Privacy-Preserving Cloud-Assisted Services

Jian Liu

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 117/2018

Privacy-Preserving Cloud-Assisted Services

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T2 of the school on 18 June 2018 at 12:15.

Aalto University
School of Science
Secure Systems

Supervising professor
Professor N. Asokan, Aalto University, Finland

Preliminary examiners
Professor Srdjan Capkun, ETH Zurich, Switzerland
Professor Stefan Katzenbeisser, Technical University of Darmstadt, Germany

Opponent
Professor Moti Yung, Columbia University and Snapchat, US

Aalto University publication series
DOCTORAL DISSERTATIONS 117/2018

ISBN 978-952-60-8043-7 (printed)
ISBN 978-952-60-8044-4 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-8044-4

Unigrafia Oy
Helsinki 2018

Finland

441 697
Printed matter

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author

Name of the doctoral dissertation
Privacy-Preserving Cloud-Assisted Services

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 117/2018

Field of research Applied cryptography

Manuscript submitted 19 March 2018 **Date of the defence** 18 June 2018

Permission to publish granted (date) 2 May 2018 **Language** English

☒ **Article dissertation** ☐ **Monograph** ☐ **Essay dissertation**

Abstract

In the last decade, there has been a move towards making traditional IT services follow a cloud-assisted services paradigm. This has triggered previously local services to be moved to a cloud-assisted setting to reap the advantages of the cloud-assisted paradigm that can work with simple client-side functionality ("thin clients"). Examples of such services are cloud storage, cloud-assisted malware checking and "machine learning as a service" (MLaas).

Despite their benefits, these kinds of services put users' privacy at risk since the data stored in the cloud and/or the requests submitted to the cloud may contain sensitive information. On the other hand, unless carefully designed, this service paradigm may nonetheless fail to protect the confidentiality of service providers' business assets (e.g., malware databases or machine learning models) against malicious users.

This dissertation shows how to leverage cryptographic technologies and trusted execution environments to design cloud-assisted services such that end users can protect their privacy, and if needed, service providers can ensure that their security/privacy requirements are not violated. We provide a general definition for privacy-preserving cloud-assisted services, investigate the privacy issues in three cloud-assisted services: lookup service, prediction service and storage service, and propose solutions on how to make them privacy-preserving.

Keywords Private set intersection, TEEs, Machine learning, Neural networks, Secure two-party computation, Cloud Storage, Deduplication

ISBN (printed) 978-952-60-8043-7 **ISBN (pdf)** 978-952-60-8044-4

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2018

Pages 150 **urn** http://urn.fi/URN:ISBN:978-952-60-8044-4

Preface

This dissertation is a summary of my doctoral studies and work at the Department of Computer Science in Aalto University.

A great many people have helped me become the researcher that I am. First of all, I would like to express my sincere thanks to my supervisor Professor N. Asokan for his excellent guidance and always setting an example as a researcher for me. It is my privilege to have you as my supervisor. I would like to extend my gratitude to Professor Benny Pinkas, who guides me to the world of cryptography.

I would like to thank all my co-authors that have contributed to this work. Special thanks to Sandeep Tamrakar, Dr. Andrew Paverd and Mika Juuti for all the insightful discussion during my research. Thanks to my colleagues Thomas Nyman, Arseny Kurnikov, Thanh Bui, Siddharth Rao, Dr. Lachlan James Gunn, Dr. Samuel Marchal, Dr. Niina Idänheimo, and all other people of Secure Systems group, with whom I share an enjoyable working experience. Additionally, I want to extend my thanks to Ágnes Kiss, Wenting Li, Yao Lu, Li Duan, Dr. Yong Li, Dr. Jan-Erik Ekberg, Dr. Ghassan Karame and Professor Dr. Thomas Schneider for the wonderful collaborations during my doctoral studies.

Many thanks to my pre-examiners Professor Dr. Srdjan Capkun and Professor Dr. Stefan Katzenbeisser for their thorough reviews and insightful feedback. Additionally, I would like to thank Professor Moti Yung for the great honor of having him as the opponent for my dissertation.

This work was funded by TEKES project Cloud-Assisted Security Services (CloSer) and Academy of Finland project Cloud Security Services (CloSe).

Finally, I want to thank my family for their everlasting encouragement, blessings. Especially, I want to express my sincere gratitude to my wife Xi Chen for her unconditional support and endless love.

Espoo, May 26, 2018,

Jian Liu

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Àgnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, Benny Pinkas. Private Set Intersection for Unequal Set Sizes with Mobile Applications. In *Proceedings of the 17th Privacy Enhancing Technologies Symposium (PETS)*, Minneapolis, USA , Pages 97-117, July 2017.

II Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, N. Asokan. The Circle Game: Scalable Private Membership Test Using Trusted Hardware. In *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, Abu Dhabi, United Arab Emirates, Pages 31-44, April 2017.

III Jian Liu, Mika Juuti, Yao Lu, N Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Dallas, Texas, USA, Pages 619-631, October 2017.

IV Jian Liu, N. Asokan, Benny Pinkas. Secure Deduplication of Encrypted Data without Additional Independent Servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Denver, Colorado, USA, Pages 874-885, October 2015.

V Jian Liu, Li Duan, Yong Li, N. Asokan. Secure Deduplication of Encrypted Data: Refined Model and New Constructions. In *Proceedings of Topics in Cryptology – CT-RSA 2018*, San Francisco, California, USA, Pages 374-393, April 2018.

Author's Contribution

Publication I: "Private Set Intersection for Unequal Set Sizes with Mobile Applications"

Jian Liu contributed to the design, analysis and implementation of this paper. Jian Liu initially proposed the offline/online pattern for privacy-preserving malware checking, and finalised the design, analysis and implementation together with Àgnes Kiss under the guidance of Thomas Schneider, Benny Pinkas and N. Asokan. All authors wrote the manuscript together.

Publication II: "The Circle Game: Scalable Private Membership Test Using Trusted Hardware"

Jian Liu contributed to the design, analysis and implementation of this scheme. Jian Liu implemented the "sequence of differences" representation of the dictionary. All authors wrote the manuscript together.

Publication III: "Oblivious Neural Network Predictions via MiniONN Transformations"

Jian Liu contributed to the design, proof and implementation of this scheme. Jian Liu initially proposed and finalised the idea of oblivious neural network transformations. Mika Juuti helped with the approximation of smooth functions. Mika Juuti and Yao Lu jointly trained the models. Jian Liu did the implementation and performance analysis under the guidance of N. Asokan. All authors wrote the manuscript together.

Publication IV: "Secure Deduplication of Encrypted Data without Additional Independent Servers"

Jian Liu contributed to the design, proof, simulation and implementation of this scheme. Jian Liu initially proposed the idea of single-server secure deduplication of encrypted data (SDoE), and finalised the design together with my co-authors Profs. Benny Pinkas and N. Asokan. Jian Liu proved its security with the help of Benny Pinkas, and did the simulation and performance evaluation under the guidance of N. Asokan. Jian Liu wrote the manuscript together with my co-authors.

Publication V: "Secure Deduplication of Encrypted Data: Refined Model and New Constructions"

Jian Liu contributed to the design and implementation of this paper. Jian Liu proposed a security model for secure deduplication of encrypted data (SDoE), and finalised the model with Li Duan and Yong Li. Jian Liu designed the two SDoE schemes and did the simulations under the guidance of N. Asokan. All authors wrote the manuscript together.

1. Introduction

When Internet expanded into the public realm in the early 1990s, the client-server paradigm experienced a resurgence. All kinds of services such as e-commerce (e.g., Amazon), knowledge (e.g., Wikipedia), etc. appeared allowing users to access them using a general-purpose client like a browser. The advent of *cloud* has moved more services to a cloud-assisted setting which allows simple client-side functionality ("thin clients") and allows service providers to keep their business assets confidential on server-side. Examples of such services include storage, malware checking and machine learning prediction services.

This service paradigm puts users' privacy at risk since the data stored in the cloud and/or the requests submitted to the cloud may contain sensitive information. On the other hand, unless carefully designed, it may fail to protect the confidentiality of service providers' business assets as it promised. This dissertation shows how to leverage cryptographic technologies and trusted execution environments to design privacy-preserving cloud-assisted services such that end users can protect their privacy, and if needed, service providers can ensure that their security and privacy requirements are not violated.

1.1 Motivation

1.1.1 Trends in cloud-assisted services

The benefits of cloud-assisted services are clear: service users can scale their service consumption up or down as they need, and access the services any time from any device; it also becomes easier for service providers to maintain their services: keeping the services available, reliable and up-to-date. Here are some examples.

Cloud storage. Cloud storage is a type of cloud-assisted service where service providers offer physical storage and a set of APIs, so that cloud storage users can store their digital data on a remote server. Cloud storage providers ensure the availability of their users' data, i.e., keep the physical storage protected

and running. Examples of such cloud storage are Microsoft OneDrive [84], Dropbox [41], Google Drive [40] and Amazon Drive [5]. Storing data in the cloud has many advantages: data can be accessed and synchronised from any device that is connected to the Internet; multiple users can collaboratively update the same file; and the cloud storage provider can provide backup for data so that they will never disappear even if users' local devices fail.

Malware checking. Traditional malware checking services require users to install a client-side tool which downloads malware information from the service provider periodically and performs the checking locally. Nowadays, malware checking is gradually migrating to a cloud-assisted paradigm: client-side tools are thin and the bulk of the malware information are held by a cloud server; clients consult the cloud server to check if a particular application is malware or not. This paradigm not only helps malware checking service providers to reduce the amount of data transfers, but also avoids disclosing the malware databases in full to all customers and hence to competitors. Google's *Verify Apps* is such a cloud-assisted malware checking service [56].

Machine Learning as a Service (MLaaS). Cloud-assisted machine learning platforms can perform most of the machine learning tasks such as data pre-processing, model training, and prediction. For example, IBM provides various machine learning APIs (e.g., weather prediction, language analysis, image recognition, machine translation, sentiment and tone analysis etc.) on its Bluemix cloud platform [66]. Microsoft's Azure Machine Learning Studio allows users to bring their own data, train a model on it, and make the resulting model available to others as an API [85]. Google's cloud platform currently provides two kinds of services: Google Translate (translation using state-of-the-art neural machine translation) [58], and Google Prediction API (similar to Microsoft Azure Machine Learning Studio) [57]. MLaaS has various of benefits. Take cloud-based prediction as an example: compared with having users download the model and run the predictions on client-side, cloud-assisted prediction makes it easier for service providers to update their models and can also protect the model itself which is usually considered as a competitive advantage of the service providers.

1.1.2 User privacy in cloud-assisted services

As we discussed, companies are increasingly turning to cloud-assisted services to overcome the challenges incurred by the traditional infrastructure. However, cloud-assisted services pose their own challenges since personal data needs to be stored, transferred through and processed by the cloud server. Service users are usually unaware of how, when, where and why their data is used. Attackers can break into the cloud servers by exploiting software vulnerabilities so that they can snoop on or steal users' private data [36]. Curious or malicious administrators in the service hosting company can do the same thing without significant technical barriers [48]. The service provider may perform data mining techniques to analyse users' data for their own benefit. Considering

the aforementioned examples, a cloud storage provider can easily peek into the files uploaded by their users. In cloud-assisted malware checking services, users have to reveal their installed applications, which allows inference of their personal information such as gender, age, religion, and relationship status [100]. In cloud-assisted prediction services, users may have to reveal more specific and important information such as health and financial status.

Therefore, we cannot rely on the cloud provider to guarantee user privacy. Meanwhile, faced with the emergence of privacy regulatory regimes like EU General Data Protection Regulation (GDPR), providers of cloud-assisted services are themselves incentivized to find ways of providing their services without seeing sensitive user data.

