UNIVERSITY OF CALIFORNIA,
IRVINE

Sharing Sensitive Information with Privacy

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Networked Systems

by

Emiliano De Cristofaro

Dissertation Committee:
Professor Gene Tsudik, Chair
Professor Claude Castelluccia
Professor Athina Markopoulou

2011

To Eve — without you, none of this would be possible

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere appreciation to all the people who helped me during my doctoral work.

I am extremely grateful to Gene Tsudik – not only for the time, patience, and commitment dedicated to advising me, but also for sparkling my curiosity and serving as a precious source of inspiration. He has pushed me to improve myself both as a researcher and as a (relatively) young man (not to mention as a tentative runner, cigar smoker, and food enthusiast).

I would also like to express my gratitude to my doctoral committee for their encouragement and precious comments.

I am indebted to a number of exceptional people and researchers, whose teaching and feedback have tremendously contributed to my research education: Carlo Blundo, Jihye Kim, Stanislaw Jarecki, Claude Castelluccia, Giuseppe Ateniese – just to mention a few. It is not easy to explain how much I learned from them.

I am very thankful to my internship supervisors and co-workers, in particular, Dirk Westhoff, Jens-Matthias Bohli, Claude Castelluccia, Imad Aad, Valtteri Niemi.

I have also been very lucky to work with other great folks: Mishari Al Mishari, Enzo Auletta, Pierre Baldi, Roberta Baronio, Aniello Del Sorbo, Xuhua Ding, Roberto Di Pietro, Anthony Durussel, Karim El Defrawy, Aurelien Francillon, Julien Freudiger, Clemente Galdi, Paolo Gasti, Dali Kaafar, Yanbin Lu, Mark Manulis, Daniele Perito, Pino Persiano, Bertram Poettering, Rino Raimato, Andrei Serjantov, Claudio Soriente, Ivan Visconti.

I extend many thanks also to all the researchers that have invited me for talks, seminars, and research visits: Filipe Beato, Claudia Diaz, and Markulf Kohlweiss from KU Leuven, Julien Freudiger and Jean-Pierre Hubaux from EPFL, Srdjan Čapkun from ETH, Cynthia Kuo from NRC Palo Alto, Rafi Ostrovski and Ivan Visconti from UCLA, Xuhua Ding from SMU, Bryan Parno and David Molnar from MSR Redmond, Tomas Toft and Carmit Hazay from Aarhus University, Mark Manulis from TU Darmstadt: the fun of traveling as well as the feedback from different environments and communities have intensely motivated me.

Last, but not least, thanks to all the friends of the SPROUT lab not mentioned above (Di Ma, Ivan Martinovic, Einar Mykletun, Rishab Nithyanand, Gabriele Oligeri, Kasper Bonne Rasmussen, John Solis, Ersin Uzun), to the "Italian community" at UCI, to my surfing buddies Julius, JoJo, and Juris, to my roommates Adam, Charline, Clara, Claudio, Fabio, Joshua, Leo, Paco, Roby, Samia, Sandra, and to all the great people that I have met along the way: you have a great "responsibility" in making these past four years a lot of fun for me.

Curriculum Vitae

EMILIANO DE CRISTOFARO

EDUCATION

2007 – Present	*University of California, Irvine*, Ph.D. Candidate, Networked Systems, GPA 3.99
2000 – 2005	*Università di Salerno, Italy*, Laurea (5-year undergraduate program) in Computer Science, Summa Cum Laude

RESEARCH EXPERIENCE

6/2010 - 10/2010	*Nokia Research Center, Lausanne, Switzerland*, PhD Intern
9/2009 - 12/2009	*INRIA Rhône Alpes, France*, Research Visitor
1/2009 - 2/2009	*Singapore Management University*, Research Visitor
6/2008 - 9/2008	*NEC Europe Labs, Heidelberg, Germany*, PhD Intern

TEACHING EXPERIENCE

3/2011 - 6/2011	*University of California, Irvine*, Teaching Assistant: Network Security
2/2006 - 7/2006	*Università di Salerno*, Teaching Assistant: Network Programming.

REFEREED CONFERENCE PUBLICATIONS

1. P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, G. Tsudik. *Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes.* 18th ACM Conference on Computer and Communications Security (CCS'11), Chicago, Illinois, October 2011.
2. C. Castelluccia, E. De Cristofaro, A. Francillon, M.A. Kafaar. *EphPub: Toward Robust Ephemeral Publishing.* 19th IEEE International Conference on Network Protocols, Vancouver, Canada, October 2011.

3. E. De Cristofaro, Y. Lu, G. Tsudik. *Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information.* 4th International Conference on Trust and Trustworthy Computing (TRUST'11), Pittsburgh, Pennsylvania, June 2011.
4. E. De Cristofaro, C. Soriente. *PEPSI: Privacy Enhancing Participatory Sensing Infrastructure.* 4th ACM Conference on Wireless Security (WiSec'11), Hamburg, Germany, June 2011.
5. E. De Cristofaro, M. Manulis, B. Poettering. *Private Discovery of Common Social Contacts.* 9th International Conference on Applied Cryptography and Network Security (ACNS'11), Nerja, Spain, June 2011.
6. E. De Cristofaro, A. Durussel, I. Aad. *Reclaiming Privacy for Smartphone Applications.* 9th IEEE International Conference on Pervasive Computing and Communications (PerCom'11), Seattle, Washington, March 2011.
7. G. Ateniese, E. De Cristofaro, G. Tsudik. *(If) Size Matters: Size-Hiding Private Set Intersection.* 14th IACR International Conference on Practice and Theory of Public Key Cryptography (PKC'11), Taormina, Italy, March 2011.
8. E. De Cristofaro, J. Kim, G. Tsudik. *Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model.* 17th IACR International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt'10), Singapore, December 2010.
9. C. Castelluccia, E. De Cristofaro, D. Perito. *Private Information Disclosure from Web Searches.* 10th Privacy Enhancing Technologies Symposium (PETS'10), Berlin, Germany, July 2010.
10. E. De Cristofaro and G. Tsudik. *Practical Private Set Intersection Protocols with Linear Complexity.* 14th International Conference on Financial Cryptography and Data Security (FC'10), Tenerife, Spain, January 2010.
11. V. Auletta, C. Blundo, A. De Caro, E. De Cristofaro, G. Persiano, I. Visconti. *Increasing Privacy Threats in the Cyberspace: the Case of Italian e-Passports.* 1st Workshop on Lightweight Cryptography for Resource-Constrained Devices (WLC'10), Tenerife, Spain, January 2010.
12. E. De Cristofaro, S. Jarecki, J. Kim, G. Tsudik. *Privacy-preserving Policy-based Information Transfer.* 9th Privacy Enhancing Technologies Symposium (PETS'09), Seattle, Washington, August 2009.

13. E. De Cristofaro, X. Ding, G. Tsudik. *Privacy-preserving Querying in Sensor Networks.* 18th IEEE International Conference on Computer Communications and Networks (IC-CCN'09), San Francisco, California, August 2009.

14. E. De Cristofaro, J.M. Bohli, D. Westhoff. *FAIR: Fuzzy-based Aggregation providing In-network Resilience for real-time WSNs.* 2nd ACM Conference on Wireless Network Security (WiSec'09), Zurich, Switzerland, March 2009.

15. C. Blundo, E. De Cristofaro, A. Del Sorbo, C. Galdi, G. Persiano. *A Distributed Implementation of the Certified Information Access Service.* 13th European Symposium on Research in Computer Security (ESORICS'08), Malaga, Spain, October 2008.

16. C. Blundo, E. De Cristofaro, C. Galdi, G. Persiano. *Validating Orchestration of Web Services with BPEL and Aggregate Signatures.* 6th IEEE European Conference on Web Services (ECOWS'08), Dublin, Ireland, November 2008.

17. E. De Cristofaro. *A Secure and Privacy-Protecting Aggregation Scheme for Sensor Networks.* 8th IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM'07), Helsinki, Finland, June 2007.

18. V. Auletta, C. Blundo, E. De Cristofaro. *HTTP over Bluetooth: a J2ME experience.* IARIA International Journal On Advances in Telecommunications. Vol. 1, 2007.

19. V. Auletta, C. Blundo, E. De Cristofaro, S. Cimato, G. Raimato. *Authenticated Web Services: A WS-Security Based Implementation.* 2nd IFIP International Conference on New Technologies, Mobility, and Security (NTMS'07), Paris, France, May 2007.

20. C. Blundo and E. De Cristofaro. *A Bluetooth-based JXME infrastructure.* 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA'07), Vilamoura, Portugal, November 2007.

21. V. Auletta, C. Blundo, E. De Cristofaro. *A J2ME transparent middleware to support HTTP connections over Bluetooth.* 2nd IARIA International Conference on Systems and Network Communications (ICSNC'07), Cap Esterel, France, August 2007.

22. V. Auletta, C. Blundo, E. De Cristofaro, G. Raimato. *Performance Evaluation for Web Services invocation over Bluetooth.* 9th ACM Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM'06), Torremolinos, Spain, October 2006.

23. V. Auletta, C. Blundo, E. De Cristofaro, G. Raimato. *A lightweight framework for Web Services invocation over Bluetooth.* 4th IEEE International Conference on Web Services (ICWS'06), Chicago, Illinois, September 2006.

INTERNATIONAL JOURNAL PUBLICATIONS

1. E. De Cristofaro and J. Kim. *Some like it private: Sharing Confidential Information based on Oblivious Authorization.* IEEE Security and Privacy, July-August, 2010.

HONORS AND AWARDS

Fall 2010	*Dissertation Fellowship – UC Irvine*
2007 - 2011	*Dean's Fellowship – Donald Bren School of Information and Computer Science, UC Irvine*

Abstract of the Dissertation

Sharing Sensitive Information with Privacy

by

Emiliano De Cristofaro

Doctor of Philosophy in Networked Systems

University of California, Irvine, 2011

Professor Gene Tsudik, Chair

Modern society is increasingly dependent on (and fearful of) massive amounts and availability of electronic information. There are numerous everyday scenarios where sensitive data must be — sometimes reluctantly or suspiciously — shared between entities without mutual trust. This prompts the need for mechanisms to enable limited (privacy-preserving) information sharing. A typical scenario involves two parties: one seeks information from the other, that is either motivated, or compelled, to share only the requested information. We define this problem as *privacy-preserving sharing of sensitive information* and are confronted with two main technical challenges: (1) how to enable this type of sharing such that parties learn no information beyond what they are entitled to, and (2) how to do so efficiently, in real-world practical terms.

This dissertation presents a set of efficient and provably secure cryptographic protocols for privacy-preserving sharing of sensitive information. In particular, Private Set Intersection (PSI) techniques are appealing whenever two parties wish to compute the intersection of their respective sets of items without revealing to each other any other information (beyond set sizes). We motivate the need for PSI techniques with various features and illustrate several concrete variants that offer significantly higher efficiency than prior work. Then, we introduce the concepts of *Authorized* Private Set Intersection (APSI) and *Size-Hiding* Private Set Intersection (SHI-PSI). The former ensures that each set element is authorized (signed) by some mutually trusted authority and prevents arbitrary input manipulation. The latter hides the size of the set held by one of the two entities, thus, applying to scenarios where both set contents and set size represent sensitive information.

Finally, we investigate the usage of proposed protocols in the context of a few practical applications. We build a *toolkit* for sharing of sensitive information, that enables (practical) privacy-preserving database querying. Furthermore, motivated by the fast-growing proliferation of personal wireless computing devices and associated privacy issues, we design a set of collaborative applications involving several participants willing to share information in order to cooperatively perform operations without endangering their privacy.

Chapter 1

Introduction

In this chapter, we introduce the general concept of sensitive information sharing and motivate our work on privacy. We also summarize major contributions and present dissertation's outline.

1.1 Sharing Sensitive Information with Privacy

The notion of *privacy* is commonly described as the ability of an individual or a group to seclude information about themselves, and thereby reveal it selectively. In many nations, laws or constitutions protect privacy as a fundamental individual right [3, 9, 4]. The availability of information *about* an individual may result in having power *over* that individual, hence, generating concerns on potential misuse by governments, corporations, or other individuals [67].

In recent years, advances in computer and communication technologies have significantly amplified privacy risks. Nowadays, data is routinely exchanged electronically and collected by third parties. Privacy concerns are no longer limited to the anonymity and untraceability of digital activities. The disclosure of private information yields an increasing number of legal, monetary, practical, or even emotional, privacy issues.

