable Outsourced Computation. In *Information Security and Cryptology*, pages 51–71. Springer, 2014.

[2] James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. Access Control in Publicly Verifiable Outsourced Computation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS*, volume 15, pages 657–662, 2015.

[3] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 863–874. ACM, 2013.

[4] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[5] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *USENIX Security*, pages 781–796, 2014.

[6] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable Delegation of Computation over Large Datasets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer Berlin Heidelberg, 2011.

[7] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[8] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Advances in Cryptology–EUROCRYPT 2011*, pages 149–168. Springer, 2011.

[9] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *EUROCRYPT*, pages 336–352. Springer, 2013.

[10] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology–CRYPTO 2014*, pages 371–389. Springer, 2014.

[11] Dario Fiore and Rosario Gennaro. Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 501–512. ACM, 2012.

[12] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *Advances in Cryptology-ASIACRYPT 2013*, pages 301–320. Springer, 2013.

[13] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[14] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT*, volume 7881, pages 626–645. Springer, 2013.

[15] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM, 2011.

[16] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010*, pages 177–194. Springer, 2010.

[17] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography*, pages 222–242. Springer, 2013.

[18] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer Berlin Heidelberg, 2012.

[19] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[20] Liang Feng Zhang and Reihaneh Safavi-Naini. Verifiable delegation of computations with storage-verification trade-off. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, volume 8712 of *Lecture Notes in Computer Science*, pages 112–129. Springer International Publishing, 2014.

[21] Yihua Zhang and Marina Blanton. Efficient secure and verifiable outsourcing of matrix multiplications. Cryptology ePrint Archive, Report 2014/133, 2014.