1.1.3 Service provider requirements in cloud-assisted services

Straightforward solutions to user privacy concerns may conflict with other requirements. Taking the cloud-assisted prediction service as an example: the trained model is usually considered as a "secret sauce" by the service providers. If it is revealed to the public, competitors may use it to provide an equivalent service. Furthermore, the model may contain information of the training data [45]. Thus, revealing the model may violate regulations like the Health Insurance Portability and Accountability Act of 1996 (HIPAA). The same issue also exists in cloud malware checking service. For cloud storage, a straightforward solution to protect user privacy is having users encrypt their files on client-side before uploading. However, storage providers may want to conserve their storage space via *cross-user deduplication*: if two users upload the same file, the server stores only a single copy. Client-side encryption makes it difficult.

1.1.4 Privacy-preserving cloud-assisted services

The risks mentioned above motivate the development of *privacy-preserving cloud-assisted services*, which provide cloud-assisted services as usual without violating users' privacy, while still meeting the service providers' security and privacy requirements.

For simplicity, we treat the server that hosts a cloud-assisted service as a highly resourced, monolithic entity $\mathcal{S}$. We denote each entity using $\mathcal{S}$'s service as a client. We denote the set of n clients of $\mathcal{S}$ by $\{\mathcal{C}_1,\mathcal{C}_2,...,\mathcal{C}_n\}$. Suppose that each $\mathcal{C}_i$ has an input x_i and $\mathcal{S}$ has input x_0. A privacy-preserving cloud-assisted service allows $\mathcal{S}$ and $\{\mathcal{C}_1,\mathcal{C}_2,...,\mathcal{C}_n\}$ to *privately* evaluate a function f:

$$(y_0,y_1,...,y_n) = f(x_0,x_1,...,x_n) \tag{1.1}$$

where $\mathcal{S}$ obtains only y_0, and each $\mathcal{C}_i$ only obtains their respective y_i.

Here private evaluation means that each $\mathcal{C}_i$ learns no information about other input x_j for $j \neq i$, except the information revealed by y_i itself. Similarly, $\mathcal{S}$ learns nothing except y_0. Stated in the ideal/real world paradigm [54]: there is an

inherently trusted entity that receives inputs $\{x_0, x_1, ..., x_n\}$ from all parties, computes $f()$ and returns result $\{y_0, y_1, ..., y_n\}$ without leaking any other information to any parties (ideal world). We require that a real world solution does not disclose more information than its ideal world version.

Let us use *outsourced computation* as an example to explain this. Suppose $\mathcal{C}_i$ wants to use $\mathcal{S}$'s computing power to compute a function f on its input x_i, but want to avoid revealing x_i to $\mathcal{S}$. In this case, $x_j = y_j = \perp$ for all $j \neq i$.

Let us take a *nearby people service* [52, 60] as another example. Users who use this service want to know their neighbouring users, but do not want to reveal their own locations to the service. In this case, x_i is $\mathcal{C}_i$'s location data and y_i is a list of users $\{\mathcal{C}_j\}$ that satisfies $D(x_i, x_j) < t$, where $D()$ is a function that calculates the distance between x_i and x_j, and t is the distance threshold.

The aforementioned services (e.g., cloud-assisted malware checking, cloud-assisted prediction and cloud storage) can also be formalised in this form, as introduced in the chapters 3, 4 and 5 respectively.

1.2 Contributions

The aims and contributions of my dissertation is to design three privacy-preserving cloud-assisted services:

- A cloud-assisted malware checking service that protects both server's database and clients' lookup queries (Service 1);
- A cloud-assisted prediction service that protects both server's model and clients' input (Service 2);
- A cloud storage service that supports client-side encryption and cross-user deduplication (Service 3).

My dissertation consists of this compendium together with the five publications included at the end. Publication I and II build Service 1 via two different approaches: private set intersection with pre-computation and trusted execution environments (TEEs). Publication III builds Service 2 via secure two-party computation combined with additively homomorphic encryption. Publication IV builds Service 3 via password authenticated key exchange, and V provides a formal security model and enhanced construction for Service 3. The contributions of the individual publications are summarized below:

Publication I: "Private Set Intersection for Unequal Set Sizes with Mobile Applications"

This publication models the cloud-assisted malware checking scenario as a private membership test and proposed a solution by exploiting the offline/online characteristics of this scenario [83]. The main aim is to shift most of the com-

munication and computation to an offline phase and have a very efficient online phase. Four existing protocols are investigated and adapted to the offline/online pattern.

Publication II: "The Circle Game: Scalable Private Membership Test Using Trusted Hardware"

This publication presents another solution for privacy-preserving cloud-assisted malware checking. It introduces a simple private membership test approach using a carousel design pattern where the entire malware dictionary is cycled through a trusted execution environment (TEE) on the cloud server. It evaluated the use of different data structures to represent the dictionary, and implements a prototype on two different TEE architectures, ARM TrustZone and Intel SGX. Furthermore, through extensive experimental analysis, it shows that the carousel approach exhibits better scalability than oblivious RAM while supporting a higher number of simultaneous queries. This publication has received an Honorable Mention in ASIACCS 2017 and it has been selected as one of the top 10 publications from Europe in the Applied Research Contest at CSAW Europe. Signal's private contact discovery scheme (announced after the publication of Publication II but designed independently) followed the same design pattern [103].

Publication III: "Oblivious Neural Network Predictions via MiniONN Transformations"

This publication presents the first approach for transforming any existing neural network to an oblivious neural network supporting privacy-preserving predictions with reasonable efficiency. Unlike prior work, it requires no change to how models are trained. To this end, we designed oblivious protocols for commonly used operations in neural network prediction models. This publication shows that our technique called MiniONN outperforms existing work in terms of response latency and message sizes. To demonstrate the wide applicability of this technique, several typical neural network models trained from standard datasets are transformed.

Publication IV: "Secure Deduplication of Encrypted Data without Additional Independent Servers"

This publication presents the first single-server scheme for secure cross-user deduplication with client-side encrypted data. This scheme allows a client uploading an existing file to securely obtain the encryption key that was used by the client who has previously uploaded that file. The scheme builds upon a well-known cryptographic primitive known as password authenticated key exchange (PAKE), and provides better security guarantees than previous work:

the first scheme that can prevent online brute-force attacks by a compromised server. The effectiveness and the efficiency of this scheme were demonstrated via simulations using realistic datasets and a prototype implementation.

Publication V: "Secure Deduplication of Encrypted Data: Refined Model and New Constructions"

This publication provides a formal security model for the single-server secure deduplication of encrypted data (SDoE). Any deduplication scheme proved secure in this model can guarantee that, for a certain file, (1) a compromised client cannot learn whether or not this file has already been uploaded by someone else, and (2) the only way for a compromised server to uniquely identify the content of this file is by doing an online brute-force attack. This publication also gives two new single-server deduplication schemes that are proved secure in this model. The effectiveness of each scheme is evaluated via simulations with realistic datasets.

1.3 Outline

This dissertation is based on the aforementioned five original publications, which are grouped into three main themes.

Chapter 2 presents the relevant background for the whole dissertation.

Chapter 3 encompasses Publication I and Publication II. It presents a privacy-preserving cloud-assisted malware checking service. Two solutions, one based on cryptography and the other based on hardware-assisted TEEs, are described in this chapter.

Chapter 4 encompasses Publication III. It presents a privacy-preserving cloud-assisted machine learning prediction service. This chapter describes the first technique that can transform any common neural network model into an oblivious network without any modifications to the training phase.

Chapter 5 encompasses Publication IV and Publication V. It presents a cloud storage service that supports deduplication of encrypted data. The chapter describes the first single-server scheme for secure cross-user deduplication with client-side encrypted data.

Chapter 6 provides a discussion on the precautions and difficulties of building privacy-preserving cloud-assisted services. It also provides a summary of this dissertation and some directions on future work.

2. Background

This chapter presents relevant background information for the rest of the dissertation.

2.1 Homomorphic encryption

Homomorphic encryption is a commonly used primitive in privacy-preserving computations, which allows operations on plaintexts to be performed directly on ciphertexts. A public key encryption scheme is *additively homomorphic* if given two ciphertexts $\hat{x}_1 := E(pk, x_1)$ and $\hat{x}_2 := E(pk, x_2)$, there is an operation $\oplus$ such that $E(pk, x_1 + x_2) \leftarrow \hat{x}_1 \oplus \hat{x}_2$. Examples of such schemes are Paillier's encryption [89], and exponential ElGamal encryption [42].

As an inverse of addition, subtraction $\ominus$ is trivially supported by additively homomorphic encryption. Furthermore, adding or multiplying a ciphertext by a constant is efficiently supported: $E(pk, a + x) \leftarrow a \oplus \hat{x}$ and $E(pk, a \cdot x_1) \leftarrow a \otimes \hat{x}_1$.

To do both addition and multiplication between two ciphertexts, fully homomorphic encryption (FHE) [49] or leveled homomorphic encryption (LHE) [23] is needed. However, current FHE cryptosystems require expensive bootstrapping operations and LHE only supports a limited number of homomorphic operations.

The ciphertext of a (homomorphic) encryption scheme is usually much larger than the data being encrypted, and the homomorphic operations on the ciphertexts take longer than those on the plaintexts. One way to alleviate this issue is to encode several messages into a single plaintext and use the *single instruction multiple data* (SIMD) [104] technique to process these encrypted messages in batches without introducing any extra cost. The LHE library [39] has implemented SIMD-based on the Chinese Reminder Theorem (CRT). We use $\tilde{\mathbf{x}}$ to denote the encryption of a vector $[x_1, ..., x_n]$ in batch using the SIMD technique.

2.2 Secure two-party computation

Secure two-party computation (2PC) is a type of protocol that allows two parties to jointly compute a function $(f_1(x,y), f_2(x,y)) = f(x,y)$ without learning each other's input. It offers the same security guarantee achieved by a trusted third party TTP computing f: both parties submit their inputs (i.e., x and y) to TTP, who computes and returns the corresponding output to each party, so that no information has been leaked except the information that can be inferred from the outputs. There are three main techniques to achieve 2PC: *arithmetic secret sharing* [15], *boolean secret sharing* [53] and Yao's *garbled circuits* [113, 112]. The ABY framework [37] is a state-of-the-art 2PC library that implements all three techniques.

Yao's garbled circuits (GC) is the most commonly used 2PC protocol. In this protocol, one party called *garbler* garbles the circuit by assigning symmetric keys to the input wires and encrypting the output wires with the keys on the input wires. Then the garbler sends the garbled circuit together with its garbled inputs (keys) to the *evaluator*. Next, the garbler and the evaluator run a 1-out-of-2 *oblivious transfer* (OT), which enables a message exchange in an oblivious way: the garbler sends two messages k_0 and k_1 and the evaluator only receives k_b depends on its input b. After OT, the evaluator receives the keys corresponding to its input bits. Now, the evaluator can evaluate the garbled circuit by decrypting the wires one by one.

Yao's garbled circuit protocol is usually used together with the following optimizations: free XOR [73] that requires no communication or cryptographic operations for evaluating XOR gates; fixed-key AES garbling [16] that requires key scheduling only once for all AES encryptions; half-gates [114] that reduces the complexity per AND gate to two encryptions (instead of four); OT Extension [68, 10, 11, 71] that pre-computes only a small number of so-called *base OTs*, from which any polynomial number of OTs can be computed using only efficient symmetric-key operations.

The SIMD technique can also be applied to secure 2PC to reduce the memory footprint of the circuit and improve the circuit evaluation time [21]. In traditional garbled circuits, each wire stores a single input, while in the SIMD version, an input is split across multiple wires so that each wire corresponds to multiple inputs. The ABY framework [37] supports this.

2.3 Password authenticated key exchange (PAKE)

Bellovin and Merritt [19] were the first to propose a *password authenticated key exchange* (PAKE) protocol for user authentication, in which an adversary making a password guess cannot verify the guess without an online attempt to authenticate itself with that password. PAKE allows two players to agree on a session key if and only if they share a short secret ("password"). If the passwords

are different then neither party can learn anything about the key output to the other party (namely, cannot distinguish that key from a random key).