However, the need for controlled (privacy-preserving) sharing of sensitive information occurs in many realistic scenarios, ranging from national security to individual privacy protection. A typical setting involves two parties: one that seeks information from the other that is either motivated, or compelled, to share (only) the requested information.

Consequently, in numerous occasions, there is a tension between information sharing and privacy. On the one hand, sensitive data needs to be kept confidential; on the other hand, data owners may be willing, or forced, to share information. We consider the following examples:

- ***Aviation Safety:*** The U.S. Department of Homeland Security (DHS) needs to check whether any passengers on any flight to/from the United States must be denied boarding or disembarkation, based on several secret lists, e.g., the *Terror Watch List* (TWL) [58]. Today, airlines submit their passenger manifests to the DHS, along with a large amount of sensitive information, including credit card numbers [143]. Besides its obvious privacy implications, this *modus operandi* poses liability issues with regard to mostly innocent passengers' data and concerns about possible data loss. (See [33] for a litany of recent incidents where large amounts of sensitive data were lost or mishandled by government agencies.) Ideally, the DHS would obtain information pertaining *only* to passengers on one of its watch-lists, without disclosing any information to the airlines.

- ***Healthcare:*** A health insurance company needs to retrieve information about a client from other insurance carriers or hospitals. Clearly, the latter cannot provide any information on other patients while the former cannot disclose the identity of the target client.

- ***Law Enforcement:*** An investigative agency (e.g., the FBI) needs to obtain electronic information about a suspect from another agency (e.g., the local police, the military, the DMV, the IRS) or from the suspect's employer. In many cases, it is dangerous (or forbidden) for the FBI to disclose the subject of its investigation. Whereas, the other party cannot disclose its entire dataset and trust the FBI to only extract desired information. Furthermore, FBI's requests might need to be pre-authorized by some appropriate authority (e.g., a federal judge issuing a warrant). This way, the FBI can only obtain information related to authorized requests.

- ***Social Networking:*** A social network user (Alice) wants to find out whether there are any other users nearby with whom she shares friends or group memberships, without relying on a third-party. Some of this information might be very sensitive, e.g., it might reveal Alice's medical issues or sexual orientation. Today, Alice would have to broadcast her information in order to discover a nearby "match", thus compromising her privacy. Whereas, Alice might be willing to disclose sensitive information only to users with a *matching* profile.

- ***Interest Sharing:*** Two or more users would like to share their common interests and activities, e.g., to discover matching locations, routes, preferences, or availabilities, without exposing any other information beyond the matching interests.

These examples motivate the need for privacy-preserving sharing of sensitive information and pose two main technical challenges: (1) how to enable this type of sharing such that parties learn no information beyond what they are entitled to, and (2) how to do so efficiently, in real-world practical terms.

1.2 Cryptographic Protocols and Open Problems

Technology advances have radically influenced our modes of communication and have equally prompted a number of privacy challenges. As a result, there has recently been a lot of research activities in the context of *Privacy-Enhancing Technologies* (PETs).[1] Modern Cryptography has played a key role within PETs, producing a number of effective cryptographic protocols for privacy protection.[2]

Below, we overview cryptographic protocols enabling *implicit authentication* and *oblivious information transfer*. We discuss their inter-dependence and highlight some open problems that have motivated our work.

1.2.1 Protocols for Implicit Authentication and Oblivious Information Transfer

Many cryptographic protocols can be defined as the secure and privacy-preserving implementation of a desired functionality [77]. They involve two or more players, each equipped with a private input, willing to compute the value of a public functionality f over their inputs. In doing so, they only learn the output of f and nothing else besides what can be deduced from the output. In other words, if protocol parties were to trust each other (or some outside party), then they could each send their local input to the trusted party, that would execute f and send each party the corresponding output. The main technical challenge is to let such a trusted party be "emulated" by mutually distrustful parties themselves. This paradigm is referred to as *Secure Multi-party Computation* (SMC) [148, 79]. SMC has been thoroughly investigated starting with Yao's garbled circuits [148], used to privately compute any function that can be expressed as a boolean circuit. For more details on SMC, we refer to [79, 121, 32, 113, 86].

Our work is focused on enabling privacy-preserving sharing of sensitive information. Its objective is the secure computation of specific functionalities, using *specialized protocols*, rather

[1]For a historical overview of Privacy-Enhanced Technologies, we refer readers to [76, 75, 60, 1, 36, 82].

[2]We argue that Modern Cryptography entails several different building blocks, including basic *cryptographic primitives* (e.g., digital signatures, encryption schemes, etc.) and more complex *cryptographic protocols*. As this dissertation focuses on the latter, we refer readers to [77, 104, 118] for an extensive background on *all* topics of Modern Cryptography.

than generic solutions. Our motivation is two-fold: (1) not all information sharing functionalities can be easily implemented using generic solutions (such as garbled circuits), and (2) it is often more efficient to design optimized special-purpose protocols.

The research community has proposed a number of interesting cryptographic protocols, that address a wide range of problems simultaneously encompassing security, privacy, authentication, and authorization. Based on prior results, we concentrate on two important directions: implicit authentication and oblivious information transfer.

Implicit Authentication

Traditional access control systems involve a client requesting access to a resource from a server. Client needs to demonstrate ownership of credentials to satisfy server's access control policies. Each party may choose to withhold more sensitive credentials until an adequate level of trust has been established through the exchange of less sensitive credentials. Nonetheless, ultimately one party must be the first to reveal its credential to the other party [90].

However, in several realistic scenarios, one might be willing to encrypt a resource such that the client gains access to it using its credentials, without revealing them to the server. To enable this type of functionality, several related cryptographic protocols have been proposed, including *Oblivious Signature Based Envelopes* [111, 127], *Hidden Credentials* [23], *Anonymous Credentials* [24, 28], and *Secret Handshakes* [8, 96, 35, 147, 99]. We review them in Section 3.1.

Oblivious Information Transfer

In many applications, entities request information from other parties, e.g., to retrieve messages, files, or database records. In many realistic scenarios, however, desired information is sensitive. Consider, for instance, a company querying a patent database server to verify the novelty of its product: the company may fear that the database server sells its recent queries to competitors. One trivial way to guarantee query privacy is to download the entire database and perform searches locally. However, this would introduce a significant bandwidth overhead; also, the server may be unwilling to release a copy of its entire database.

Several cryptographic protocols have been proposed to address this problem. The most prominent technique is *Oblivious Transfer* [136], that allows a sender to transfer one of potentially many messages to a receiver, remaining oblivious about the message transferred (if any). Other related concepts are *Private Information Retrieval* (PIR) [40, 74, 39, 131, 11] and *Private Set Inter-*

section (PSI) [66, 108, 87, 44, 85, 98, 100]. We discuss them in detail in Section 3.2.

In particular, we anticipate that PSI techniques constitute a fundamental part of this dissertation: they serve as the main building block to enable privacy-preserving sharing of sensitive information. PSI involves two parties, a server and a client, each with a private input set. PSI lets parties run a cryptographic protocol that only disclose to the client the set intersection, and nothing to the server (beyond client set size). Several PSI constructions have been proposed, with different complexities, tools, assumptions, and adversarial models. Prior work on PSI is extensively discussed in Section 3.3.

1.2.2 Some Open Problems

In recent years, there has been an increased interest in cryptographic protocols for implicit authentication and oblivious information transfer. Nonetheless, much remains to be done. Below, we identify and discuss several relevant open problems in the field, that we attempt to address in this dissertation.

Combining Oblivious Information Transfer and Implicit Authorization

Sensitive information is often requested by some authority based on some legitimate need. The challenge is how to allow access to only duly authorized information and, at the same time, to obtain needed information without divulging what is being requested. In other words, we need to enable mechanisms to obliviously transfer information on top of protocols that allow implicit authentication of interacting parties. In fact, one feature common to protocols for implicit authentication is the use of credentials that certify that a user is a member of a certain group. These are then (obliviously) used for authentication, to establish a secret, or to grant access to some resource. However, one *open problem* is how to adapt these concepts to settings where credentials are related to the *information* the client requests and is authorized on, rather than to a group membership. For instance, an FBI agent may be authorized to access a suspect's electronic file from an employer, given that the agent holds a valid warrant, issued by a court *explicitly* for that suspect. In doing so, the employer might need to remain oblivious to whether the requestor is a member of any specific organization, or whether it holds any credential at all.

To this end, our first research contribution – presented in Chapter 4 – introduces the concept of *Privacy-preserving Policy-based Information Transfer (PPIT)*, geared for any scenario with a need to transfer information between parties that: (1) are willing and/or obligated to transfer information in an accountable and policy-guided (*authorized*) manner, (2) need to ensure privacy of data owner by preventing unauthorized access, and (3) need to ensure privacy of requester's authorization(s) that grant it access to the data. We highlight and address some issues that arise in adapting protocols for implicit authentication (specifically, Hidden Credentials and Oblivious Signature-Based Envelopes) to the PPIT setting.

Arbitrary Inputs in Private Set Intersection

In the context of PSI, one important *open problem* is how to prevent malicious parties from altering their input sets. In PSI, the client learns the set intersection while the server learns nothing. In some setting, this may represent a severe threat to server's privacy. For instance, a malicious client may populate its input set with its best guesses of the server set (especially, if the set is easy to exhaustively enumerate). This would maximize the amount of information it learns. In the extreme case, the client could even claim that its set contain all possible items. Although the server could impose a limit on this size, the client could still vary its set over multiple protocol runs.

We argue that this issue cannot be effectively addressed without some mechanism to *authorize* client inputs. For this reason, we introduce the concept of *Authorized Private Set Intersection (APSI)* (in Chapters 5 and 6), where an off-line certification authority authorizes client input sets.

We show that APSI and PPIT concepts are related and attempt to bridge the gap between oblivious information transfer and implicit authorization.

Practicality of Available Private Set Intersection Protocols

Despite previously proposed PSI constructs, the quest for their efficiency is still underway. One *open problem* is how to design PSI protocols that involve a number of cryptographic operations (such as modular exponentiations), linear in the size of input sets. Prior results, e.g., [85, 98], asymptotically achieve this bound, however, their practicality is limited by the high cost of basic underlying operations. Also, we advocate the availability of PSI protocols that do not impose any expensive cryptographic operations on client side, thus, facilitating application scenarios where clients operate from limited-resource devices.

To this end, we design and implement a set of PSI constructions that improve the efficiency of state-of-the-art (Chapters 5 and 6). A thorough experimental analysis, in Appendix A, empirically confirms our improvements.

Hiding Input Sizes

One common feature of all protocols for oblivious information transfer, including PSI, is that client input size is always revealed to the server. This also applies to generic secure two-party computation techniques, such as Yao's garbled circuits [148]. However, in many scenarios, input size represents sensitive information, including the case of the Terror Watch List discussed above. Therefore, an interesting open problem in the context of PSI is how to keep client set size secret. In Chapter 7, we introduce the concept of *Size-Hiding Private Set Intersection (SHI-PSI)*.

1.3 Practical Aspects of Privacy-Preserving Sharing of Sensitive Information

Another challenge is how to build and deploy efficient mechanisms for sharing sensitive information with privacy. One possible concern is related to the computational and communication overhead introduced by cryptographic protocols for privacy protection. In addition, one needs to consider real-world application scenarios, thus, designing flexible and usable techniques.

To this end, we design and implement a *Toolkit for Privacy-preserving Sharing of Sensitive Information* (in Chapter 8). We consider realistic database-querying applications involving two parties: a server, in possession of a database, and a client, performing disjunctive equality queries. In doing so, the client does not disclose to the server its query, while the server is ensured that the client only obtains records matching the query. Although our main building blocks are PSI techniques, we address several interesting challenges, stemming from adapting PSI techniques to database settings. For instance, while in PSI set items are assumed to be unique, most databases contain duplicate values (e.g., “sex=female”).

Next, we turn to the *mobile environment* (in Chapter 9): we design collaborative applications involving participants—with limited reciprocal trust—willing to share sensitive information from their smartphones, and use it to (cooperatively) perform operations without endangering their privacy. We focus on two application scenarios: (i) privacy-preserving interest sharing, i.e., discovering shared interests without leaking users' private information, and (ii) private scheduling, i.e., privately determining common availabilities and location preferences that minimize associate costs.