Bellare et al. gave a formal game-based definition for the security of PAKE [18]. They assume that there is an adversary $\mathcal{A}$ that has complete control over the environment (mainly, the network), and thus provides the inputs to both players. Formally, at the beginning of the game, a random bit b is chosen. The permitted queries are formally defined in [18] and summarized as follows:

$\mathsf{Send}(U_i, M)$: causes the message M to be sent to instance U_i, which does the computation as specified by the protocol, updates the state, and returns the output of the computation to $\mathcal{A}$. If this query causes U_i to accept or terminate, this information will also be shown to $\mathcal{A}$.

$\mathsf{Execute}(A_i, B_j)$: causes the protocol to be executed to completion between A_i and B_j, and outputs the transcript of the execution.

$\mathsf{Reveal}(U_i)$: causes the output of the session key held by U_i.

$\mathsf{Test}()$: if $b = 1$, causes the output of the session key; otherwise, a string drawn uniformly from the space of session keys.

$\mathsf{Corrupt}(U_i)$: causes the output of the password held by U_i.

Let $\mathsf{Succ}_{\mathcal{A}}^{\mathsf{PAKE}}(\lambda)$ be the event that $\mathcal{A}$ outputs a bit $b' = b$ but none of the following events happens:

1. a $\mathsf{Reveal}(U_i)$ query occurs;
2. a $\mathsf{Reveal}(U_j)$ query occurs where U_j is the partner of U_i;
3. a $\mathsf{Corrupt}(U_i)$ query occurs before U_i defined its key and a $\mathsf{Send}(U_i, M)$ query occurred.

The PAKE protocol is considered secure if the only way to break it is via an online brute-force attack. Concretely, this means that, if passwords are uniformly and independently drawn from a dictionary of size n:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PAKE}}(\lambda) \leq \frac{n_{se}}{n} + \mathsf{negl}(\lambda),$$

where n_{se} is the number of Send queries (to distinct instances U_i).

We use PAKE to build a cloud storage that supports dedeuplication on encrypted data. The security model for our deduplication protocol follows the security model of PAKE.

2.4 Trusted execution environments (TEEs)

A *Trusted Execution Environment* (TEE) [2] is a system security primitive that isolates and protects security-critical logic from all other software on the platform. All software outside the TEE is said to be running in the *Rich Execution*

Environment (REE), which usually includes the operating system and the majority of the platform's software. A piece of application logic running in the TEE is referred to as a *Trusted Application* (TA), whilst an application running in the REE is a *Client Application* (CA).

Fundamentally, a TEE protects the confidentiality and integrity of a TA's data. A TEE usually provides some form of *remote attestation*, which allows remote verifiers to ascertain its current configuration and behavior. In modern systems, the capability to establish and enforce a TEE is often provided by the CPU itself. This leads to very strong hardware-enforced security guarantees, and also improves performance by enabling the TEE to execute on the main CPU. The two most prevalent commercial TEEs are: ARM TrustZone and Intel SGX.

ARM TrustZone [1] is a TEE architecture that is widely deployed on smartphones and is now being deployed on infrastructure-class AMD CPUs. TrustZone provides a platform-wide TEE, called the *secure world*, which is fully isolated from the REE or *normal world*. All interaction between the REE and TEE is mediated by the CPU. The secure world usually runs a trusted OS, such as Kinibi from Trustonic [4]. Due to the constraints of the platform, the trusted OS may limit TEE's memory (e.g., Kinibi limits each TEE to 1 MB).

The recent *Software Guard Extensions* (SGX) technology [82] from Intel allows individual applications to establish their own TEEs, called *enclaves*. An enclave can contain application logic and secret data, protecting the confidentiality and integrity of them from all other software on the platform, including other enclaves, applications, or the (untrusted) OS. SGX includes remote attestation capabilities to provide remote parties with assurance about the code running in an enclave [6].

2.5 Neural Networks

Recently, a particular machine learning framework, *neural networks* (sometimes referred to as *deep learning*), has gained much popularity due to its record-breaking performance in many tasks such as image classification [75], speech recognition [35] and complex board games [69]. A neural network consists of a pipeline of layers. Each layer receives input and processes it to produce an output that serves as input to the next layer. Conventionally, layers are organized so that the bottom-most layer receives input data (e.g., an image or a word) and the top-most layer outputs the final predictions. A typical neural network[1] processes input data in groups of layers, by first applying *linear transformations*, followed by the application of a nonlinear *activation function*. Sometimes a *pooling operation* is included to aggregate groups of inputs.

The most common linear transformations in neural networks are matrix

[1] `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`

multiplications and additions:

$$\mathbf{y} := \mathbf{W} \cdot \mathbf{x} + \mathbf{b}, \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^{l \times 1}$ is the input vector, $\mathbf{y} \in \mathbb{R}^{n \times 1}$ is the output, $\mathbf{W} \in \mathbb{R}^{n \times l}$ is the *weight matrix* and $\mathbf{b} \in \mathbb{R}^{n \times 1}$ is the *bias vector*. *Convolution* is a type of linear transformation, which computes the dot product of small "weight tensors" (filters) and the neighborhood of an element in the input. The process is repeated, by sliding each filter by a certain amount in each step. The size of the neighborhood is called *window size*. The step size is called *stride*. In practice, for efficiency reasons, convolution is converted into matrix multiplication and addition as well [30], similar to equation 2.1, except that input and bias vector are matrices: $\mathbf{Y} := \mathbf{W} \cdot \mathbf{X} + \mathbf{B}$.

Neural networks use nonlinear transformations of data called *activation functions* to model nonlinear relationships between input data and output predictions. We identify three common categories:

- *Piecewise linear activation functions* This category of functions can be represented as a set of n linear functions within specific ranges, each of the type $f_i(y) = a_i y + b_i, y \in [y_i, y_{i+1}]$, where y_i and y_{i+1} are the lower and upper bounds for the range. This category includes the activation functions:

Identity function (linear): $f(\mathbf{y}) = [y_i]$

Rectified Linear Units (ReLU): $f(\mathbf{y}) = [\max(0, y_i)]$

Leaky ReLU: $f(\mathbf{y}) = [\max(0, y_i) + a \min(0, y_i)]$

Maxout (n pieces): $f(\mathbf{y}) = [\max(y_1, \ldots, y_n)]$

- *Smooth activation functions* A smooth function has continuous derivatives up to some desired order over some domain. Some commonly used smooth activation functions are:

Sigmoid (logistic): $f(\mathbf{y}) = \left[\frac{1}{1+e^{-y_i}}\right]$

Hyperbolic tangent (tanh): $f(\mathbf{y}) = \left[\frac{e^{2y_i}-1}{e^{2y_i}+1}\right]$

Softplus: $f(\mathbf{y}) = [\log(e^{y_i} + 1)]$

The sigmoid and tanh functions are closely related [55]:

$$\tanh(x) = 2 \cdot \text{sigmoid}(2x) - 1. \tag{2.2}$$

They are collectively referred to as *sigmoidal* functions.

- *Softmax* Softmax is defined as:

$$f(\mathbf{y}) = \left[\frac{e^{y_i}}{\sum_j e^{y_j}}\right]$$

It is usually applied to the last layer to compute a probability distribution in categorical classification so that its outputs all add up to one. However, in

prediction phase, usually it is sufficient to use argmax over the outputs of the last layer to predict the most likely outcome.

Neural networks also commonly use pooling operations that arrange input into several groups and aggregate inputs within each group. Pooling is commonly done by calculating the average or the maximum value among the inputs (*mean* or *max pooling*). Convolution and pooling operations are only used if the input data has spatial structure (e.g., images, sounds).

3. Cloud-Assisted Lookup Services

Cloud-assisted lookup services are being used in scenarios such as checking a smartphone application against a malware database, querying whether a password is present in a database of leaked passwords, or testing whether a DNA sequence is present in an online biological database. Privacy is a common concern in all these services, because the queries submitted to the server may contain sensitive personal information about the user. For example, knowledge of applications installed on a user's device can be used to infer personal characteristics of the user including age, gender, religion, and relationship status [100]. It is therefore in the interests of service providers to demonstrably preclude any ability to infer personal information about their users.

This goal of privately checking the existence of an item in a remote database is referred to as *private membership test* (PMT): a user wants to test whether an item is a member of a large set held by a remote server, without revealing the item and learning anything about the rest of the set. We propose two novel solutions. The first is to have the server put the blinded elements of the set into a compact data structure (e.g., bloom filter or cuckoo filter), and send it to clients. The client who wants to query an item runs an oblivious protocol to get its blinded version, and tests the membership locally. The second solution assumes a hardware-assisted TEE (e.g., SGX) exists on server. We first have the server put the set into a compact data structure, and then have a trusted application process clients' queries by cycling the database through the TEE, which performs the membership test.

3.1 Private membership test (PMT)

Mapping to the model introduced in Section 1.1.4, the input of the server $\mathcal{S}$ is a set of entries called a dictionary $X = \{x_1, x_2, ..., x_n\}$, and the input of a client $\mathcal{C}$ who wants to lookup the dictionary is an item x. Other clients' input and output

are $\perp$. A PMT protocol allows $\mathcal{S}$ and $\mathcal{C}$ to privately evaluate a function f:

$$f(X,x) = \begin{cases} (\perp,1) \text{ if } x \in X \\ (\perp,0) \text{ otherwise} \end{cases} \tag{3.1}$$

The natural cryptographic primitive to build PMT is *private set intersection* (PSI) [94, 92]. However, a PSI involving a dictionary of size n requires $\mathcal{O}(n)$ communication between $\mathcal{S}$ and $\mathcal{C}$ [94]. Another possible cryptographic candidate is *private information retrieval* (PIR), which allows $\mathcal{C}$ to retrieve x_i without revealing i. We can adapt PIR for PMT by inserting the dictionary into a data structure for membership test (e.g., bloom filter, cuckoo filter) and have $\mathcal{C}$ retrieve the corresponding entry privately. However, this solution requires $\mathcal{S}$ to perform $\mathcal{O}(n)$ expensive operations like modular exponentiations.

3.2 Adversary model

We assume that an adversary $\mathcal{A}$ can compromise either $\mathcal{S}$ or $\mathcal{C}$, but not both. If $\mathcal{A}$ compromises both $\mathcal{S}$ and $\mathcal{C}$, it can get whatever information it desires. The objective of a compromised $\mathcal{S}$ is to learn $\mathcal{C}$'s queries, which can be used to profile users. Similarly, the objective of a compromised $\mathcal{C}$ is to learn $\mathcal{S}$'s database. We assume a probabilistic polynomial-time (PPT) $\mathcal{A}$, i.e., it cannot break cryptographic primitives like state-of-the-art cryptographic hash functions or encryption algorithms.

In our second solution, we assume that $\mathcal{S}$ is equipped with a hardware-assisted TEE, and $\mathcal{A}$ has full control of the REE, but not the TEE. We also assume that there is a secure channel between TEE and $\mathcal{C}$ so that $\mathcal{A}$ cannot learn the messages exchanged via this channel. Furthermore, we assume that the size of the dictionary is too large to be fit into the secure memory of TEE. Therefore, the *access patterns* from TEE to the dictionary are visible to $\mathcal{A}$.

3.3 Research questions

We consider the scenario of cloud-assisted mobile malware checking with a malware dictionary of 2^{26} entries (~67 million identifiers) [72]. We first limit the entries to identifiers because they are prevalently used in current malware checking schemes. We will discuss the limitations of this assumption in the end of this chapter and give solutions. In this scenario, false negatives are never allowed, but a small false-positive rate (FPR) is acceptable. A FPR of 2^{-10} is recommended by a leading anti-malware vendor (F-Secure) [72]. We chose 2^{-10} in our ARM TrustZone setting and 2^{-14} in our SGX setting since it is more efficient to operate on byte-aligned data structures in SGX. Given that on average there are 95 apps [3] installed on each device, the majority of users will

rarely encounter a false positive even considering a few updates for each app. This chapter investigates the following research questions:

RQ1. Can we adapt previous cryptographic solutions to suit the above scenario?