1.4 Summary of Contributions

This dissertation investigates the design of efficient and provably secure mechanisms for privacy-preserving sharing of sensitive information.

1. We explore the relationship between cryptographic protocols for oblivious information transfer and implicit authentication. We motivate the need for efficient cryptographic protocols that: (1) allow access to only duly authorized information, and (2) release needed information without divulging what is being requested.

2. We focus on Private Set Intersection (PSI) techniques as our main building block. First, we aim at designing limited-overhead PSI constructions that are significantly more efficient than state-of-the art. Our protocols involve fast cryptographic operations linear in the size of input sets. Next, we introduce and instantiate Authorized Private Set Intersection (APSI), a PSI variant that prevents parties from arbitrarily manipulating their inputs. Finally, we motivate the need for Size-Hiding Private Set Intersection (SHI-PSI) and present the first PSI construction that hides the size of one party's input set.

3. We build an efficient and ready-to-use toolkit for privacy-preserving sharing of sensitive information, in the database context. As part of the toolkit design we address several challenges stemming from adapting PSI to database settings.

4. We present a novel architecture geared for privacy-sensitive smartphone applications where personal information is shared among smartphone users and decisions are made based on given optimization criteria.

1.5 Organization

This dissertation is organized as follows.

- Chapter 2 provides background information on notation, computational assumptions, adversarial models, and cryptographic tools.

- Chapter 3 discusses relevant related work in the context of privacy-preserving cryptographic protocols.

- Chapter 4 motivates and introduces the concept of Privacy-preserving Policy-based Information Transfer (PPIT). It formalizes PPIT functionality (alongside its security model) and presents three efficient instantiations.

- Chapter 5 constructs several PSI protocols, secure in the presence of semi-honest adversaries, that are significantly more efficient than state-of-the-art. We introduce the concept of APSI and show how some PPIT instantiations can be (inefficiently) adapted to APSI. We then propose a more practical APSI protocol and derive efficient PSI from it. Finally, we introduce an even more efficient PSI protocol geared for scenarios where the server performs some pre-computation and/or the client has limited computational resources.

- Chapter 6 proposes PSI protocols, that are secure in the presence of malicious adversaries, under standard assumptions. It proposes a linear-complexity APSI protocol in the malicious model – the first of its kind. Finally, it presents a (plain) PSI construction that is significantly more efficient than state-of-the-art.

- Chapter 7 introduces the concept of Size-Hiding in PSI, where the size of the set held by one party is hidden from the other.

- Chapter 8 presents the design and implementation of a toolkit for privacy-preserving sharing of sensitive information that uses efficient PSI protocols as its main building block.

- Chapter 9 investigates privacy-preserving techniques geared for mobile applications where sensitive information is shared between smartphone users.

- Chapter 10 concludes the dissertation and discusses outstanding research issues.

1.6 Collaboration

Most of the material in this thesis has been published in a preliminary form in conferences, workshops, and journals, co-authored with several researchers. Specifically, work presented in Chapter 4 has been done in collaboration with Stanislaw Jarecki and Jihye Kim [47], in Chapter 6 with Jihye Kim [48], and in Chapter 7 with Giuseppe Ateniese [5]. Also, Yanbin Lu collaborated on the results presented in Chapter 8 [49], while Anthony Durussel and Imad Aad – to work in Chapter 9 [46].

Part I

Foundations of Privacy-Preserving Sharing of Sensitive Information

Chapter 2

Preliminaries

In this chapter, we provide background information on notation, relevant computational assumptions, adversarial models, and cryptographic tools.

2.1 Notation

Negligible Function. A function $f(\tau)$ is *negligible* in the security parameter τ if, for every polynomial p, it holds that $f(\tau) < 1/|p(t)|$, for large enough t.

Signatures. Throughout this dissertation, we use public-key signature schemes, where each scheme is a tuple of algorithms $\mathsf{DSIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$, representing key setup, signature generation and verification, respectively. Specifically, $\mathsf{KGen}(\tau)$ returns a public/private key-pair, on input a security parameter τ. $\mathsf{Sign}_{\mathsf{sk}}(m)$ returns a signature σ on message m. Whereas, $\mathsf{Vrfy}_{\mathsf{pk}}(\sigma, m)$ returns 1 or 0 indicating that σ is valid or invalid signature on m, under pk.

Symmetric-Key Encryption. We also employ *semantically secure* symmetric encryption.[1] We assume the key space to be τ_1-bit strings, where τ_1 is a (polynomial) function of a security parameter τ. We use $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$ to denote symmetric-key encryption and decryption (both under key k), respectively.

[1]For a cryptosystem to be semantically secure, it must be infeasible for a computationally bounded adversary to derive significant information about a message (plaintext) when given only its ciphertext. For a formal definition of semantic security, refer to [104].

Random Values. We use $a \leftarrow_r A$ to designate that variable a is chosen uniformly at random from set A.

2.2 Cryptosystems

Schnorr Signatures [139]. Let p be a large prime and q be a large prime factor of $p-1$. Let g be an element of order q in $\mathbb{Z}_p^*$, $\mathcal{M}$ be the message space and $H_1 : M \rightarrow \mathbb{Z}_q^*$ be a suitable cryptographic hash function. The signer's secret key is: $a \leftarrow_r \mathbb{Z}_q^*$ and the corresponding public key is: $y = g^a \bmod p$. The values: p, q and y are public, while a is only known to the signer. A signature $\sigma = (e, s)$ on input message M is computed as follows:

1. Select a random value $k \in \mathbb{Z}_q^*$.
2. Compute $e = H_1(M, g^k \bmod p)$.
3. Compute $s = ae + k \bmod q$.

A Schnorr signature (e, s) on message M, is verified by checking that $H_1(M, g^s, y^{-e} \bmod p)$ matches e.

Paillier Cryptosystem [133]. Given a number $n = pq$ (where p and q are two large prime numbers), we define z as a n-th residue modulo n^2 if there exists a number $y \in \mathbb{Z}_{n^2}^*$ s.t. $z = y^n \bmod n^2$. The problem of deciding if z is a n-th residue is believed to be computationally hard, under so-called Decisional Composite Residuosity Assumption (CDRA). The Paillier cryptosystem involves the following algorithms:

- *Key generation:* Select $n = pq$ where p and q are two random large prime numbers. Pick a random generator $g \in \mathbb{Z}_{n^2}^*$ s.t. $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ exists, given that $\lambda = lcm(p-1, q-1)$ and $L(x) = \frac{(x-1)}{n}$. The public key is (n,g) and the private key is (λ,μ).
- *Encryption:* To encrypt a message $m \in \mathbb{Z}_n$, select a random $r \in \mathbb{Z}_n^*$ and compute the ciphertext: $E_r(m) \overset{\text{def}}{=} g^m \cdot r^n \bmod n^2$. Note that ciphertexts are elements of $\mathbb{Z}_{n^2}$.
- *Decryption:* Given a ciphertext $c \in \mathbb{Z}_{n^2}$, decrypt as: $L(c^\lambda \bmod n^2) \cdot \mu \bmod n$, where $L(x) = \frac{(x-1)}{n}$.

The Paillier cryptosystem is additively homomorphic, i.e.,, given $E_{r_1}(m_1) = (g)^{m_1} \cdot (r_1)^n \bmod n^2$ and $E_{r_2}(m_2) = (g)^{m_2} \cdot (r_2)^n \bmod n^2$, one can compute:

$$E_{r_1 r_2}(m_1 + m_2 \bmod n) = E_{r_1}(m_1) \cdot E_{r_2}(m_2) \bmod n^2 = (g)^{m_1+m_2} \cdot (r_1 r_2)^n \bmod n^2$$

Also note that, given $E_r(m) = g^m \cdot r^n \bmod n^2$ and $z \in \mathbb{Z}_n$, one can compute:

$$E_{r^z}(m \cdot z \bmod n) = E_r(m)^z \bmod n^2 = g^{mz} \cdot r^{nz} \bmod n^2$$

Identity Based Encryption (IBE) [142, 21]. We consider Boneh and Franklin's Identity Based Encryption (IBE) scheme [21]. It is composed by four algorithms: *setup*, *extract*, *encrypt*, *decrypt*.

- *Setup*: given a security parameter τ, is used to generate a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Then a random $s \in \mathbb{Z}_q^*$, a random generator $P \in \mathbb{G}_1$, P are chosen and Q is set such that $Q = sP$. (P, Q) are public parameters, whereas, s is the private master key. Finally, two cryptographic hash function, $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^\tau$ are also chosen.
- *Extract*: given a string $ID \in \{0,1\}^*$, is used to compute the corresponding private key $sH_1(ID)$.
- *Encrypt*: is used to encrypt a message M under a public key ID: for a picked random $r \in \mathbb{Z}_q^*$ the ciphertext is set to be $C = \langle U, V \rangle = \langle rP, M \oplus H_2(\hat{e}(Q, H_1(ID)^r)) \rangle$.
- *Decrypt*: is used to decrypt a ciphertext $C = \langle U, V \rangle$, by computing $M = V \oplus H_2(\hat{e}(U, sH_1(ID)))$.

2.3 Assumptions

We now present some cryptographic assumptions used in the rest of this dissertation.

Definition 2.1 (RSA Assumption on Safe Moduli). *Let* $\mathsf{RSA\text{-}Gen}(1^\tau)$ *be an algorithm that chooses two random primes* p', q' *s.t.* $|p'| = |q'| = \tau$ *and* $p = 2p' + 1$ *and* $q = 2q' + 1$ *are also primes, and outputs pairs* (N, e) *where* $N = pq$, e *is a small prime such that* $gcd(e, \phi(N)) = 1$. *We say that the* RSA problem on safe moduli *is* (τ, t)*-hard if, for every algorithm* $\mathcal{A}$ *running in time* t*, the probability:*

$$Pr[(N, e) \leftarrow_r \mathsf{RSA\text{-}Gen}(1^\tau), z \leftarrow_r \mathbb{Z}_N^* : A(N, e, z) = y \text{ s.t. } y^e = z \bmod N]$$

is a negligible function of τ.

Definition 2.2 (DDH Assumption). *Let $\mathbb{G}$ be a cyclic group and let g be its generator. Assume that the bit-length of the group size is τ. The* Decisional Diffie-Hellman problem (DDH) *is (τ, t)-hard in $\mathbb{G}$ if, for every efficient algorithm $\mathcal{A}$ running in time t, the probability:*

$$|\Pr[x, y \leftarrow_r \{0,1\}^\tau : A(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \leftarrow_r \{0,1\}^\tau : A(g, g^x, g^y, g^z) = 1]|$$

is a negligible function of τ.

Definition 2.3 (CDH Assumption). *Let g be a generator of a cyclic group $\mathbb{G}$ of order q. The* Computational Diffie-Hellman Problem (CDH) *in $\mathbb{G}$ is (τ, t)-hard if, for every algorithm $\mathcal{A}$ running in time t, the probability:*

$$\Pr[x, y \leftarrow_r \mathbb{Z}_q : \mathcal{A}(g, g^x, g^y) = g^{xy}]$$

is a negligible function of τ.

DDH oracle. A DDH oracle in group G is an algorithm that returns 1 on queries of the form (g, g^x, g^y, g^z) for $z = xy \bmod q$, and 0 on queries of the form (g, g^x, g^y, g^z) for $z \neq xy \bmod q$.

Definition 2.4 (GDH Assumption). *Let g be a generator of a cyclic group $\mathbb{G}$ of order q. The* Gap Diffie-Hellman Problem (GDH) *in group $\mathbb{G}$ is (τ, t)-hard if for every algorithm $\mathcal{A}$ running in time t, with access to the DDH oracle $\mathsf{DDH}_{\mathbb{G}}$ in group $\mathbb{G}$, the probability:*

$$\Pr[x, y \leftarrow_r \mathbb{Z}_q : \mathcal{A}^{\mathsf{DDH}_{\mathsf{G}}}(g, g^x, g^y) = g^{xy}]$$

is a negligible function of τ.

Definition 2.5 (BDH Assumption). *Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups of prime order q. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be an admissible bilinear map and let P be a generator of $\mathbb{G}_1$. The* Bilinear Diffie-Hellman Problem (BDH) *in $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$ is (τ, t)-hard if, for every algorithm $\mathcal{A}$ running in time t,the probability:*

$$\Pr[a, b, c \leftarrow_r \mathbb{Z}_q^* : \mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}]$$

is a negligible function of τ.