RQ2. Can we design a TEE-based PMT protocol that provides better performance than other solutions in the above scenario?

3.4 PMT via private set intersection with pre-computation

3.4.1 Design overview

We propose a practical solution by approaching the problem from an *offline*/*online* setting. Specifically, we introduce the following three phases:

1. a *base phase* where $\mathcal{S}$ and $\mathcal{C}$ run data-independent precomputation, e.g., agree on the public parameters;
2. a *setup phase* where query-independent precomputation is performed on $\mathcal{S}$'s input and the results are transferred to $\mathcal{C}$s, who then potentially performs additional computations;
3. an *online phase* where the operations dependent on the $\mathcal{C}$s' queries are performed.

We aim to shift most of the communication and computation work into the base and setup phases to have an efficient online phase. Take our malware checking scenario as an example, after installing the client-side tool provided by the service provider, $\mathcal{C}$ may run the base and setup phases overnight, so that the online phase can be finished within a short period of time. Updates (i.e., insertion or deletion) on the dictionary are also supported by only running the efficient online phase.

In the setup phase, $\mathcal{S}$ blinds each element x_i in X, resulting in a blinded dictionary $\tilde{X} = \{\tilde{x}_1, ..., \tilde{x}_n\}$. Then $\mathcal{S}$ inserts $\tilde{X}$ into a compact data structure (e.g., bloom filter or cuckoo filter) and transfers it to $\mathcal{C}$. Whenever $\mathcal{C}$ wants to check x, it runs an oblivious protocol with $\mathcal{S}$ to get $\tilde{x}$ without revealing any information about x, after which it can check the existence of x locally. We investigate four techniques to achieve this goal: RSA based [34], Diffie-Hellman (DH) based [64], Naor-Reingold (NR) based [88, 62], and garbled circuit (GC-AES) based [93].

We take the GC-AES based solution with a bloom filter as an example (the details of all solutions are elaborated in Publication I). The base phase starts with $\mathcal{S}$ generating the secret key k for AES encryption. Then, m garbled tables are generated and sent to $\mathcal{C}$, where m is a parameter agreed by $\mathcal{S}$ and $\mathcal{C}$. After m queries, $\mathcal{S}$ and $\mathcal{C}$ need to run the setup phase again. Besides this, $l \cdot m$ OTs

are precomputed using OT extension, where l is the number of bits for each element x. Next, in the setup phase, $\mathcal{S}$ inserts $\tilde{X} = \{E(k,x_1),...,E(k,x_n)\}$ into a bloom filter BF and sends it to $\mathcal{C}$. In the online phase, for each bit of its inputs, $\mathcal{C}$ runs an oblivious transfer to retrieve its corresponding keys for the garbled circuit. Then, $\mathcal{C}$ evaluates the AES garbled circuit and checks if the result is in the bloom filter.

To insert a new element x' in, $\mathcal{S}$ can either send $E(k,x')$ to $\mathcal{C}$ or send the positions of the bloom filter that need to be changed to one. Since bloom filter does not support delete operation, we need to use a counting bloom filter [44]. Instead of storing bits, the counting bloom filter stores small counter values that are increased by one on insert and decreased by one on delete. To update the dictionary, $\mathcal{S}$ just needs to send the positions of the counters that need to be increased (for insert) or decreased (for delete).

3.4.2 Implementation and evaluation

We implemented all four solutions and systematically compared their performance. We ran the server-side application on a remote server and ran the client both on a PC and on an Android smartphone. To the best of our knowledge, this is the first comparison of PSI protocols on a smartphone.

The settings are depicted in Publication I, and the main results are shown in Table 3.1. Our results show that the GC-AES based solution achieves the best overall performance. Due to the fact that it only uses AES encryptions on the server's large database, its setup phase is orders of magnitude more efficient than that of any other protocol. In our PC implementation, its online phase is also the most efficient one. Our results on PC indicate that advancements on hardware-accelerated encryption on smartphones could greatly improve the performance of our proposed solution.

3.5 The carousel approach for PMT

3.5.1 Design overview

If a hardware-assisted TEE is assumed to exist on the server, a natural solution is to have $\mathcal{C}$ send queries to the TA via a secure channel, and have TA do the membership test. This naive approach leaks information about the queries from the dictionary positions that TA accessed. We can have the TA encrypt and shuffle the dictionary, and access it to answer each query. However, it still leaks information from the access patterns. For example, if $\mathcal{S}$ wants to know if a legitimate $\mathcal{C}$ is querying an item x, it can query x by generating a dummy $\mathcal{C}$, and see if both queries go to the same position in the dictionary. Several prior works [13, 65, 81] have shown how to solve this problem by combining

	Base (ms)	**Setup (ms)**		**Online (ms)**	**Update (ms)**
Task		Encryption	BF		
RSA	56	3 441 906	309	7,38	0,11
DH	**1**	462 496	**257**	3,49	0,11
ECC-DH	**1**	1 325 400	**257**	2,91	0,11
NR	119	758 400	309	10,82	0,11
GC-AES	1 312	**70**	309	**2,49**	**0,9**

(a) PC setting (Intel Core i7 with 3.5 GHz and 16 GB RAM)

	Base (ms)	**Setup (ms)**		**Online (ms)**	**Update (ms)**
Task		Encryption	BF		
RSA	5 492	19 892 745	**1 590**	60	8
DH	**1**	3 014 656	50 880	**23**	8
ECC-DH	**1**	167 837 696	50 880	363	8
NR	45 035	12 100 105	**1 590**	247	8
GC-AES	456 683	**1 851**	**1 590**	8 470	**4**

(b) Smartphone setting (Android 6.0.1 phone with Quad-core 2.5 GHz Krait 400)

Figure 3.1. Runtimes in milliseconds (taken from Publication I). The parameter choices are as follows: $n = 2^{20}$, $m = 2^{10}$, $l = 128$. Best values marked in bold.

TEE with *oblivious RAM* (ORAM), which shuffles the database for each query. Path ORAM [107], a recent ORAM scheme allows the TA to touch only $O(\log n)$ items to shuffle the database. Therefore, this solution has a constant communication overhead and only $O(\log n)$ computational overhead per query. However, supporting m simultaneous queries will incur $O(m \log n)$ cost.

We propose a new *carousel design pattern* by having the TA continuously cycle through the dictionary, such that a batch of queries can be answered within a single carousel cycle. The overview of our carousel approach is shown in Figure 3.2.

As described in Publication I, $\mathcal{S}$ inserts the dictionary X into a compact data structure $Y = \{y_1, ..., y_{n'}\}$. Since the TA needs to cycle the whole Y, the overhead simply depends on the size of Y. After experimentally evaluating the performance of several well-known data structures, we chose *4-ary cuckoo filter* for our protocol. We refer to the resulting private membership test solution as "Cuckoo-on-a-carousel". We refer to Table 1 of Publication II for detailed comparisons.

To avoid leaking information from timing channels, we enforce that a query must remain in the TEE for exactly one full carousel cycle. To achieve this, we divide Y into chunks and associate queries with the identifier of the chunk with which they arrived. The TA reads each chunk of Y sequentially and compares each entry in the chunk with the queries inside its memory and records the results. We also have TA perform constant-time processing for every entry in Y.

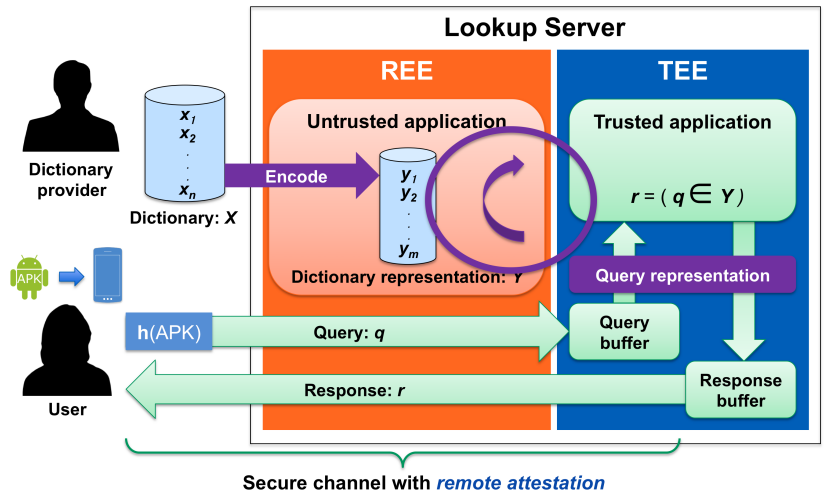


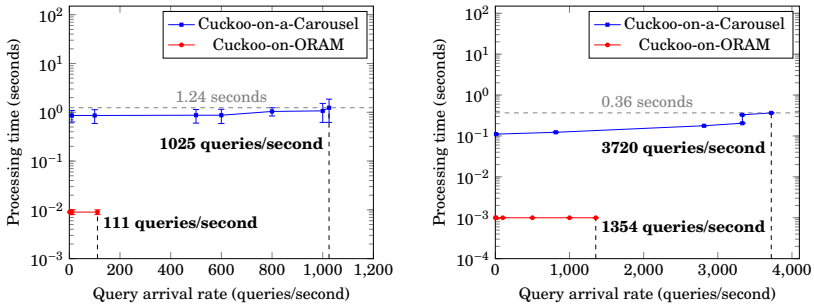
Figure 3.2. Overview of the carousel approach (taken from Publication II).

3.5.2 Implementation and evaluation

We implemented the full system on the two most commonly used TEE architectures: ARM TrustZone and Intel SGX. We generated a dictionary of $n = 2^{26}$ entries and inserted them into a 4-ary cuckoo filter. We also implemented the essential parts of Path ORAM [107] on both platforms. Since ORAM itself is not specifically designed for PMT, we first inserted the dictionary into a suitable data structure and then stored it in ORAM. We chose cuckoo filter here as well, since it requires the fewest memory accesses. We call this approach "Cukoo-on-ORAM". More implementation details and environment setup can be found in Section 7 of Publication II.

Figure 3 and Figure 4 in Publication II show the latencies for processing a single batch of queries on both platforms. Cuckoo-on-ORAM provides a small response latency (9 ms) for a single query, but the queries need to be processed sequentially. For example, it takes 18 seconds to process 2 000 queries on ARM TrustZone, which is beyond the acceptable tolerance of a malware checking service. In contrast, Cuckoo-on-a-carousel can process a batch of queries within on circle. As a result, Cuckoo-on-a-carousel on average takes only 1.83 seconds to process 2,000 queries on ARM TrustZone. Results for Intel SGX show a similar pattern, although with significantly lower latencies (e.g., Cuckoo-on-a-carousel on SGX takes 0.282 seconds to process 2,000 queries).

The advantage of Cuckoo-on-a-carousel over Cuckoo-on-ORAM is more prominent if we consider the steady-state performance. Figure 3.3a and Figure 3.3b show the steady state performance of Cuckoo-on-ORAM and Cuckoo-on-a-carousel for different query arrival rates. We identified the breakdown point where TA can no longer guarantee a bounded query response latency. For example in ARM



(a) ARM TrustZone: Samsung Exynos 5250 development board from Arndale with a 1.7 GHz dual-core ARM Cortex-A17 processor.

(b) Intel SGX: SGX-enabled HP EliteDesk 800 G2 desktop PC with a 3.2 GHz Intel Core i5 6500 CPU and 8 GB of RAM.

Figure 3.3. Steady-state processing time for uniform query arrival rates (taken from Publication II). Vertical lines indicate breakdown points.

TrustZone, we found that for query arrival rates above 1030 queries/second, the Cuckoo-on-a-carousel query response latency cannot be sustained. In contrast, with 1 025 queries/second, the response latency of Cuckoo-on-a-carousel are stable, and we therefore conclude that the breakdown point is between 1 025 and 1 030 queries/second. On the other hand, the breakdown point for Cuckoo-on-ORAM is much smaller, i.e., 111 queries/second. Results for Intel SGX show a similar pattern: 3 720 queries/second for Cuckoo-on-a-carousel and 1 354 queries/second for Cuckoo-on-ORAM.

3.6 Summary and discussion

Motivated by the scenario of privacy-preserving cloud-assisted malware checking, we introduce two new approaches for private membership test. In the first approach, we optimize the efficiency of private set intersection via precomputation so that it can efficiently deal with an unequal number of inputs. This approach is suitable for the scenarios where a large amounts of data is allowed to be transferred in a preprocessing phase.

In the second approach, we propose a new “carousel” design pattern by having the TA continuously cycle the dictionary, such that a batch of queries can be answered after exactly one carousel circle. This approach is applicable if there is a hardware-assisted TEE on server-side and the data set is not very large (limited by 2^{26} entries). For larger data sets, we can divide them into subsets and use a TA to run carousel for each subset. However, we need to carefully route the queries to the targeted subset so that they will not leak any information. For example, we can route all queries to all TAs and each TA only process the queries that are targeting its subset. We also need to consider the case of larger number of simultaneous queries which may make our scheme cross the breakdown point. In this case, we can also use multiple TAs but have each TA maintain a single

replicated dataset.

In addition to malware checking, our PMT schemes can also be applied to other scenarios such as those we introduced in the beginning of this chapter. In particular, Signal independently adopted a similar approach for their private contact discovery [103], so that Signal users can efficiently determine whether the contacts in their address book are Signal users without revealing the contacts in their address book to the service.

A malware developer can easily circumvent this kind of identifier-based detection by modifying a single bit of the malware. Furthermore, there is always a time delay for updating the malware dictionary, so that the newly published malware cannot be found in the dictionary. Nowadays, antivirus companies usually adopt machine learning methods for malware detection. Clients locally extract some features of an app and sends these features to the server. The server runs a previously trained machine learning model on these features to detect if this app is malicious. In the next chapter, we introduce another technique that can make this process privacy-preserving as well.

4. Cloud-Assisted Prediction Services

Cloud-assisted prediction is commonly used in scenarios such as image recognition [20], online diagnosis [43] and credit-risk evaluation [8]. In particular, deep *neural networks* (also known as *deep learning*), has become increasingly popular since it gets record-breaking performance in many fields. On the other hand, such cloud-based prediction services put clients' privacy at risk since the input data submitted to the cloud service may contain sensitive information. A natural question to ask is, *given a model, whether is it possible to run it in an oblivious way*: the server learns nothing about clients' input, and clients learn nothing about the model except the prediction results. Efficient solutions have been proposed for machine learning models such as decision trees, SVM and naive Bayes [25, 14, 22, 110]. However, privacy-preserving deep learning models, which we refer to as *oblivious neural networks* (ONN), have not been sufficiently studied.

We propose the *first* technique that can transform any common neural network model into an oblivious neural network *without modifying the training phase*. To this end, we design oblivious protocols for the common operations used in neural network models. In particular, we make nonlinear functions such as sigmoid and tanh amenable to our transformation technique with a negligible loss in accuracy.

4.1 Oblivious neural networks (ONN)

A neural network model is typical defined as:

$$\mathbf{z} := (\mathbf{W}_L \cdot f_{L-1}(...f_1(\mathbf{W}_1 \cdot \mathbf{X} + \mathbf{B}_1)...) + \mathbf{b}_L) \tag{4.1}$$

It processes input data in a set of layers, by first applying *linear transformations* ($\mathbf{W}_i \cdot \mathbf{X} + \mathbf{B}_i$), followed by the application of a function $f_i()$ such as *activation functions* and *pooling operations*. In the prediction phase, all common linear transformations can be reduced to matrix multiplications and additions. Commonly used activation functions are: ReLU [109, 98, 76], leaky ReLU [59, 105],

maxout [28, 86] and tanh [32]. Commonly used pooling operations are mean and max pooling.

Mapping to the model introduced in Section 1.1.4, the desired functionality for the neural network prediction is defined in equation 4.2. The input of the server $\mathcal{S}$ is a neural network model M, and the input of a client $\mathcal{C}$ is a set of input features x. Other clients' input and output are $\perp$. An ONN allows $\mathcal{S}$ and $\mathcal{C}$ to privately evaluate a function f which represents the combination of all layers:

$$(\perp, z) = f(M, x); \text{ where } z \text{ is the prediction result} \tag{4.2}$$

Gilad-Bachrach et al. [50] propose CryptoNets, which have $\mathcal{C}$s encrypt their input data using somewhat homomorphic encryption and have $\mathcal{S}$ run the neural network on the encrypted data. However, somewhat homomorphic encryption only supports linear operations and low-degree polynomials. Therefore, during training, they use a specific activation function ("square") and pooling operation (mean pooling), which are not typically used in neural networks. As a result, their model only works in MNIST dataset. In addition, CryptoNets transformations incur a large performance overhead. Mohassel and Zhang [87] also propose new activation functions that can be efficiently computed by garbled circuits, and use them in the training phase. However, it still requires changes to the training phase. We suggest that an ideal ONN transformation technique should be able to deal with any neural networks without modify the training phase. This means the ability to handle common activation functions.

4.2 Adversary model

Similarly to our assumption about PMT, we assume that an adversary $\mathcal{A}$ can compromise either $\mathcal{S}$ or $\mathcal{C}$, but not both. A compromised $\mathcal{S}$ tries to learn $\mathcal{C}$s' inputs, and a compromised $\mathcal{C}$ tries to learn parameters in $\mathcal{S}$'s model. We assume $\mathcal{A}$ to be *semi-honest*, i.e., trying to learn their desired information by following the protocol.

We do not aim to protect the sizes of the query and the model, and which activation function is being used[1]. However, such information can be protected by adding dummy operations. By simply issuing queries, $\mathcal{C}$ can use $\mathcal{S}$'s prediction service to extract an equivalent or near-equivalent model (*model extraction* attacks [108]), or even infer the training set (*model inversion* [46] or *membership inference* attacks [102]). In a client-server setting, $\mathcal{S}$ can rate limit prediction requests from a given $\mathcal{C}$, thereby slowing down this attack or bounding this information leakage.

[1]Service providers usually make such information public in the forms of white papers or scientific articles. But they will not publish the parameters inside the model, which will reveal information about their training data.

4.3 Research questions

RQ3. Can we design a practical ONN transformation technique to convert a trained neural network model (trained with commonly used operations) to an ONN?

4.4 The MiniONN approach for ONN

4.4.1 Design overview

We propose MiniONN, the *first* technique that can transform any common neural network model into an *oblivious neural network*. The core idea of MiniONN is to have $\mathcal{S}$ and $\mathcal{C}$ *additively share* each of the input *and* output values for every layer of a neural network. That is, at the beginning of every layer, $\mathcal{S}$ holds $\mathbf{X}^{\mathcal{S}}$ and $\mathcal{C}$ holds $\mathbf{X}^{\mathcal{C}}$ s.t. $\mathbf{X}^{\mathcal{S}} + \mathbf{X}^{\mathcal{C}} \pmod N = \mathbf{X}$, which is the input of that layer in the non-oblivious version of that neural network. The output values $\mathbf{Y}^{\mathcal{S}}$ and $\mathbf{Y}^{\mathcal{C}}$ will be used as inputs for the next layer.

Figure 4 of Publication III shows the protocol for oblivious linear transformation. For each row of $\mathbf{W}$ and each column of $\mathbf{X}^{\mathcal{C}}$, $\mathcal{S}$ and $\mathcal{C}$ jointly generate a dot-product triplet (shown in Figure 3 of Publication III): $u + v \pmod N = \mathbf{w} \cdot \mathbf{x}^{\mathcal{C}}$ in the precomputation phase. In the online phase, $\mathcal{S}$ only needs to calculate $\mathbf{Y}^{\mathcal{S}} := \mathbf{W} \cdot \mathbf{X}^{\mathcal{S}} + \mathbf{B} + \mathbf{U}$, and $\mathcal{C}$ only needs to set $\mathbf{Y}^{\mathcal{C}} := \mathbf{V}$. As a result, each element of $\mathbf{Y}^{\mathcal{S}}$ and $\mathbf{Y}^{\mathcal{C}}$ satisfy:

$$\begin{aligned} y^{\mathcal{S}} + y^{\mathcal{C}} &= \mathbf{w} \cdot \mathbf{x}^{\mathcal{S}} + b + u + v \\ &= w_1(x_1 - x_1^{\mathcal{C}}) + , ..., + w_l(x_l - x_l^{\mathcal{C}}) + b + u + v \\ &= (w_1 x_1 + , ..., + w_l x_l + b) - (w_1 x_1^{\mathcal{C}} + , ..., + w_l x_l^{\mathcal{C}}) + u + v \\ &= y \end{aligned}$$

Next, we introduce the notation of an oblivious activation function which receives $y^{\mathcal{C}}$ from $\mathcal{C}$ and $y^{\mathcal{S}}$ from $\mathcal{S}$, and outputs $x^{\mathcal{C}}$ to $\mathcal{C}$ and $x^{\mathcal{S}} := f(y^{\mathcal{S}} + y^{\mathcal{C}}) - x^{\mathcal{C}}$ to $\mathcal{S}$, where $x^{\mathcal{C}}$ is a random number generated by $\mathcal{C}$. We classify activation functions into two categories: piecewise-linear functions and non-piecewise-linear functions. For piecewise-linear functions (e.g., ReLU), we can easily compute them obliviously using a secure two-party computation (2PC) protocol. For non-piecewise-linear functions (e.g., sigmoid), unlike piecewise-linear functions, it is non-trivial to make them oblivious using 2PC.

To this end, we approximate a non-piecewise-linear function $f()$ by splitting it into several pieces and approximating each piece using a linear function. The linear functions are chosen such that the overall goodness-of-fit is maximized. The approximation method is detailed in Section 5.3.2 of Publication III. The ap-

proximated version is in fact a piecewise-linear function, so it can be transformed in the same way using a 2PC protocol.

For pooling layers, we distinguish between max-pooling and mean-pooling. For mean-pooling, we just have $\mathcal{S}$ and $\mathcal{C}$ calculate the sum of their respective shares and keep track of the divisor. For max-pooling, we use a 2PC protocol to reconstructs each y_i and return the largest, masked by a random number.

4.4.2 Implementation and evaluation

We fully implemented MiniONN in C++ and evaluated its performance. We measured response latency (including the network delay) and message sizes during the whole procedure, i.e., from the time $\mathcal{C}$ begins to generate its request to the time it obtains the final prediction. The settings can be found in Section 6 of Publication III.

The results in Table 2 and Table 3 of Publication III show that MiniONN achieves significantly lower response latency and message sizes compared with previous work SecureML [87] and CryptoNets [50].

We trained new neural network models for three popular datasets in machine learning: MNIST, CIFAR-10 and PTB. For the PTB dataset, we used the real sigmoid as the activation functions for training, but replaced them with their corresponding approximations for predictions. The models can be found in Section 6.2 of Publication III. The latency and message size of all three neural networks after being transformed by MiniONN are shown in Table 4.1. The performance of the ONNs for MNIST and PTB is reasonable, but the ONN for CIFAR-10 is expensive. This is because the model in CIFAR-10 (Figure 13 in Publication III) is quite large: it has 7 activation layers, and each layer has $2^{10} - 2^{16}$ neurons.

	Latency (s)		Message Sizes (MB)		Accuracy
	offline	online	offline	online	ONN (Original)
ReLU/CNN/MNIST (Figure 12 of Publication III)	3.58	5.74	20.9	636.6	99.31% (99.31%)
ReLU/CNN/CIFAR-10 (Figure 13 of Publication III)	472	72	3046	6226	81.61% (81.61%)
Sigmoidal/LSTM/PTB (Figure 14 of Publication III)	13.9	4.39	86.7	474	cross-entropy loss:4.76 (4.74)

Table 4.1. Performance of MiniONN transformations of models with common activation functions and pooling operations (taken from Publication III). Server-side: Intel Core i5 CPU with 4 3.30 GHz cores and 16 GB memory; Client-side: Intel Core i5 CPU machine with 4 3.20 GHz cores and 8 GB memory.