2.4 Tools

In this section, we consider signatures of knowledge of a discrete logarithm and equality of two discrete logarithms in a cyclic group $\mathbb{G} = \langle g \rangle$. In particular, we consider $\mathbb{G}$ where either its order or the bit-length of its order is known. Fujisaki and Okamoto [68] show that standard proofs of knowledge that work in a group of known order are also proofs of knowledge in this setting. We define discrete logarithm of $y \in \mathbb{G}$ with respect to base g as any integer $x \in \mathbb{Z}$ such that $y = g^x$ in $\mathbb{G}$. We assume a security parameter $\tau > 1$.

Definition 2.6 (ZK of DL over a known order group)**.** *Let $y, g \in \mathbb{G}$ of order q. A pair $(c, s) \in \{0,1\}^\tau \times \mathbb{Z}_q$ verifying $c = H(y||g||g^s y^c||m)$ as a signature of knowledge of the discrete logarithm of $y = g^x$ w.r.t. base g, on message $m \in \{0,1\}^*$.*

Definition 2.7 (ZK of DL over an unknown order group)**.** *Let $y, g \in \mathbb{G}$ where the group order is unknown, but its bit-length is known to be l. A pair $(c, s) \in \{0,1\}^\tau \times \pm\{0,1\}^{\epsilon(l+\tau)+1}$ verifying $c = H(y||g||g^s y^c||m)$ is a signature of knowledge of the discrete logarithm of $y = g^x$ w.r.t. base g, on message $m \in \{0,1\}^*$.*

The player in possession of the secret $x = \log_g y$ can generate the signature by choosing a random $t \in \mathbb{Z}_q$ (or $\pm\{0,1\}^{\epsilon(l+\tau)}$) and then computing c and s as: $c = H(y||g||g^t||m)$ and $s = t - cx$ in $\mathbb{Z}_q$ (or in $\mathbb{Z}$).

Definition 2.8 (ZK of EDL over a known order group)**.** *Let $y_1, y_2, g, h \in \mathbb{G}$ of order q. A pair $(c, s) \in \{0,1\}^\tau \times \mathbb{Z}_q$ verifying $c = H(y_1||y_2||g||h||g^s y_1^c||h^s y_2^c||m)$ is a signature of knowledge of the discrete logarithm of both $y_1 = g^x$ w.r.t. base g and $y_2 = h^x$ w.r.t. base h, on message $m \in \{0,1\}^*$.*

Definition 2.9 (ZK of EDL over an unknown order group)**.** *Let $y_1, y_2, g, h \in \mathbb{G}$ where the group order is unknown, but its bit-length is known to be l. A pair $(c, s) \in \{0,1\}^\tau \times \pm\{0,1\}^{\epsilon(l+\tau)+1}$ verifying that $c = H(y_1||y_2||g||h||g^s y_1^c||h^s y_2^c||m)$ is a signature of knowledge of the discrete logarithm of both $y_1 = g^x$ w.r.t. base g and $y_2 = h^x$ w.r.t. base h, on message $m \in \{0,1\}^*$.*

The player in possession of the secret $x = \log_g y_1 = \log_h y_2$ can generate the signature by choosing a random $t \in \mathbb{Z}_q$ (or $\pm\{0,1\}^{\epsilon(l+\tau)}$) and then computing c and s as: $c = H(y_1||y_2||g||h||g^t||h^t||m)$ and $s = t - cx$ in $\mathbb{Z}_q$ (or in $\mathbb{Z}$).

2.5 Adversarial Models

One distinguishing factor on the security of cryptographic protocols is the adversarial model which is typically either semi-honest or malicious. In the rest of this dissertation, the term *adversary* refers to insiders, i.e., protocol participants. Outside adversaries are not considered, since their actions can generally be mitigated via standard network security techniques. We follow the well-known formulations by Goldreich [77], summarized below.

Protocols secure in the presence of *semi-honest adversaries* (or honest-but-curious) assume that participants faithfully follow all protocol specifications and do not misrepresent any information related to their inputs, e.g., size and content. However, during or after protocol execution, any participant might (passively) attempt to infer additional information about the other participant's input. This model is formalized by considering an ideal implementation where a Trusted Third Party (TTP) receives the inputs of both participants and outputs the result of the defined function. Security in the presence of semi-honest adversaries requires that, in the real implementation of the protocol (without a TTP), each participant does not learn more information than in the ideal implementation.

Security in the presence of *malicious adversaries* allows arbitrary deviations from the protocol. In general, however, it does not prevent participants from refusing to participate in the protocol, modifying their inputs, or prematurely aborting the protocol. Security in malicious model is achieved if the adversary (interacting in the real protocol, without the TTP) can learn no more information than it could in the ideal scenario. In other words, a secure protocol emulates (in its real execution) the ideal execution that includes a TTP. This notion is formulated by requiring the existence of adversaries in the ideal execution model that can simulate adversarial behavior in the real execution model.

We refer to [77] for formal definitions of semi-honest and malicious behavior in general cryptographic protocols.

Chapter 3

Related Work

In this chapter, we discuss relevant related work in the context of privacy-preserving cryptographic protocols.

3.1 Cryptographic Protocols for Implicit Authentication

Techniques for implicit authentication leverage oblivious (or hidden) credentials to verify that a user is member of a certain group.

We briefly discussed implicit authentication protocols in Section 1.2.1; below, we overview state-of-the-art constructs.

Secret Handshakes (SH-s) [8, 96, 35, 147, 99]. SH-s (also called Affiliation-Hiding Authentication) allow two parties with group membership credentials issued by the same trusted entity – called *Group Authority* (GA) – to *privately* authenticate each other. Specifically, each party can prove to the other that it has a valid credential, however, this proof hides the identity of the issuing organization, unless the other party also has a valid credential from the same organization. An extension of the SH concept – known as Affiliation-Hiding Authenticated Key Exchange (AH-AKE) – can be used to establish a common shared secret upon success of the SH protocol [96, 97, 116]. Some protocols, such as [97], also support *multiple* credentials (i.e, multiple GAs), whereas, others relax GA trust assumptions [116].

Hidden Credentials (HC-s) [23]. Using HC-s, each party can create a public key corresponding to an arbitrary string (e.g., “FBI agent”) and the public key of a Trusted Third Party (TTP). Only the TTP can issue the corresponding private key to the “owner” of the string. One can then send messages to another entity based on credentials that she may or may not have. The sender may not know that the receiver is an FBI agent: however, the former is ensured that the latter decrypts the message only if knowing the private key corresponding to “FBI agent”. Note that this problem is similar to SH-s. However, SH-s require that parties mutually authenticate using credentials from the same issuer. In contrast, HC-s allow the sender to send a message depending only on receiver's credentials – the sender does not even need to have any credentials of her own.

Oblivious Signature-based Envelopes (OSBE-s) [111, 127]. OSBE-s allow a sender to release some information to a receiver conditional upon the latter's possession of a signature, issued by a trusted authority on a message known to both parties (e.g., “FBI agent”), while the sender learns nothing about the signatures held by the receiver. OSBE-s are very similar to HC-s, however, they require that parties agree on a message that the signature presumably signs. In other words, the sender needs to disclose its policy to the client.

Anonymous Credentials (AC-s) [24, 28]. AC-s allow a credential provider to issue a user an anonymous credential on various attributes. The user can then prove to a third party that she possesses valid credentials issued by that provider, yet without revealing further information about credentials and attributes. Note, however, that AC proofs disclose (some) information about credential issuers.

3.2 Protocols for Oblivious Information Transfer

As discussed in Section 1.2.1, several cryptographic constructs enable oblivious transfer of information between two entities. We review them below.

Oblivious Transfer (OT) [136]. The need for an *Oblivious Transfer* (OT) mechanism was first pointed out by Rabin [136]. The classic OT formulation involves a sender with n secret messages and a receiver with one index (i). The receiver wants to retrieve the i-th among sender's messages (and nothing else), without the sender learning i. Several OT constructs has been proposed [56, 25, 123, 124, 29]. OT is also a fundamental tool of Public-Key Cryptography, as proven by Killian [107].

Private Information Retrieval (PIR) [40]. PIR enables a client to retrieve an item from a server (public) database without revealing which item it is retrieving, with the additional requirement that communication overhead must be strictly lower than linear in the database size. PIR techniques follow two possible approaches: they either employ data replication and assume multiple non-cooperating servers [40, 11, 39, 131], or they use a single computationally-bounded server [109, 73, 114]. In PIR, privacy of server's database is not ensured, i.e., the client might receive records (or part of them) beyond those requested. In Symmetric-PIR (SPIR) [74], the server releases to the client exactly one data item per query, thus realizing OT with communication overhead lower than linear. Similar to OT, PIR clients need to know and input the index of the desired item in server's database. An extension enabling retrieval by keywords is Keyword-PIR (KPIR) [39, 131]. For more details on PIR, we refer to [132, 115].

3.3 Private Set Intersection

An important tool for privacy-preserving sharing of sensitive information is *Private Set Intersection* (PSI). This section reviews prior work on PSI. We start with the general formulation and then consider two variants: Authorized Private Set Intersection (APSI) and Size-Hiding Private Set Intersection (SHI-PSI).

3.3.1 Available PSI Protocols

PSI is a protocol involving a server and a client, on inputs $\mathcal{S} = \{s_1, \ldots, s_w\}$ and $\mathcal{C} = \{c_1, \ldots, c_v\}$, respectively, that results in the client obtaining $\mathcal{S} \cap \mathcal{C}$. As a result of running PSI, set sizes are reciprocally disclosed to both server and client. In the variant called *PSI with Data Transfer (PSI-DT)*, each item in server set has an associated data record, i.e., server's input is $\mathcal{S} = \{(s_1, data_1), \cdots, (s_w, data_w)\}$, and client's output is defined as $\{(s_j, data_j) \in \mathcal{S} \mid \exists c_i \in \mathcal{C}\ s.t.\ c_i = s_j\}$.

There are two classes of PSI protocols: one based on Oblivious Polynomial Evaluations (OPE) [125], and the other based on Oblivious Pseudo-Random Functions (OPRF-s) [65].

Freedman, Nissim, and Pinkas [66] introduce the concept of Private Set Intersection and and propose a protocol based on OPE. They represent a set as a polynomial, and elements of the set as its roots. A client encodes elements in its private set $\mathcal{C}$ as the roots of a v-degree polynomial over a ring R, i.e., $f = \prod_{i=1}^{v}(x - c_i) = \sum_{i=0}^{k} \alpha_i x^i$. Then, assuming pk_C is client's public key for any additively homomorphic cryptosystem (such as Paillier's [133]), the client encrypts the coefficients with pk_C, and sends them to server. The latter homomorphically evaluates f at each $s_j \in \mathcal{S}$. Note that $f(s_j) = 0$ if and only if $s_j \in \mathcal{C} \cap \mathcal{S}$. For each $s_j \in \mathcal{S}$, returns $u_j = E(r_j f(s_j) + s_j)$ to the client (where r_j is chosen at random and $E(\cdot)$ denotes additively homomorphic encryption under pk_C). If $s_j \in \mathcal{C} \cap \mathcal{S}$ then the client learns s_j upon decrypting. If $s_j \notin \mathcal{C} \cap \mathcal{S}$ then u_j decrypts to a random value. To enable data transfer, the server can return $E(r_j f(s_j) + (s_j || data_j))$, for each s_j in its private set $\mathcal{S}$. The protocol in [66] incurs the following complexities: The number of server operations depends on the evaluation of client's encrypted polynomial with v coefficients on w points (in $\mathcal{S}$). Using Paillier cryptosystem [133] and a 1024-bit modulus, this costs $O(vw)$ of 1024-bit mod 2048-bit exponentiations.[1] On the other hand, client computes $O(v + w)$ of 1024-bit mod 2048-bit exponentiations. However, server computation can be reduced to $O(w \log \log v)$ using: (1) Horner's rule for polynomial evaluations, and (2) a hashing-to-bins method (see [66] for more details). If one does not need *data transfer*, it is more efficient to use the Exponential ElGamal cryptosystem [54] (i.e., an ElGamal variant that provides additively homomorphism).[2] Such a cryptosystem does not provide efficient decryption, however, it allows client to test whether a ciphertext is an encryption of "0", thus, to learn that the corresponding element belongs to the set intersection. As a result, efficiency is improved, since in ElGamal the computation may make use of: (1) very short random exponents (e.g., 160-bit) and (2) shorter moduli in exponentiations (1024-bit). The PSI protocol in [66] is secure against honest-but-curious adversaries in the standard model, and can be extended to malicious adversaries in the Random Oracle Model (ROM), at an increased cost.

Hazay and Nissim [87] present an improved construction of [66], in the presence of malicious adversaries without ROM, using zero-knowledge proofs to let client demonstrate that encrypted polynomials are correctly produced. Perfectly hiding commitments, along with an Oblivious Pseudo-Random Function evaluation protocol, are used to prevent server from deviating from the protocol. The protocol in [87] incurs $O(v + w(\log \log v + m))$ computational and $O(v + w \cdot m)$ communication complexity, where m is the number of bits needed to represent a set element.

Kissner and Song [108] also propose OPE-based protocols involving (potentially) more than two players. They present one technique secure in the standard model against semi-honest and one – against malicious adversaries. The former incurs quadratic – $O(vw)$ – computation (but linear communication) overhead. The latter uses expensive generic zero-knowledge proofs to

[1]Encryption and decryption in the Paillier cryptosystem [133] involve exponentiations mod n^2: if $|n| = 1024$ bits, then $|n^2| = 2048$ bits (where n is the public modulus). For more details, see Section 2.2.

[2]In the Exponential ElGamal variant, encryption of message m is computed as $E_{g,y}(m) = (g^r, y^r \cdot g^m)$ instead of $(g^r, m \cdot y^r)$, for random r and public key y.

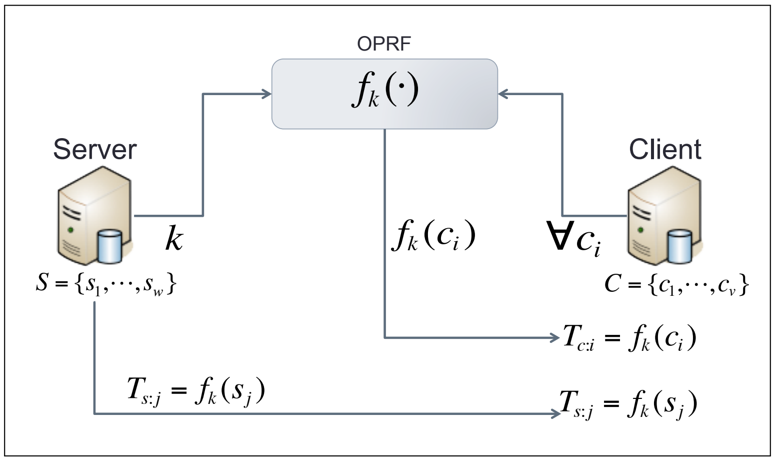


Figure 3.1: High-level view of Private Set Intersection protocols based on Oblivious Pseudo-Random Functions.

prevent parties from deviating to the protocol. Also, it is not clear how to enable *data transfer*.

Dachman-Soled, et al. [44] also present an OPE-based PSI construction, improving on [108]. Their protocol incorporates a secret sharing of polynomial inputs: specifically, as Shamir's secret sharing [140] implies Reed-Solomon codes [138], generic (i.e., expensive) zero-knowledge proofs can be avoided. Complexity of resulting protocol amounts to $O(wk^2 \log^2(v))$ in communication and $O(wvk \log(v) + wk^2 \log^2(v))$ in computation, where k is a security parameter.

Other techniques rely on *Oblivious Pseudo-Random Functions* (OPRF-s), introduced in [65]. An OPRF is a two-party protocol that securely computes a pseudo-random function $f_k(\cdot)$ on key k contributed by the sender and input x contributed by the receiver, such that the former learns nothing from the interaction and the latter learns only the value $f_k(x)$. Most prominent OPRF-based protocols are presented below. The intuition behind OPRF-based PSI protocols is illustrated in Figure 3.1: server and client interact in v parallel execution of the OPRF $f_k(\cdot)$, on input k and $c_i, \forall\ c_i \in \mathcal{C}$, respectively. As the server transfers $T_{s:j} = f_k(s_j), \forall\ s_j \in \mathcal{S}$ and the client obtains $T_{c:i} = f_k(c_i), \forall\ c_i \in \mathcal{C}$, the client learns the set intersection by finding matching $(T_{s:j}, T_{c:i})$ pairs, while it learns nothing about values $s_l \in \mathcal{S} \setminus \mathcal{S} \cap \mathcal{C}$, since $f_k(s_l)$ is indistinguishable from random, if $f_k(\cdot)$ is a pseudo-random function.[3]

Hazay and Lindell [85] propose the first PSI construction based on OPRF-s. In it, the server generates a secret random key k, then, for each $s_j \in \mathcal{S}$, computes $u_j = f_k(s_j)$, and sends the client the set $\mathcal{U} = \{u_1, \cdots, u_w\}$. Next, client and server engage in an OPRF computation of

[3]For more details on pseudo-random functions, we refer to [104, 78].

$f_k(c_i)$ for each $c_i \in \mathcal{C}$. Finally, the client learns that $c_i \in \mathcal{C} \cap \mathcal{S}$ if (and only if) $f_k(c_i) \in \mathcal{U}$. [85] introduces two constructions: one secure in the presence of malicious adversaries with one-sided simulatability, the other – in the presence of covert adversaries [6].

Jarecki and Liu [98] improve on [85] by constructing a protocol secure in the standard model against both malicious parties, based on the Decisional q-Diffie-Hellman Inversion assumption, in the Common Reference String (CRS) model, where a safe RSA modulus must be pre-generated by a trusted party. The OPRF in [98] is built using the Camenisch-Shoup additively homomorphic cryptosystem [30] (CS for short). However, this technique can be optimized, leading to the work by Belenkiy, et al. [12]. In fact, the OPRF construction could work in groups of 160-bit prime order, unrelated to the RSA modulus, instead of (more expensive) composite order groups [98]. Thus improved, the protocol in [98] incurs the following computational complexity: the server needs to perform $O(w)$ PRF evaluations, specifically, $O(w)$ modular exponentiations of m-bit exponents mod n^2, where m the number of bits needed to represent set items and n^2 is typically 2048-bit long. The client needs to compute $O(v)$ CS encryptions, i.e., $O(v)$ m-bit exponentiations mod 2048 bits, plus $O(v)$ 1024-bit exponentiations mod 1024 bits. The server also computes $O(v)$ 1024-bit exponentiations mod 1024 bits and $O(v)$ CS decryptions – i.e., $O(v)$ 1024-bit exponentiations mod 2048 bits. Complexity in malicious model grows by a factor of 2. The input domain size of the pseudo-random function in [98] is limited to be polynomial in the security parameter, since the security proof requires the ability to exhaustively search over input domain.

Jarecki and Liu [100] also propose a PSI protocol based on a related concept – *Unpredictable Functions* (UPFs). One specific UPF, $f_k(x) = H(x)^k$, is used as a basis for two-party computation (in ROM), with the server contributing the key k and the client – the argument x. The client picks a random exponent α and sends $y = H(x)^\alpha$ to the server, that replies with $z = y^k$, so that the client recovers $f_k(x) = z^{1/\alpha}$. This is similar to techniques proposed in [94] and [57]. Similar to OPRFs, the UPF can be used to implement secure computation of (Adaptive) Set Intersection, under the *One-More-Gap-DH* assumption in ROM [14]. The resulting protocol is, however, remarkably faster: random exponents can be taken from a subgroup. Therefore, the computational complexity of the UPF-based PSI in [100] amounts to $O(w + v)$ exponentiations with short exponents at server side and $O(v)$ at client side (e.g., 160-bit mod 1024-bit). Communication complexity is also linear is input set size, i.e., $O(w + v)$.

In summary, prior work has yielded a number of PSI techniques. However, as usually happens, the next step is to improve their efficiency. We identify the need for linear-complexity PSI constructions, entailing fast cryptographic operations (e.g., using short exponents), and relying only on standard computational assumptions (e.g., without using assumptions of the *One-More* type), in the presence of both semi-honest and malicious adversaries. Also, PSI interactions may often involve players with relatively unbalanced computational power, e.g., a client might be represented by a device with limited resources, such as smartphones.

3.3.2 Authorized Private Set Intersection

We now review work resembling APSI, which we will define in Chapter 5.

Recall from Section 1.2.2 that in PSI the client learns the set intersection while the server learns nothing: this might threaten server's privacy if the client maliciously populate its input set with its best guesses of the server set (especially, if the set is easy to exhaustively enumerate). In the extreme case, the client could even claim that its set contain all possible elements. We claim that this issue cannot be effectively addressed without some mechanism to *authorize* client inputs. The intuition behind APSI is that client's input items need to be certified (i.e., authorized) by an appropriate (offline) trusted authority, in such a way that the client has access to only duly authorized items. The challenge is to do so without divulging to the server any information about client inputs or authorizations.

We will define APSI as follows: it is a protocol involving a server and a client, on input, respectively, $\mathcal{S} = \{s_1, \ldots, s_w\}$ and $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$. It results in the client obtaining $\{s_j \in \mathcal{S} \mid \exists (c_i, \sigma_i) \in \mathcal{C} \text{ s.t. } c_i = s_j \wedge \sigma_i \text{ is valid authorization on } c_i\}$. A very similar functionality can also be realized from *Privacy-preserving Policy-based Information Transfer* (PPIT), presented in Chapter 4.

One related concept is *Authorised Private Searches on Public-key Encrypted Data* [27]. In it, a server encrypts records and associated keywords using an Identity-Based Encryption (IBE) scheme [21]. A client can search for a given keyword only if it has a corresponding trapdoor, issued by a TTP. In doing so, (1) server learns nothing about client's trapdoors, and (2) client learns nothing about keywords not matching its searches. Note, however, that the testing algorithm in [27] requires the client to test each trapdoor against each encrypted keyword it receives, thus, incurring a quadratic overhead. Furthermore, [27] is built on top of the Boyen-Waters IBE scheme [22]. In it, encryption requires 6 exponentiations and takes 6 group elements, while decryption requires 5 bilinear map operations. As a result, the efficiency of this scheme quickly becomes impractical for increasing input sizes.

Also related to APSI is the technique in [31], that allows a TTP to ensure that all protocol inputs are valid and bound to each protocol participant. The proposed protocol is *mutual* (i.e., both parties receive the intersection) and incurs quadratic computation and communication overhead (similar to the PSI protocol on which it is based, i.e., [108]).

3.3.3 Size-Hiding Protocols

As discussed in Section 1.2.2, there is no available PSI construct that hides the size of participants' inputs – an important requirement in some realistic scenarios. In Chapter 7, we introduce the first Size-Hiding Private Set Intersection (SHI-PSI).

Note that even generic techniques for secure two-party computation (e.g., [79, 148], discussed in 1.2.1) reveal the sizes of both parties inputs.

Ishai and Paskin [95] consider privacy in branching programs. Given a branching program P (held by a server) and encryption c of message x (held by a client), the technique in [95] computes ciphertext c' from which $P(x)$ can be decoded (using the corresponding secret key). Size of c' depends, polynomially, on sizes of x and P. Thus, neither client computation nor communication overhead depends on server input size P, that remains secret to client. Although one could theoretically attempt to implement PSI with a branching program and hide *server* input size, we argue that this generic construction would involve a high computational overhead – polynomial in the size of inputs.

Some work focuses on secure computation of *pattern matching* [85, 71, 105, 88], where a client holds a pattern and a server holds an arbitrarily-long text string. The goal of the client is to learn where the pattern appears in the text, without revealing it to the server or learning anything else about server's input. However, the size of P_1's pattern is always revealed to P_2. [88] sketches a possible way to hide pattern size, however, only by means of random padding. As we will discuss later in Chapter 7, this is approach exposes the upper bound. It also imposes a substantial performance penalty, as protocol complexity increases from linear to quadratic.

Finally, the need for hiding input sizes is discussed in [120]. A server publishes a short snapshot of its private database, i.e., a commitment. Later, a client can request the server to prove whether a given item, x, belongs to the committed set. Neither the commitment nor the proof reveals the size of server database. However, the problem addressed in [120] is quite different from (size-hiding) PSI.