4.5 Summary and discussion

We present MiniONN, the first approach that can transform any neural network into an oblivious form. Our benchmarks show that MiniONN achieves lower response latency and message sizes than prior work [87, 50]. However, it is still too expensive to transfer very large and deep neural networks. Several follow-up works have tried to improve the performance of MiniONN. Chiraag et al. propose a fast homomorphic matrix-vector multiplication protocol to avoid the generation of dot-product triplets [70]. As a result, the precomputation phase has been significantly improved. Riazi et al. present Chameleon which generates dot-product triplets in a more efficient way using a semi-honest third party (STP), The STP can be a separate computing node or it can be implemented based on TEEs [97]. However, both improvements still need to run garbled circuits for each neuron in the online phase in the same way as MiniONN.

An interesting question for future work is how to make full use of TEEs to improve the performance of MiniONN. One option is to have $\mathcal{S}$ offload its model to the client-side TEEs and run the predictions on client-side. This solution is suitable for the case where the $\mathcal{C}$ has no persistent network connections. Another option is to keep the model on server-side and have the TA run the model and answer $\mathcal{C}$'s requests. In addition to the performance improvements, both solutions can also enforce policy checking on the requests. For example, before answering the queries, TA can analyze them to detect and drop potentially adversarial queries for model inversion [46] and membership inference attacks [102].

5. Cloud-Assisted Storage Services

As cloud storage services become increasingly popular and user bases keep growing, cloud storage providers begin to save storage space via *cross-user deduplication*: if different users upload the same file, the server stores a single copy. On the other hand, it is in the interests of users to encrypt their files via *semantically secure* encryption schemes on client-side. In this case, identical files are uploaded as completely independent ciphertexts, which thwarts deduplication. Finding a way to do *secure deduplication of encrypted data* (SDoE) is an active research topic. To this end, we propose the first single-server solution and it can prevent online brute-force attacks by a compromised server.

5.1 Secure deduplication of encrypted data (SDoE)

There are two functions to be mapped to the model introduced in Section 1.1.4: upload f_{upload} and download f_{download}. During uploading, the server $\mathcal{S}$ inputs its local storage Φ of encrypted files, and the uploader $\mathcal{C}_i$ inputs the file F_i it wants to upload. In addition, other clients $\mathcal{C}_j$ input (F_j, k_{F_j}), where F_js are the files they have uploaded and k_{F_j}s are the encryption keys of these files. They jointly evaluate f_{upload}:

$$(\Phi', \perp, ..., k_{F_i}, ..., \perp) = f_{\mathsf{upload}}(\Phi, (F_1, k_{F_1}), ..., F_i, ..., (F_n, k_{F_n})) \tag{5.1}$$

where Φ' is the new storage after uploading: if $F_i \in \{F_1, ..., F_n\}$, $\Phi' = \Phi$. Otherwise, $\Phi' = \Phi \cup E(k_{F_i}, F_i)$. During downloading, $\mathcal{S}$ inputs its storage Φ and $\mathcal{C}_i$ inputs k_{F_i}. They jointly evaluate f_{download}:

$$(\Phi', F_i) = f_{\mathsf{download}}(\Phi, k_{F_i}) \tag{5.2}$$

An SDoE scheme allows $\mathcal{S}$ and $\mathcal{C}$s to privately evaluate f_{upload} and f_{download}.

The most commonly used SDoE scheme is *convergent encryption* [38], which uses the file hash as the encryption key. Consequently, identical files will lead to the same ciphertext, and deduplication can be easily done. However, this solution is vulnerable to *offline brute-force attacks* by a compromised $\mathcal{S}$ since

it is deterministic. Other solutions require the aid of additional *independent* servers [17, 96, 106], which is a strong assumption that is very difficult to meet in commercial contexts (more discussions about this are referred to Section 6.2). Furthermore, these schemes are vulnerable to *online brute-force attacks* by a compromised $\mathcal{S}$ colluding with some $\mathcal{C}$s.

5.1.1 Adversary model

We assume that an adversary $\mathcal{A}$ can compromise the uploader $\mathcal{C}_i$, the server $\mathcal{S}$, any subset of $\mathcal{C}_j$s, or any collusion of these parties. The goal of $\mathcal{A}$ is to learn the files of the entities it has not compromised. We assume $\mathcal{A}$ behaves arbitrarily for a single upload procedure, e.g., refuses to participate in the protocol, substitutes its inputs with other values, or aborts the protocol prematurely. In addition, when considering the long-term operation of the system, $\mathcal{A}$ can perform the following attacks depending on the perspective:

- *Compromised $\mathcal{C}_i$:* for a predictable file F, an uploader can upload all possible files that F can be, and observe which one causes deduplication from the side-channels[1] between $\mathcal{S}$ and $\mathcal{C}$ [61];
- *Compromised $\mathcal{S}$:* if $\mathcal{S}$ gets a deterministic representation (e.g., cryptographic hash or convergent encryption) of a predictable file, it can easily test all possible files;
- *Compromised $\mathcal{S}$ and $\mathcal{C}$s:* they can run the protocol for every "guess" and check whether deduplication happens.

In addition, a compromised $\mathcal{S}$ can easily detect whether a certain file has been uploaded by running the deduplication protocol once. Since $\mathcal{S}$ always knows that deduplication happens, no known SDoE scheme can prevent this attack. This attack is not included in our adversary model. (We model this attack formally in our security model in Section 5.3.2.)

5.2 Research questions

RQ5. Is it possible to design a single-server scheme for secure deduplication of encrypted data (SDoE)?

RQ6. What is a formal security model for SDoE that can capture long-term operation of the system?

[1] For example, in client-side deduplication, files are not required to be uploaded if deduplication happens.

5.3 PAKE-based SDoE

5.3.1 Design overview

In our solution, when an uploader $\mathcal{C}_i$ wants to upload a file F_i, it first sends a short hash $sh_i = SH(F_i)$ to the server $\mathcal{S}$. Since the short hash has a certain collision rate, $\mathcal{S}$ cannot uniquely identify F_i via an offline brute-force attack if F_i has enough entropy. Now, suppose that sh_i matches sh_j, which is a short hash of F_j previously uploaded by $\mathcal{C}_j$. $\mathcal{S}$ needs to figure out if this happens because $F_i = F_j$, and in that case, arrange to securely transfer k_{F_j} from $\mathcal{C}_j$ to $\mathcal{C}_i$. We achieve this by having $\mathcal{C}_i$ and $\mathcal{C}_j$ run an oblivious key sharing protocol so that $\mathcal{C}_i$ can receive k_{F_j} if and only if $F_i = F_j$, and a random key otherwise. $\mathcal{C}_j$ is referred to as a *checker* in this protocol.

Our oblivious key sharing protocol is based on *password authenticated key exchange (PAKE)*: $\mathcal{C}_i$ and $\mathcal{C}_j$ use h_i and h_j (which are the cryptographic hashes of their files F_i and F_j) as their respective input "passwords" to run a PAKE, $\mathcal{C}_i$ gets k_i and $\mathcal{C}_j$ gets k_j, which are equal if $h_i = h_j$ and are independent otherwise. Next, $\mathcal{C}_j$ uses k_j to deliver k_{F_i} to $\mathcal{C}_i$, which is equal to k_{F_j} iff $k_i = k_j$. A naive solution would just have $\mathcal{C}_j$ send $E(k_j, k_{F_j})$ to $\mathcal{C}_i$, but it allows a subtle attack from $\mathcal{C}_i$. More details about the attack and solution are elaborated in Section 5 of Publication IV.

To this end, $\mathcal{C}_i$ can use this key to encrypt its file, and $\mathcal{S}$ can perform deduplication if the ciphertext is equal to the one uploaded by $\mathcal{C}_j$. The protocol is shown in Figure 6 of Publication IV.

To prevent compromised $\mathcal{S}$ and $\mathcal{C}$ run online brute-force attacks for every "guess", each $\mathcal{C}$ enforce a rate limit on each of its files. Specifically, it sets a bound on the maximum number of PAKE runs it would serve as a checker or as an uploader for each file.

However, additional attacks are still possible when considering the long-term operation of the system. For example, a malicious client can upload a file and then pretend to be offline. Later it uploads the same file using another identity. If it gets the the same key as the one it got before, it knows that the file has been uploaded by someone else. We address this issue by having $\mathcal{C}$s always get random keys when they upload their files. We propose two schemes. The first one (shown in Figure 2 of Publication V) borrows the idea from proxy re-encryption [12]. $\mathcal{S}$ only keeps a single copy of duplicated files. When $\mathcal{C}$ wants to download its file, $\mathcal{S}$ re-encrypts the file so that $\mathcal{C}$ will download the same ciphertext as the one it uploaded. However, this scheme requires public-key operations on the entire file, which is not efficient for large files.

We observe that confidential files are usually small and unpopular. So we propose a second scheme (shown in Figure 3 of Publication V) that only deduplicates popular files and only protects the privacy of unpopular files. For unpopular files, $\mathcal{C}$s get random keys and download the same ciphertexts as they uploaded.

If those files become popular later, $\mathcal{S}$ deletes all duplicated copies and provides a way similar to re-encryption to help $\mathcal{C}$s to transform their keys to the right keys.

5.3.2 Security model

We model the security of an SDoE scheme with games played between an adversary $\mathcal{A}$ and a challenger. The challenger holds some secrets and $\mathcal{A}$ can interact with the challenger using different queries. At the end of each game, $\mathcal{A}$ outputs what it has learned about the secrets and $\mathcal{A}$ wins if its output is correct.

We proposed different security models for malicious $\mathcal{S}$ and $\mathcal{C}$ respectively. For compromised $\mathcal{C}$s, we want to model that by interacting with $\mathcal{S}$, it cannot learn whether a file already exists in the cloud storage.

RegisterCorrupt() $\mathcal{A}$ can register a (new) compromised $\mathcal{C}$, so that it can perfectly impersonate $\mathcal{C}$ from this moment on.

Send(M) This oracle computes on the input message M following the SDoE scheme and returns the output message in the view of all corrupted parties to $\mathcal{A}$. It models that an adversary can tamper with single messages in the SDoE scheme.

Test() $\mathcal{A}$ signals the end of the security game to the challenger, ceases all the interaction with oracles, and outputs a pair (F^*, b^*).

Definition 1. *Let λ be the security parameter. Given the queries described above, we define the security experiment $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda)$ for an* SDoE *protocol Π against compromised clients as follows: $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda) = 1$ if $\mathcal{A}$ replies to* Test() *with (F^*, b^*) and either of the following events happens:*

- *If $b^* = 0$ and F^* has not been uploaded before.*
- *If $b^* = 1$ and F^* has been uploaded before.*

But none *of the following events happens before $\mathcal{A}$ outputs (F^*, b^*):*

- *$\mathcal{A}$ has impersonated honest $\mathcal{C}$s*
- *$\mathcal{A}$ has forced an honest $\mathcal{C}$ to send any messages.*

Definition 2. *We define the advantage of an adversary $\mathcal{A}$ in the experiment $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda)$ as*

$$\mathbf{Adv}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda) = 1] - \frac{1}{2}$$

For a compromised $\mathcal{S}$, we want to model that the only way for it to identify the content of a file is performing an online brute-force attack. We allow $\mathcal{A}$ that has compromised $\mathcal{S}$ to make the following types of queries in the security experiments:

RegisterCorrupt() The same as that of compromised $\mathcal{C}$s.

Send(M) The same as that of compromised $\mathcal{C}$s.

AccessDB() $\mathcal{A}$ gets all the ciphertexts on $\mathcal{S}$ and the uploader list of each ciphertext.

Execute(P,F) As the initiator, $\mathcal{A}$ invokes a complete (sub-)protocol P on the input file F and obtains all the messages exchanged, following the description of the protocol P.