3.3.4 Additional Constructs

Secret Handshakes (SH-s) slightly resemble APSI as they can be viewed as a symmetric set intersection protocol (with authorization) where the set is of size one. Some Secret Handshakes, however, are bi-directional authentication protocols. Thus, they are not directly applicable to one-way (client-to-server) authentication scenarios. Other related constructs, such as Hidden Credentials (HC-s) and Oblivious Signature-Based Envelopes (OSBE-s) provide uni-directional primitives. However, as discussed in the next chapter, it is not clear how to adapt them to APSI.

Also, PSI shares some features with Private Information Retrieval (PIR), as they both allow a client to privately retrieve information from a server. However, in PIR, the server is willing to release any of its data to the client. Furthermore, Symmetric-PIR (SPIR) additionally protects server's privacy, however, the client needs to input the index of the desired item in server's database (unlike PSI). Finally, Keyword-PIR (KPIR) does not consider server privacy. It also involves multiple rounds of PIR executions, and requires multiple non-cooperating servers [131]. It is also not clear how to adapt PIR techniques to ensure that the client is authorized to retrieve the requested item.

Finally, generic secure computation techniques could also be used to realize PSI, e.g., by means of Yao's garbled circuits [148]. However, such techniques are notoriously inefficient, since the size of the circuit would at least be quadratic in the size of players' inputs.

Chapter 4

Transferring Confidential Information based on Implicit Signature Verification

In this chapter, we motivate and introduce the concept of Privacy-preserving Policy-based Information Transfer (PPIT). PPIT combines mechanisms for Implicit Authentication and Oblivious Information Transfer. After formalizing PPIT functionality and its security model, we present three efficient instantiations obtained, respectively, from RSA signatures, Schnorr signatures, and Identity-based Encryption.

4.1 Introduction & Motivation

There are many scenarios where sensitive information is requested by some authority due to some legitimate need. The challenge for the information owner is to allow access to only duly authorized information, whereas, the challenge for the information requester is to obtain needed information without divulging what is being requested.

Consider the following example. University of Lower Vermont (ULoVe) is confronted with an FBI investigation focused on one of its faculty members (Alice). The University is understandably reluctant to allow FBI unlimited access to its employee records. For its part, FBI is unwilling to disclose that Alice is the target of investigation. There might be several reasons for FBI's stance: (1) Concern about unwarranted rumors and tarnishing Alice's reputation, e.g., leaked information might cause legal action and result in bad PR for the FBI; (2) The need to keep the investigation secret, i.e., preventing malicious insiders (ULoVe employees) from forewarning Alice about the investigation. Ultimately, ULoVe must comply with FBI's demands, especially, if the latter is armed with appropriate authorization (e.g., a court order) from, e.g., the US Attorney General's office. However, the authorization presumably applies only to Alice. Assuming all communication between ULoVe and FBI is electronic, there seems to be an impasse. An additional nuance is that, even if ULoVe is willing to provide FBI unrestricted access to all its employee records, FBI may not want the associated liability. This is because mere possession of ULoVe sensitive employee information would require FBI to demonstrate that the information is/was treated appropriately and disposed of when no longer needed. Considering a number of recent incidents of massive losses of sensitive government and commercial employees' records [33], FBI might be unwilling to assume additional risk.

In general, we consider the need to transfer information (or, more generally, perform some data-centric task) between two parties who are willing and/or obligated to transfer information in an accountable and policy-guided (*authorized*) manner. Therefore, the main technical challenge is how to enable the information owner to efficiently and obliviously compute proper authorization decisions, while: (1) preserving privacy of its data, and (2) preserving privacy of requester's authorizations.

To this end, this chapter introduces and formalizes the concept of Privacy-preserving Policy-based Information Transfer (PPIT). PPIT considers the following setting, involving an information owner (*server*), a requester (*client*), and an authorization authority (*CA*). The server holds a database of records in the form $(\mathsf{ID}, \mathsf{D})$: ID denotes a unique record identifier and D the associated information. The client is interested in acquiring a specific record, e.g., that identified by the string ID^*. In order to do so, it needs to obtain an appropriate authorization from CA. PPIT ensures that the client attains information pertaining ID^*, while: (1) the server learns nothing about client's interests or authorizations, and (2) the client learns nothing about any server's record unless it is duly authorized.

Note that PPIT makes no assumption on the format of database records or their identifiers. For instance, records can be strings, database entries, files, or even binary data.

4.2 Preliminaries

This section introduces the PPIT primitive, including: players, components, and security definitions.

4.2.1 Players

PPIT involves three entities: server, client, and CA:

- **Server** – stores a list of records, $\mathcal{I} = \big\{(\mathsf{ID}_{s:j}, \mathsf{D}_j) | \mathsf{ID}_{s:j} \in \{0,1\}^l\big\}_{j=1}^{w}$, where each $\mathsf{ID}_{s:j}$ is a l-bit string that uniquely identifies a record and D_j denotes the associated information (with arbitrary length).
- **Client** – holds a pair (ID^*, σ), where ID^* is an l-bit unique identifier and σ is an authorization for ID^*.
- **CA** – is an off-line trusted third party that authorizes clients to access specific records.

4.2.2 PPIT Algorithms

PPIT is composed of three algorithms: $(\mathsf{Setup}, \mathsf{Authorize}, \mathsf{Transfer})$:

- $\mathsf{Setup}(1^\tau)$: Given a security parameter τ, CA, after selecting an appropriate digital signature scheme, $\mathsf{DSIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$, generates a key-pair (sk,pk), via KGen, and publishes pk.
- $\mathsf{Authorize}(\mathsf{sk}, \mathsf{ID}^*)$: CA issues an authorization σ on a given identifier ID^*, contributed by the client, where $\sigma = \mathsf{Sign}_{\mathsf{sk}}(\mathsf{ID}^*)$. For each invocation of $(\sigma = \mathsf{Authorize}(\mathsf{sk}, \mathsf{ID}^*))$, $\mathsf{Vrfy}(\mathsf{pk}, \mathsf{ID}^*, \sigma) = 1$.
- $\mathsf{Transfer}$: Server and client interact on public input pk, on server's private input $\mathcal{I} = \big\{(\mathsf{ID}_{s:j}, \mathsf{D}_j) | \mathsf{ID}_{s:j} \in \{0,1\}^l\big\}_{i=1}^{w}$ and client's private input (σ, ID^*). At the end of $\mathsf{Transfer}$, server has no output and client outputs:

$$\{(\mathsf{ID}_{s:j}, \mathsf{D}_j) \in I \mid \exists j\ s.t.\ \mathsf{ID}_{s:j} = \mathsf{ID}^* \text{ and } \mathsf{Vrfy}(\mathsf{ID}^*, \sigma) = 1\}.$$

4.2.3 Security & Privacy Requirements

PPIT must satisfy the following security and privacy requirements.

Correctness. A PPIT scheme is *correct* if, at the end of $\mathsf{Transfer}$, the client outputs D, given that:

(1) $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\tau)$ and $\sigma = \mathsf{Authorize}(\mathsf{ID}^*)$,

(2) Server and client run $\mathsf{Transfer}$ on input $(\mathsf{ID}^*, \mathsf{D})$ and (ID, σ), respectively.

Security. PPIT security guarantees that only clients authorized to access data D can learn any information about D. Formally, we say that a PPIT scheme is *secure* if any polynomially bounded adversary $\mathcal{A}$ cannot win the following game, with probability non-negligibly over 1/2. The game is between $\mathcal{A}$ and a challenger Ch:

1. Ch runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\tau)$.

2. $\mathcal{A}$, on input pk, adaptively queries Ch a polynomial number q of times on a set of strings $Q = \{\mathsf{ID}_i | \mathsf{ID}_i \in \{0,1\}^l\}_{i=1}^q$. For every ID_i, Ch responds by giving $\mathcal{A}$ a signature $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{ID}_i)$.

3. $\mathcal{A}$ announces a new identifier string, $\mathsf{ID}^* \notin Q$, and generates two equal-length data record $(\mathsf{D}_0{}^*, \mathsf{D}_1{}^*)$.

4. Ch picks one record by selecting a random bit $b \leftarrow_r \{0,1\}$, and executes server's part of $\mathsf{Transfer}$ on public input pk and private inputs $(\mathsf{ID}^*, \mathsf{D}_b^*)$.

5. $\mathcal{A}$ outputs b' (and wins if $b' = b$).

Server Privacy. While the previous definition captures privacy of server's *data*, we now focus on privacy of server's *identifiers*. A PPIT scheme allows only authorized clients to learn any information about the ID-s inputted by server in the interaction with the client. Decoupling server privacy from server security is needed to capture two different problems. In fact, server privacy is not required when identifiers are public. For instance, in the University scenario discussed earlier in Section 4.1, the list of ULoVe employees (and thus their identifiers) might be public. Formally, we say that a PPIT scheme is *server-private* if no polynomially bounded adversary $\mathcal{A}$ can win the following game with probability non-negligibly higher than 1/2. The game proceeds between $\mathcal{A}$ and Ch:

1. Ch runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\tau)$.

2. $\mathcal{A}$, on input pk, adaptively queries Ch a number q of times on a set of strings $Q = \{\mathsf{ID}_i | \mathsf{ID}_i \in \{0,1\}^l\}_{i=1}^q$. For every ID_i, Ch responds by giving $\mathcal{A}$ a signature $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{ID}_i)$.

3. $\mathcal{A}$ announces two new identifier strings, $(\mathsf{ID}_0{}^*, \mathsf{ID}_1{}^*) \notin Q$, and generates a data record D^*.

4. Ch picks one identifier by selecting a random bit $b \leftarrow_r \{0,1\}$, and executes server's part of $\mathsf{Transfer}$ on public input pk and private inputs $(\mathsf{ID}_b{}^*, \mathsf{D}^*)$.

5. $\mathcal{A}$ outputs b' (and wins if $b' = b$).

Security and server privacy games could be merged into one. In fact, it is possible to modify $\mathcal{A}$ to announce two pairs $(\mathsf{ID_0}^*, \mathsf{D_0}^*), (\mathsf{ID_1}^*, \mathsf{D_1}^*)$ and let Ch pick a random bit b and execute server's part of $\mathsf{Transfer}$ on input $(\mathsf{ID}_b{}^*, \mathsf{D}_b{}^*)$. The security property alone is obtained by restricting $\mathcal{A}$'s challenge query so that $(\mathsf{ID_0}^* = \mathsf{ID_1}^*)$, while server privacy alone is obtained if $(\mathsf{D_0}^* = \mathsf{D_1}^*)$.

Client Privacy. Client privacy guarantees that no information is leaked about client's input to a malicious server. Formally, a PPIT scheme is *client-private* if no polynomially bounded adversary $\mathcal{A}$ can win the following game with the probability non-negligibly over 1/2. The game is between $\mathcal{A}$ and Ch:

1. Ch executes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\tau)$.

2. $\mathcal{A}$, on input sk, chooses two strings $\mathsf{ID_0}^*$, $\mathsf{ID_1}^*$ and two strings $\sigma_0{}^*, \sigma_1{}^*$.

3. Ch picks a random bit $b \leftarrow_r \{0, 1\}$ and interacts with $\mathcal{A}$ by following $\mathsf{Transfer}$ on behalf of client on public input pk and private inputs $(\mathsf{ID}_b{}^*, \sigma_b{}^*)$.

4. $\mathcal{A}$ outputs b' (and wins if $b' = b$).

4.3 RSA-PPIT

PPIT Intuition. The main idea behind PPIT is the following. The server and the client engage in a cryptographic protocol and *conditionally* agree on a shared key, used to establish a session encryption key (à la Diffie-Hellman). The necessary condition upon key establishment is an *implicit* verification on client's possession of a digital signature. In other words, the server, for each record, computes a key by obliviously verifying client's signature; on the other hand, the client extracts the same key (from the protocol) only if it holds a valid signature (issued by CA) for the corresponding record.

We now present our first PPIT instantiation, i.e., RSA-PPIT – based on RSA signatures.

Setup. On input of security parameter τ, CA generates a safe RSA modulus $N = pq$, i.e., $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' are primes. The algorithm picks a random element g generator of QR_N. RSA exponents (e, d) are chosen in the standard way. The secret key is

$\mathsf{sk} = (p, q, d)$ and the public key $\mathsf{pk} = (N, g, e)$. The algorithm also fixes a full-domain hash function $H_1 : \{0,1\}^* \to \mathbb{Z}_N$, and two other cryptographic hash functions $H_2 : \{0,1\}^* \to \{0,1\}^\tau$, $H_3 : \{0,1\}^* \to \{0,1\}^\tau$.

Authorize. To issue an authorization on ID^* to a client, CA computes an RSA signature on ID^*, $\sigma = H_1(\mathsf{ID}^*)^d \bmod N$. The signature on ID^* can be verified by checking if $\sigma^e \bmod N = H_1(\mathsf{ID}^*)$.

Transfer. This protocol is between a client a the server, where public input is $\mathsf{pk} = (N, e, g)$, and client's private input is (ID^*, σ), where $\sigma^e = H_1(\mathsf{ID}^*) \bmod N$, and server's private input is $\mathcal{I} = \left\{(\mathsf{ID}_{s:j}, \mathsf{D}_j) | \mathsf{ID}_{s:j} \in \{0,1\}^l\right\}_{i=1}^{w}$. The resulting protocol is illustrated in Figure 4.1. Proofs appear in Section 4.7.

[Common input: $N, e, g, H_1(\cdot), H_2(\cdot), H_3(\cdot)$]

Client, on input: (ID^*, σ), where:
$\sigma = H_1(\mathsf{ID}^*)^d$

Server, on input: $\mathcal{I}$, where:
$\mathcal{I} = \{(\mathsf{ID}_{s:1}, \mathsf{D}_1), \ldots, (\mathsf{ID}_{s:w}, \mathsf{D}_w)\}$

Client: $R_c \leftarrow_r \mathbb{Z}_{N/4},\ \mu = \sigma^2 \cdot g^{R_c}$

$$\xrightarrow{\mu}$$

Server:

If $\mu \notin \mathbb{Z}_N^*$ then abort
$R_s \leftarrow_r \mathbb{Z}_{N/4}$, $Z = g^{eR_s}$
For $j = 1, \ldots, w$:
  $K_{s:j} = (\mu)^{eR_s} \cdot H_1(\mathsf{ID}_j)^{-2R_s}$
  $T_{s:j} = H_2(K_{s:j}),\ k_{s:j} = H_3(K_{s:j})$
  $CT_{s:j} = \mathsf{Enc}_{k_{s:j}}(\mathsf{D}_j)$

$$\xleftarrow[\{CT_{s:1}, \ldots, CT_{s:w}\}]{Z, \{T_{s:1}, \ldots, T_{s:w}\}}$$

Client:

$K_c = Z^{R_c}$
$T_c = H_2(K_c), k_c = H_3(K_c)$
If $\exists\, T_{s:j}$ s.t. $T_{s:j} = T_c$, then
  $\mathsf{D}^* = \mathsf{Dec}_{k_c}(CT_{s:j})$
  Output: $(\mathsf{ID}^*, \mathsf{D}^*)$

[All computation is $\bmod N$]

Figure 4.1: Our RSA-PPIT instantiation.

To see that RSA-PPIT is *correct*, observe that, if $\mathsf{ID}^* = \mathsf{ID}_{s:j}$, then:

$$\begin{aligned} K_{s:j} =& (\mu)^{eR_s} \cdot H_1(\mathsf{ID}_{s:j})^{-2R_s} = (H_1(\mathsf{ID}^*)^{2d} \cdot g^{R_c})^{eR_s} \cdot H_1(\mathsf{ID}_{s:j})^{-2R_s} = \\ =& H_1(\mathsf{ID}^*)^{2R_s} \cdot g^{eR_cR_s} \cdot H_1(\mathsf{ID}_{s:j})^{-2R_s} = g^{eR_sR_c} = Z^{R_c} = K_c \end{aligned} \tag{4.1}$$

Thus, $T_{s:j} = H_2(K_{s:j}) = H_2(K_c) = T_{c:i}$ (similarly $K_{s:j} = K_c$).

Protocol Complexity. The protocol in Figure 4.1 incurs the following complexities. Server overhead is dominated by $O(w)$ modular exponentiations (in the RSA group), whereas, client computation amounts to $O(1)$. Communication overhead is dominated by server's response, i.e., w ciphertexts and hash values.

We remark that this construction is loosely based on RSA-OSBE from [111] (overviewed in Section 3.1). Recall, however, that OSBE targets the transmission of a single message, while PPIT applies to scenarios where the server holds multiple (w) records. Nonetheless, one may try to adapt RSA-OSBE to the PPIT scenario by running w batched invocations of RSA-OSBE. This would incur the same asymptotic complexity (i.e., linear), however, the client would be forced to perform $O(w)$ decryptions – a significant overhead in scenarios where records are long (e.g., in the order of megabytes).

Whereas, our construction minimizes client computation using a ***tagging*** technique that lets the server label each ciphertext with a unique tag. In RSA-PPIT, each tag $T_{s:j}$ is the output of a one-way function (in practice, a cryptographic hash function modeled as a random oracle) computed over a Diffie-Hellman key $K_{s:j}$. As discussed above, if (and only if) the client holds and inputs a signature on the corresponding $\mathsf{ID}_{s:j}$, it reconstructs the Diffie-Hellman key and can *match* the tag. This way, the client does not need to try decrypting *all* ciphertexts.

Also note that the client sends the server ($\mu = \sigma^2 \cdot g^{R_c}$), whereas, in RSA-OSBE, the client would send ($\sigma \cdot H_1(\mathsf{ID})^{R_c}$). We square σ to guarantee that μ is in QR_N, a crucial detail in our proofs that has been overlooked [111]. The use of g will allow the client to ***batch*** computation in case it has multiple inputs, as we show in Section 4.6.2.

4.4 Schnorr-PPIT

In this section, we present another PPIT instantiation, based on Schnorr signatures (see Section 2.2).

Setup. On input of a security parameter τ, CA generates a Schnorr key: (p, q, g, a, y), where p, q are primes, s.t. q divides $p - 1$ but q^2 does not divide $p - 1$, g is a generator of a subgroup in $\mathbb{Z}_p^*$ of order q, a is picked randomly in $\mathbb{Z}_q^*$, and $y = g^a \bmod p$. The public key is $\mathsf{pk} = (p, q, g, y)$ and the secret key is $\mathsf{sk} = a$. The algorithm also defines hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_q^*$, $H_2 : \{0,1\}^* \to \{0,1\}^\tau$, $H_3 : \{0,1\}^* \to \{0,1\}^\tau$.

Authorize. To issue authorization on a given string ID^*, CA computes a Schnorr signature $\sigma = (X, s)$ where $X = g^k \bmod p$ and $s = k + a \cdot H_1(\mathsf{ID}^*, X) \bmod q$, for a random $k \in \mathbb{Z}_q^*$. The signature on ID^* is verified by checking whether $g^s = X \cdot y^{H_1(\mathsf{ID}^*, X)} \bmod p$.

Transfer. This protocol is between a client and a server, where public input is $\mathsf{pk} = (p, q, g, y)$, and client's private input is $(\mathsf{ID}^*, \sigma = (X, s))$, s.t. $g^s = X \cdot y^{H_1(\mathsf{ID}^*, X)} \bmod p$ and server's private input is $\mathcal{I} = \left\{(\mathsf{ID}_{s:j}, \mathsf{D}_j) | \mathsf{ID}_{s:j} \in \{0,1\}^l\right\}_{i=1}^{w}$. The resulting protocol is illustrated in Figure 4.2. Proofs appear in Section 4.7.

[Common input: $N, e, g, H_1(\cdot), H_2(\cdot), H_3(\cdot)$]

Client, on input: (ID^*, σ), where:
$\sigma = (X, s)$

Server, on input: $\mathcal{I}$, where:
$\mathcal{I} = \{(\mathsf{ID}_{s:1}, \mathsf{D}_1), \ldots, (\mathsf{ID}_{s:w}, \mathsf{D}_w)\}$

Client: $g^s = X \cdot y^{H_1(\mathsf{ID}^*, X)}$

$$\xrightarrow{X}$$

Server:
If $X^{(p-1)/q} \neq 1 \bmod p$ then abort.
$R_s \leftarrow_r \mathbb{Z}_q^*$, $Z = g^{R_s}$
For $j = 1, \ldots, w$:
  $K_{s:j} = (y^{H_1(\mathsf{ID}_{s:j}, X)} \cdot X)^{R_s}$
  $T_{s:j} = H_2(K_{s:j}), \; k_{s:j} = H_3(K_{s:j})$
  $CT_{s:j} = \mathsf{Enc}_{k_{s:j}}(\mathsf{D}_j)$

$$\xleftarrow[\{CT_{s:1}, \ldots, CT_{s:w}\}]{Z, \{T_{s:1}, \ldots, T_{s:w}\}}$$

Client:
$K_c = Z^s$
$T_c = H_2(K_c)$, $k_c = H_3(K_c)$
If $\exists\, T_{s:j}$ s.t. $T_{s:j} = T_c$, then
  $\mathsf{D}^* = \mathsf{Dec}_{k_c}(CT_{s:j})$
  Output: $(\mathsf{ID}^*, \mathsf{D}^*)$

[All computation is $\bmod\ p$]

Figure 4.2: Our Schnorr-PPIT instantiation.

To see that Schnorr-PPIT is *correct*, observe that, if $\mathsf{ID}^* = \mathsf{ID}_{s:j}$, then:

$$K_{s:j} = (y^{H_1(\mathsf{ID}_{s:j},X)} \cdot X)^{R_s} = (g^{aH_1(\mathsf{ID}_{s:j},X)} \cdot g^k)^{R_s} = (g^{aH_1(\mathsf{ID}^*,X)+k})^{R_s} = g^{sR_s} = K_c \quad (4.2)$$

Thus, $T_{s:j} = H_2(K_{s:j}) = H_2(K_c) = T_{c:i}$ (similarly $K_{s:j} = K_c$).

Protocol Complexity. Similar to its RSA counterpart, the protocol in Figure 4.2 incurs linear computation and communication complexity. Server overhead is dominated by $O(w)$ modular exponentiations in the Schnorr setting, i.e., using short exponents. Client computation amounts to $O(1)$, whereas, communication overhead is dominated by server's response, i.e., w ciphertexts and hash values.

Schnorr-PPIT protocol is loosely based on the Schnorr-OSBE construction in [127]. However, in order to minimize client computation, we add a tagging technique similar to RSA-PPIT. Also, proofs for Schnorr-PPIT differ substantially from those of Schnorr-OSBE, given the different nature of privacy requirements and players' inputs (one message in OSBE vs many records in PPIT).

4.5 IBE-PPIT

In this section, we show how to obtain a PPIT instantiation from any *Anonymous* Identity-Based Encryption (IBE) scheme.[1]

Recall that IBE is a public-key system where any string can be used as a valid public key. A trusted third party (Private Key Generator or PKG), holding a secret master key, can generate private keys corresponding to any public key, by signing the latter using the secret master key.

One-round PPIT can be instantiated using *any* Anonymous IBE scheme. CA acts as PKG and, during Setup, runs IBE setup algorithm to generate the KDC master key and global IBE system parameters. Then, during $\mathsf{Authorize}$, CA authorizes a client, on a given ID^*, by issuing the IBE private key corresponding to ID^*. Finally, during $\mathsf{Transfer}$, the server encrypts any D_j under the identifier string $\mathsf{ID}_{s:j}$: the client decrypts it if (and only if) it holds the IBE private key corresponding to $\mathsf{ID}_{s:j}$.

This is somehow similar to IBE-OSBE, explored in [111]. However, in IBE-OSBE, the use of anonymous IBE to achieve key-privacy (in the sense of [13]) is optional. Whereas, this is

[1]We refer to Section 2.2 for details on IBE. Anonymous-IBE [22] additionally requires that a computationally bounded adversary cannot infer any information, from only a ciphertext, about the public key string used to encrypt.

a fundamental requirement in our scheme: an adversary who correctly guesses the encryption key used to generate a ciphertext would immediately violate server privacy.

Realizing PPIT from any Anonymous IBE system would let the client perform a number of decryption linear in the number of server's records. We now show a PPIT instantiation that uses a specific IBE system (i.e., Boneh and Franklin IBE [21], introduced in Chapter 2.2) and reduces client's computation to $O(1)$, using a tagging technique. Proofs appear in Section 4.7.