Test() $\mathcal{A}$ outputs two files F_0, F_1 with equal length. Upon receiving F_0, F_1, the challenger chooses $b \xleftarrow{\$} \{0,1\}$ and replies with a ciphertext $C_b = Enc(k_{f_b}, F_b)$. $\mathcal{A}$ performs the above queries and then outputs a bit b^*.

Definition 3. *We define the security experiment* $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda)$ *for a* SDoE *scheme* Π *against a compromised server as follows:* $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda) = 1$ *if* $\mathcal{S}$ *replies to* Test() *with* $b^* = b$*, but* none *of the following events happens before* $\mathcal{A}$ *outputs the bit* b^**:*

- $\mathcal{A}$ *has included* F_0 *or* F_1 *in its online brute-force attacks.*
- $\mathcal{A}$ *has compromised the targeted* $\mathcal{C}$*.*

Recall that a compromised $\mathcal{S}$ can easily detect whether a certain file has been uploaded by running the deduplication protocol once. That means $\mathcal{A}$ has included either F_0 or F_1 in its online butte-force attacks.

Definition 4. *We define the advantage of an adversary* $\mathcal{A}$ *in the experiment* $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda)$ *as*

$$\mathbf{Adv}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda) = 1] - \frac{1}{2}$$

The detailed security model can be found in Section 2 of Publication V. We proved the final versions of our SDoE scheme to be secure in this model. The proof can be found in Section 4.1 of Publication V.

5.3.3 Simulation and evaluation

Our use of rate limiting and popularity thresholds can impact deduplication effectiveness. We used realistic simulations to study the effect of various parameter choices in our protocol on deduplication effectiveness. We used a dataset of Android apps[2] to approximate a dataset of media files such as audio and video files from many users. We assume that they follow the same distribution since they share some features: created by a few authors; published on online stores; obtained either by paying money or for free; and usually shared among users via some distribution sites. Since our Android apps dataset is relatively small, we used the Synthetic Minority Over-sampling (SMOTE) Technique [29] to generate

[2] `ssg.aalto.fi/projects/malware/`

extra samples. The details about the dataset and over-sampling can be found in Section 5 of Publication V.

A file usually has few upload requests when it is generated, and may become increasingly popular over time. To model this case, we assume the rate of upload requests of a single file follows the shape of a normal distribution $\mathcal{N}(\mu, \sigma^2)$ where μ and σ are chosen arbitrarily. Specifically, for a file F_i that has x_i total copies, the number of copies of F_i uploaded at time point t is

$$y_i = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(t-\mu_i)^2}{2\sigma_i^2}} x_i \tag{5.3}$$

Then the total number of files uploaded at time point t is $\sum y_i$. We assume that they are uploaded in random order. We did this for all time points and measured the final deduplication effectiveness.

To provide equal security guarantees with previous work [17], we set the number of possible files as 825 000. To ensure security, we set the length of short hash as 13. Then, a compromised $\mathcal{S}$ is allowed to have $825000/2^{13} \approx 100$ guesses. Thus we set the rate limit as 100, i.e., a $\mathcal{C}$ will run PAKE at most 100 times for a certain file as both uploader and checker. We use these parameters in our simulations and measure deduplication effectiveness using the *dedpulication percentage* ρ:

$$\rho = \left(1 - \frac{\text{Storage size}}{\text{Size of upload requests in total}}\right) \tag{5.4}$$

However, if the file size information is not available (which was the case in the dataset we used), we can approximate ρ as follows (on the assumption that most files are of roughly the same size):

$$\rho = \left(1 - \frac{\text{Number of all files in storage}}{\text{Total number of upload requests}}\right) \tag{5.5}$$

In fact, the deduplication effectiveness can be improved if the file size information is available. When sending the short hash, the uploader could include the file size level as well, since this information will be revealed any way. In this case, the server can only run PAKE for those files with the same short hash and in the same size level.

We first assume that all $\mathcal{C}$s are online during the simulation and all files will be deduplicated (not only popular files). The results show that the maximium ρ can be 94.85%, which is close to the perfect deduplication percentage of 97.59% where all files are deduplicated. Then, we assigned an *offline rate* to each $\mathcal{C}$ as its probability to be offline during one run of the deduplication protocol, and measured ρ by varying the offline rates. The results show that ρ is still reasonably high even for for relatively high offline-rate of up to 70%, but drops quickly beyond that.

Recall that the third scheme only deduplicates popular files that have a number of copies that are larger than a threshold, called *popularity threshold*. To investigate how the deduplication effectiveness will be affected, we set the

offline rate as 0.5, and run the simulation with different popularity thresholds. The results show that ρ drops quickly if the popularity threshold is larger than 32. That means we can get a high deduplication effectiveness if we deduplicate files that have more than 32 copies. The detailed results can be found in Figure 4 of Publication V.

5.4 Summary and discussion

We investigate the problem of secure deduplication of encrypted data (SDoE) and propose a formal security model for this problem. We propose two single-server SDoE protocols and prove their security in our model. We show that both of them can achieve reasonable deduplication effectiveness via simulations with realistic datasets.

As we pointed out in Chapter 5.1.1, a compromised $\mathcal{S}$ can easily detect whether a certain file is in the storage by running the deduplication protocol once, since $\mathcal{S}$ always knows that deduplication happens. For example, by this way, the Recording Industry Association of America (RIAA) can easily find that if a copyrighted file is in the storage. Then they can serve a warrant on the storage provider to find out the uploaders of that file. One possible solution for this problem is *deniable encryption* [26], which enables $\mathcal{C}$s to generate a decoy key that can lead to a plausible plaintext. But currently there is no deduplication scheme that supports deniable encryption. Another option is anonymization, i.e., $\mathcal{C}$s can store and retrieve their files via an anonymous channel, so that $\mathcal{S}$ has no idea about the uploader of each file. But this approach prevent $\mathcal{S}$ from keeping tack of the quota for each $\mathcal{C}$.

A simple SDoE scheme is to have the user identify sensitive files. The client-side program can then use convergent encryption for non-sensitive files and semantically secure encryption for sensitive files. This approach has three drawbacks: it is too burdensome for average users, it reveals which files are sensitive, and it foregoes deduplication of sensitive files altogether. We discuss user factors in privacy-preserving applications in Section 6.4.

6. Discussion and Conclusion

This chapter provides a discussion on the precautions and difficulties of building privacy-preserving cloud-assisted services. It also provides a summary of this dissertation and some directions on future work.

6.1 Cryptography vs. trusted hardware

Recall that secure multiparty computation offers the same security guarantee achieved by a trusted third party TTP: parties submit their inputs to TTP who computes and returns the corresponding output to each party so that no information has been leaked except the information that can be inferred from the outputs. Hardware-assisted TEEs can provide similar functionality as TTP so that secure multiparty computation can be made much simpler and more efficient. Furthermore, in many cases, use of hardware-assisted TEEs can make it possible to design privacy-preserving variants of services that would otherwise be not possible [74]. However, there are following caveats when using TEEs in practice.

Limited capabilities. Even though a TEE can be considered as a TTP but it has limited capabilities. For example, the ARM Trust Zone device configuration that we used for the PMT prototype (Section 7, Publication II) only provides 1 MB of secure memory. Intel SGX provides a larger memory, but it is still limited to 94 MB. This limitation prevents a TEE from directly acting as a TTP in some scenarios such as SDoE and PMT where one party provides an extremely large input. In Publication II, we have to let the TEE cycle through the entire dictionary to prevent leakage via access patterns. If the dictionary is larger, we must use more complex cryptographic primitives such as ORAM. Furthermore, some functionalities are not available inside TEEs, e.g., GPU processing. A general solution is to split the code between a minimal code base running inside TEEs and code running outside [78]. Keeping the trust base minimal has another benefit of helping the developers to formally reason about the security of their code running inside TEEs, e.g., for the absence of backdoors.

Side channels. TEEs are known to be vulnerable to side channel attacks.

For example, memory management in SGX is left to the REE. Therefore, an attacker can force page faults at any point of the enclave execution to learn the secret-dependent enclave control flow or data access patterns from the requested pages [111]. Liu et al. [79] presented an even stronger attack by exploiting the fact that the CPU's level 3 (L3) cache is shared among all cores, and if the adversary can control the other cores when the enclave is executing, it can observe the enclave's memory access pattern at cache line (CL) granularity. As suggested by Intel [67], SGX enclave code that deals with sensitive information must use side channel resistant algorithms to process them. However, algorithms must be specifically designed for this purpose with a performance penalty [24].

Reliance on TEE manufacturers. Recall that remote attestation allows clients to verify that TEEs are running the expected code. However, clients need to rely on the TEE manufacturers to make sure that they are real TEEs. Furthermore, most TEE implementations are proprietary and closed-source, which makes it difficult for security experts to fully evaluate its security. Regardless of the academic efforts to develop open-source TEEs (e.g., Sanctum [33]) that can be fully examined, commercial TEEs still dominate the market of trusted hardware.

In conclusion, TEEs are useful tools for designing privacy-preserving cloud-assisted services. However, we need to be careful when using them: minimising the code running inside TEEs and thinking about fallbacks when TEEs are corrupt. A good example is the use of monotonic counters to prevent equivocations [77].

6.2 Trust distribution

Other than relying on TEEs, we can also simplify the design of privacy-preserving cloud-assisted services by distributing the trust, i.e., assuming multiple non-colluding servers.

With this assumption, we can realize PMT using traditional multi-server PIR [31, 51], which is efficient in both communication and computation. Take Gilboa and Ishai's two-server scheme as an example [51]. It only requires polylogarithmic query size and $O(n)$ symmetric key operations. Thus, we do not need to either transfer the whole dictionary or rely on TEEs. In addition, if there are multiple independent servers, we can adapt some more efficient MPC protocols [9, 47] to improve the performance of oblivious neural networks.

As we mentioned in Section 5, assuming an independent key server can also simplifies the design of SDoE [17]. Specifically, they improve convergent encryption by introducing an independent key server that holds a secret to assist the key generation. Uploaders can generate file keys by running an oblivious pseudorandom function (OPRF) with the key server. The OPRF allows uploaders to generate their keys without revealing their files, and without learning anything about the secret.

However, existence of multiple non-colluding servers is a strong assumption that is very difficult to meet in commercial contexts, since these servers must be set up by different organisations that will not collude with each other, and they may have different platforms, implementations and policies. In most cases, we have no way to verify if they have colluded or not. Furthermore, in some scenarios e.g., a ledger shared by different banks, multiple servers need to synchronise with each other, which may require some expensive consensus protocol [27, 80].

6.3 Edge computing

Recently, *edge computing* is proposed to optimise cloud-assisted services by processing the data at the edge of the network [91]. Recall that in cloud computing the resources are pooled together and their usage are centrally monitored and controlled. Edge computing, on the other hand, pushes resources away from centralized points to the extremes of a network, i.e., the edge. This reduces the communications bandwidth because only (intermediate) results are being transferred, instead of the raw data. Furthermore, since the data is processed at the edge, this paradigm also helps to reduce the information leakage for end users.

A good example of edge computing is *collaborative* (or *federated*) machine learning [101], which allows multiple parties to collaboratively train a model by having each party download the current model, improve it with local data, and then share the updated parameters with other parties. Since all the training data is kept locally, this paradigm is a potential solution for efficient yet privacy-preserving machine learning. However, recent work has shown that a malicious party is able to influence the learning process and deceive other parties into releasing more detailed information [63].

In conclusion, edge computing is a potential paradigm that can help to reduce the reliance on a central cloud server. However, we need to be careful about the information leakage from other side channels.

6.4 Human factors

We have been focusing on developing privacy-preserving cloud-assisted services from a cryptographic point of view. However, human factors should be considered as well when deploying the protocols.

Even if the protocol is proved to be secure, attackers can still break the protocol if they get the keys or other secrets which are assumed to be known only by the users. This can be achieved by hacking users' local machines, phishing users' passwords or even persuading or forcing a user to reveal the secret. Therefore, a basic design principle is to reduce the reliance on the participation of the users.

Another human factor that hinders the deployment of privacy-preserving cloud-assisted services is that end users pay less attention to their privacy. According to a survey from Gigaya [95], "people care more about convenience than privacy". They found that 60% of 4 000 polled respondents opt to use their Facebook, Twitter or Google account credentials to log in to other sites, even through most of them think that these sites will sell their data. Moreover, another survey done by Scott and Donald [99] shows that normal users would not like to pay so much (around $5) for their privacy guarantees. This phenomenon prevents companies from developing and deploying privacy-preserving cloud-assisted services. For those users who care about privacy, it is still a challenge to make them believe that the protocol really works, since we cannot expect every normal user to understand the technical details.

6.5 Laws and regulations

It is well-known that technology alone is not enough to ensure users' security and privacy. For example, Anderson [7] noticed that security failures are usually due to perverse incentives instead of the lack of suitable technical protection mechanisms. On the other hand, as mentioned in Section 6.4, companies may not have enough incentives to protect users' privacy. Therefore, related laws and regulations are needed to be made to force attackers to reconsider the tradeoff between benefit and risk of performing attacks, and to force companies to develop privacy-preserving mechanisms.

The EU General Data Protection Regulation (GDPR) [90] is such an example. It makes "privacy by design" as a part of the legal requirement. Specifically, it forces companies to include data protection mechanism form the onset of the designing of the systems, rather than an addition. With the enforcement of these regulations, we believe privacy-preserving cloud-assisted services will be widely deployed eventually.

6.6 Conclusion and future work

In this dissertation, we investigate the privacy issues in three cloud-assisted services: lookup service, prediction service and storage service. We suggest that privacy should be taken into account when we design such services.

To this end, we first provide a general definition for privacy-preserving cloud-assisted services. Then, we show how to adapt aforementioned three services (i.e., cloud-assisted malware checking, cloud-assisted machine learning prediction and cloud storage service) to this form. We provide instantiations for the all these privacy-preserving cloud-assisted services:

1. We build a privacy-preserving cloud-assisted lookup service via two different kinds of solutions: precomputed private set intersection and trusted

hardware.

2. We build a privacy-preserving cloud-assisted prediction service via secure two-party computation combined with additively homomorphic encryption.

3. We build a cloud storage service that supports deduplication on encrypted data via password authenticated key exchange.

In the future work, we will keep improving the performance and security guarantees of the above work. For example, our privacy-preserving cloud-assisted prediction service is still too slow to be used for deep neural networks, and it leaks information from the prediction results. In addition, we will look into other cloud-assisted assisted services and make them privacy-preserving. We also want to propose a general solution for all kinds of cloud-assisted assisted services. We will investigate how to achieve this by combining cryptographic techniques with hardware-assisted TEEs. We will consider more about the human factors and laws requirements. Our ultimate goal is to make cloud-assisted services secure, usable and deployable.

References

[1] AMD Secure Processor. http://www.amd.com/en-us/innovations/software-technologies/security.

[2] GlobalPlatform: Device specifications for trusted execution environment. http://www.globalplatform.org/specificationsdevice.asp.

[3] How android users interact with their phones. https://yahooaviate.tumblr.com/image/95795838933.

[4] Kinibi Trusted Execution Environment (TEE). https://www.trustonic.com/products/kinibi.

[5] Amazon. Amazon Drive, 2017. https://www.amazon.com/clouddrive.

[6] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.

[7] R. Anderson. Why information security is hard - an economic perspective. In *Seventeenth Annual Computer Security Applications Conference*, pages 358–365, Dec 2001.

[8] Eliana Angelini, Giacomo di Tollo, and Andrea Roli. A neural network approach for credit risk evaluation. *The quarterly review of economics and finance*, 48(4):733–755, 2008.

[9] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 805–817, New York, NY, USA, 2016. ACM.

[10] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 535–548, New York, NY, USA, 2013. ACM.

[11] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 673–701, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[12] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, February 2006.

[13] M. Backes, A. Kate, M. Maffei, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *2012 IEEE Symposium on Security and Privacy*, pages 257–271, May 2012.

[14] Mauro Barni et al. Secure evaluation of private linear branching programs with medical applications. In *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security*, pages 424–439, Saint-Malo, France, 2009. Springer.

[15] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[16] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE, 2013.

[17] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In *USENIX Security*, pages 179–194. USENIX Association, 2013.

[18] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings*, pages 139–155, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[19] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84. IEEE Computer Society, 1992.

[20] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[21] Dan Bogdanov, Roman Jagomägis, and Sven Laur. A universal toolkit for cryptographically secure privacy-preserving data mining. In *Proceedings of the 2012 Pacific Asia Conference on Intelligence and Security Informatics*, PAISI'12, pages 112–126, Berlin, Heidelberg, 2012. Springer-Verlag.

[22] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[23] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, July 2014.

[24] Benjamin A Braun, Suman Jana, and Dan Boneh. Robust and efficient elimination of cache and timing side channels. *arXiv preprint arXiv:1506.00189*, 2015.

[25] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 498–507, 2007.

[26] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In Jr. Kaliski, BurtonS., editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 90–104. Springer Berlin Heidelberg, 1997.

[27] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[28] Jia-Ren Chang and Yong-Sheng Chen. Batch-normalized maxout network in network. *CoRR*, abs/1511.02583, 2015.

[29] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[30] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[31] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50, Oct 1995.

[32] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[33] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 857–874, Austin, TX, 2016. USENIX Association.

[34] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, pages 143–159. Springer, 2010.

[35] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, Jan 2012.

[36] National Vulnerability Database. Cve statistics, 2011. `http://web.nvd.nist.gov/view/vuln/statistics`.

[37] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[38] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 617–624, 2002.

[39] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Manual for using homomorphic encryption for bioinformatics. *Microsoft Research*, 2015.

[40] Google Drive. Google Drive, 2017. `https://www.google.com/drive/`.

[41] Dropbox. Dropbox, 2017. `https://www.dropbox.com/`.

[42] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[43] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the International Conference on Machine Learning*, 2013.

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

[45] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333, New York, NY, USA, 2015. ACM.

[46] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 17–32. USENIX Association, 2014.

[47] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 – May 4, 2017, Proceedings, Part II*, pages 225–255, Cham, 2017. Springer International Publishing.

[48] Gawker. Gcreep: Google engineer stalked teens, spied on chats., 2010. `http://gawker.com/5637234/`.

[49] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing*, STOC '09, page 169. ACM Press, 2009.

[50] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210, 2016.

[51] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 640–658, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[52] GLYNK. Chat & Meet New People Nearby. `https://play.google.com/store/apps/details?id=com.glynk.app&hl=en`.

[53] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[54] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[55] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[56] Google. Android Security 2015 Year In Review. `http://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2015_Report_Final.pdf`.

[57] Google. Google Cloud Prediction API. `https://cloud.google.com/prediction/docs/`.

[58] Google. Google Translate. `https://translate.google.com/`.

[59] Benjamin Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014.

[60] A. Gupta, M. Miettinen, M. Nagy, N. Asokan, and A. Wetzel. Peersense: Who is near you? In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 516–518, March 2012.

[61] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, 8(6):40–47, Nov 2010.

[62] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference (TCC'08)*, volume 4948 of *LNCS*, pages 155–175. Springer, 2008.

[63] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on Computer & communications security*, CCS '17, pages 1099–1112. ACM, 2017.

[64] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce (EC'99)*, pages 78–86, 1999.

[65] A. Iliev and S. W. Smith. Protecting client privacy with trusted computing at the server. *IEEE Security Privacy*, 3(2):20–28, March 2005.

[66] InfoWorld. Ibm debuts first watson machine-learning APIs, year = 2014, note = `https://www.infoworld.com/article/2822814/machine-learning/ibm-debuts-first-watson-machine-learning-apis.html`.

[67] Intel. Intel SGX and Side-Channels, 2017. `https://software.intel.com/en-us/articles/intel-sgx-and-side-channels`.

[68] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 145–161, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[69] Nicola Jones. Nature: Computer science: The learning machines, 2014. `http://www.nature.com/news/computer-science-the-learning-machines-1.14481`.

[70] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. *arXiv preprint arXiv:1801.05507*, 2018.

[71] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure ot extension with optimal overhead. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 724–741, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[72] Alexey Kirichenko. Personal communication. F-Secure, 2015.

[73] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[74] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N Asokan. Protecting web passwords from rogue servers using trusted execution environments. *arXiv preprint arXiv:1709.01261*, 2017.

[75] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[76] Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 464–472, 2016.

[77] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 1–14, Berkeley, CA, USA, 2009. USENIX Association.

[78] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rüdiger Kapitza, Christof Fetzer, and Peter Pietzuch. Glamdring: Automatic application partitioning for intel SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 285–298, Santa Clara, CA, 2017. USENIX Association.

[79] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, May 2015.

[80] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *arXiv preprint arXiv:1612.04997*, 2016.

[81] Jacob R. Lorch, Bryan Parno, James Mickens, Mariana Raykova, and Joshua Schiffman. Shroud: Ensuring private access to large-scale data in the data center. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, FAST'13, pages 199–214, Berkeley, CA, USA, 2013. USENIX Association.

[82] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1, New York, NY, USA, 2013. ACM.

[83] T. Meskanen, J. Liu, S. Ramezanian, and V. Niemi. Private membership test for bloom filters. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 515–522, Aug 2015.

[84] Microsoft. Microsoft OneDrive. `https://onedrive.live.com/`.

[85] Microsoft. Microsoft Azure Machine Learning Studio, 2014. `https://studio.azureml.net`.

[86] Dmytro Mishkin and Jiri Matas. All you need is a good init. *CoRR*, abs/1511.06422, 2015.

[87] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (S&P'17)*. IEEE, May 2017. `http://ieeexplore.ieee.org/document/7958569`.

[88] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, March 2004.

[89] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[90] EU Parliament. The EU General Data Protection Regulation. `https://www.eugdpr.org`.

[91] Milan Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[92] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 515–530, Berkeley, CA, USA, 2015. USENIX Association.

[93] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT'09*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[94] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 797–812, Berkeley, CA, USA, 2014. USENIX Association.

[95] The Washington Post. People care more about convenience than privacy online, 2014.

[96] Pasquale Puzio, Refik Molva, Melek Önen, and Sergio Loureiro. Cloudedup: Secure deduplication with encrypted data for cloud storage. In *CloudCom*, pages 363–370. IEEE Computer Society, 2013.

[97] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. *arXiv preprint arXiv:1801.03239*, 2018.

[98] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. APAC: augmented pattern classification with neural networks. *CoRR*, abs/1505.03229, 2015.

[99] Scott Savage and Donald M Waldman. The value of online privacy. 2013. `http://dx.doi.org/10.2139/ssrn.2341311`.

[100] Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. Predicting user traits from a snapshot of apps installed on a smartphone. *SIGMOBILE Mob. Comput. Commun. Rev.*, 18(2):1–8, June 2014.

[101] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1310–1321, New York, NY, USA, 2015. ACM.

[102] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE, 2017.

[103] Signal. Technology preview: Private contact discovery for Signal. `https://signal.org/blog/private-contact-discovery/`.

[104] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 71(1):57–81, Apr 2014.

[105] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[106] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 99–118, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[107] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM.

[108] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX, 2016. USENIX Association.

[109] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, number 3, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.

[110] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *Privacy Enhancing Technologies (PoPETs)*, 2016(4):335–355, 2016.

[111] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656, May 2015.

[112] Andrew C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

[113] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *Foundations of Computer Science (FOCS'82)*, pages 160–164. IEEE, 1982.

[114] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT'15*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

Aalto-DD 117/2018

ISBN 978-952-60-8043-7 (printed)
ISBN 978-952-60-8044-4 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science

www.aalto.fi

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

DOCTORAL
DISSERTATIONS