Setup. On input of security parameter τ, CA generates a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Then a random $s \in \mathbb{Z}_q^*$, a random generator $P \in \mathbb{G}_1$, P are chosen and Q is set such that $Q = sP$. (P, Q) are public parameters, s is CA's private master key. Finally, three cryptographic hash function, $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^\tau$ are chosen.

Authorize. To issue an authorization on ID^* to a client, CA issues $\sigma = s \cdot H_1(\mathsf{ID}^*)$.

Transfer. This protocol is between a client a the server, where public input is (P, Q), and client's private input is (ID^*, σ), while server's is $\mathcal{I} = \left\{(\mathsf{ID}_{s:j}, \mathsf{D}_j) | \mathsf{ID}_{s:j} \in \{0,1\}^l\right\}_{i=1}^{w}$. The resulting protocol is illustrated in Figure 4.3.

[Common input: $P, Q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, H_1(\cdot), H_2(\cdot)$]

Client, on input: (ID^*, σ)

Server, on input: $\mathcal{I}$, where

$\mathcal{I} = \{(\mathsf{ID}_{s:1}, \mathsf{D}_1), \ldots, (\mathsf{ID}_{s:w}, \mathsf{D}_w)\}$

$z \leftarrow_r \mathbb{G}_1$, $Z = zP$

For $j = 1, \ldots, w$:

$K_{s:j} = \hat{e}(Q, H_1(\mathsf{ID}_{s:j}))^z$

$T_{s:j} = H_2(K_{s:j}),\ k_{s:j} = H_3(K_{s:j})$

$CT_{s:j} = \mathsf{Enc}_{k_{s:j}}(\mathsf{D}_j)$

$$\xleftarrow[\{CT_{s:1}, \ldots, CT_{s:w}\}]{Z, \{T_{s:1}, \ldots, T_{s:w}\}}$$

$K_c = \hat{e}(Z, \sigma)$

$T_c = H_2(K_c), k_c = H_3(K_c)$

If $\exists\, T_{s:j}$ s.t. $T_{s:j} = T_c$, then

$\mathsf{D}^* = \mathsf{Dec}_{k_c}(CT_{s:j})$

Output: $(\mathsf{ID}^*, \mathsf{D}^*)$

[All computation is $\bmod q$]

Figure 4.3: Our IBE-PPIT instantiation.

To see that IBE-PPIT is *correct*, observe that, if $\mathsf{ID}^* = \mathsf{ID}_{s:j}$, then:

$$\begin{aligned} K_{s:j} &= \hat{e}(Q, H_1(\mathsf{ID}_{s:j}))^z = \hat{e}(sP, H_1(\mathsf{ID}_{s:j}))^z = \hat{e}(zP, H_1(\mathsf{ID}_{s:j}))^s \\ &= \hat{e}(Z, s \cdot H_1(\mathsf{ID}_{s:j})) = \hat{e}(Z, \sigma) = K_c \end{aligned} \tag{4.3}$$

Thus, $T_{s:j} = H_2(K_{s:j}) = H_2(K_c) = T_{c:i}$ (similarly $K_{s:j} = K_c$).

We acknowledge, that re-use of randomness z for each tag in the IBE scheme is similar to [23]. However, our approach provides multi-encryption (i.e., encryption of different messages) instead of broadcast encryption [61]. Moreover, we embed the tags to reduce the number of decryptions to $O(1)$.

Protocol Complexity. The protocol in Figure 4.3 is one-round and incurs linear computation and communication complexity. The server performs linear computation: its overhead is dominated by $O(w)$ modular exponentiations and bilinear map pairings. Client computation is $O(1)$. Whereas, communication overhead amounts to w ciphertexts and hash values, transmitted from the server to the client.

4.6 Discussion

This section introduces additional (optional) security/privacy requirements for PPIT. It also discusses a PPIT extension where the client batches multiple authorizations into a single PPIT interaction. Finally, it presents a performance comparison of PPIT instantiations.

4.6.1 Unlinkability and Forward Security

We now introduce the concept of server/client unlinkability in PPIT, as well as forward security.

Server unlinkability: prevents a malicious client from guessing whether any two interactions (specifically, any two instances of the $\mathsf{Transfer}$ protocol) are related, thus, learning whether or not the server runs on the same inputs. Formally, we say that a PPIT scheme is *server-unlinkable* if no polynomially bounded adversary $\mathcal{A}$ can win the following game with probability non-negligibly higher than 1/2. The game proceeds between $\mathcal{A}$ and a challenger Ch:

1) Ch runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\tau)$.

2) $\mathcal{A}$, on input pk, adaptively queries Ch a number q of times on a set of strings $Q = \{\mathsf{ID}_i | \mathsf{ID}_{oj} \in \{0,1\}^l\}_{i=1}^q$. For every ID_i, Ch responds by giving $\mathcal{A}$ a signature $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{ID}_i)$.

3) $\mathcal{A}$ announces two new identifier strings, $(\mathsf{ID}_0{}^*, \mathsf{ID}_1{}^*) \notin Q$, and generates a data record D^*.

4a) Ch executes server's part of $\mathsf{Transfer}$ on public input pk and private inputs $(\mathsf{ID}_0{}^*, \mathsf{D}^*)$.

4b) Ch picks one identifier by selecting a random bit $b \leftarrow_r \{0,1\}$, and executes the server's part of $\mathsf{Transfer}$ on public input pk and private inputs $(\mathsf{ID}_b{}^*, \mathsf{D}^*)$.

5) $\mathcal{A}$ outputs b' (and wins if $b' = b$).

Client Unlinkability: prevents a server from learning whether any two interactions are related, and learning whether the client runs on the same input. If it is not guaranteed, the server may learn if the client is retrieving the same record or is holding the same CA authorization, over multiple interactions. Consider the FBI scenario discussed in Section 4.1: although client privacy prevents the University from learning the identity of the employee under investigation, the University could still infer that the *same* employee is under FBI investigation. The adversarial game for client unlinkability mirrors that for server unlinkability (with $\mathcal{A}$ and Ch playing inverted roles), thus, we omit it here.

Forward Security:[2] guarantees that:

1. An adversary who learns all of server's data (ID-s and records) cannot violate client privacy of past (recorded) $\mathsf{Transfer}$ interactions. (This is already captured through the notion of *client privacy*.)

2. An adversary who learns client's authorization(s) cannot violate security and server privacy of past (recorded) $\mathsf{Transfer}$ interactions.

We discuss whether our PPIT instantiations support additional privacy requirements discussed above:

- Unlike RSA-PPIT, Schnorr-PPIT does not offer *client unlinkability*, since the value $X = g^k$ sent by the client stays fixed for a given ID. Whereas, IBE-PPIT is trivially client-unlinkable, since no message is sent from the client to the server.

[2]We point out that our forward-security definitions here are only informal, while it is an interesting open problem how to provide formal definitions and proofs of forward security in the context of PPIT.

- All PPIT instantiations guarantee server unlinkability, given that server's randomness is generated anew for each $\mathsf{Transfer}$ execution (specifically, R_s in RSA- and Schnorr-PPIT, and z in IBE-PPIT).
- We argue that RSA-PPIT provides built-in forward security, while Schnorr-PPIT and IBE-PPIT schemes do not provide it.

4.6.2 Batched PPIT for Multiple Client Authorizations

Thus far, we have modeled PPIT as a functionality between a server, on input a set of records, and a client, on input one (alleged) authorization. Nonetheless, we now consider whether or not PPIT can be efficiently extended to support client with multiple authorizations. We now discuss a modified setting and sketch an extension for all PPIT instantiations.

The client runs on input

$$\Sigma = \left\{ (\mathsf{ID}_{c:i}, \sigma_i) \mid \mathsf{ID}_{c:i} \in \{0,1\}^l \wedge \mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{ID}_{c:i}, \sigma_i) = 1 \right\}_{i=1}^{v},$$

i.e., the set of v pairs defining a record identifier along with the corresponding authorization.

We claim that this extension has no effect on our security model or on our proofs, however, it might impact protocol efficiency. To ease presentation, we first discuss the PPIT variant for multiple authorizations using IBE-PPIT. Next, we focus on RSA- and Schnorr-PPIT.

IBE-PPIT. The protocol in Figure 4.3 can be adapted, almost with no modification, to the scenario with client's multiple authorizations. $\mathsf{Transfer}$ in IBE-PPIT is a one-round interaction where no information is sent from client to server. Record encryption keys, as well as the tags, do not depend on any input sent by the client. As a result, we only need to modify the protocol as follows: the client now computes, for each $(\mathsf{ID}_{c:i}, \sigma_i) \in \Sigma$, $K_{c:i} = \hat{e}(Z, \sigma_i)$, as well as $T_{c:i} = H_2(K_{c:i})$ and $k_{c:i} = H_3(K_{c:i})$ and decrypts all the records with a matching tag. Server computation and bandwidth utilization remains linear in the number of server records, i.e., $O(w)$, while client computation is linear in the number of authorizations, i.e., $O(v)$.

RSA-PPIT and Schnorr-PPIT. In all corresponding PPIT instantiations, the server computes a Diffie-Hellman key for each record. This key is computed based on (i) record identifiers, and (ii) client's message containing the blinded (alleged) authorization. Thus, if we extend to v authorizations, the client would send v messages, and the server would compute $(w \cdot v)$ Diffie-Hellman keys. For instance, in RSA-PPIT, the client would send $\mu_i = ({\sigma_i}^2 \cdot g^{R_{c:i}})$ for every $i = 1, \ldots, v$, and the server would compute $K_{s:ij} = ((\mu_i)^{eR_s} \cdot H_1(\mathsf{ID}_{s:j})^{-2R_s})$, for $i = 1, \ldots, v$ and $j = 1, \ldots, w$. This would result in to $O(w \cdot v)$ server computation overhead, as well as $O(w \cdot v)$ bandwidth utilization.

Clearly, quadratic number of modular exponentiations performed by the server prompts some scalability concerns. Therefore, we now propose a technique to reduce the number of exponentiations to linear, i.e., $O(w+v)$. We note that, in RSA-PPIT, the server can compute, *separately*: (i) $(\mu_i)^{eR_s}$, and (ii) $H_1(\mathsf{ID}_{s:j})^{-2R_s}$, thus, performing $(w + v)$ modular exponentiations and $(w \cdot v)$ modular multiplications. If pre-computation is allowed, the server can pre-compute $H_1(\mathsf{ID}_{s:j})^{-2R_s}$ exponentiations.

Whereas, to the best of our knowledge, it is not possible to reduce server computational overhead for Schnorr-PPIT.

4.6.3 Performance Analysis

We now compare performance of proposed PPIT schemes. We focus on the extended setting where the client holds multiple authorization. Table 4.1 summarizes, for each instantiation, the communication and (server/client) computation overhead, as well as the computational assumptions under which they are secure. We also recap their additional security features. Server and client computation is measured in terms of public-key operations, i.e., exponentiations in the case of RSA-PPIT and Schnorr-PPIT, and bilinear map operations in the case of IBE-PPIT. However, for server operations in RSA-PPIT, we distinguish between exponentiations and multiplications. Recall that w is the number of records stored by the server, and v – the number of authorizations held by the client.

IBE-PPIT is the most efficient by all counts, since it: (1) takes one round, (2) requires a linear number of public key operations for both server and client, and (3) consumes linear amount of bandwidth. Whereas, both Schnorr-PPIT and RSA-PPIT are two-round protocols. Schnorr-PPIT has quadratic – $O(w \cdot v)$ – computation and bandwidth overheads, while RSA-PPIT requires $O(w + v)$ exponentiations and $O(w \cdot v)$ multiplications on the server.

However, in the standard PPIT setting, or whenever the client runs on a small number of authorizations, Schnorr-PPIT and RSA-PPIT are faster because they use less expensive operations (modular exponentiations versus bilinear maps). Specifically, aiming at 80-bit security, the dominant cost factor varies with the scheme (in increasing order): (1) in Schnorr-PPIT, it is 160-bit exponentiations mod 1024-bit moduli, (2) in RSA-PPIT, it is 1024-bit exponentiations mod 1024-bit moduli, and (3) in IBE-PPIT, it is bilinear map pairings on 160-bit order groups.

	RSA-PPIT	**Schnorr-PPIT**	**IBE-PPIT**
Rounds	2	2	1
Server Computation	$O(w+v)$ exps $O(w \cdot v)$ mults	$O(w \cdot v)$	$O(w)$
Client Computation	$O(v)$	$O(v)$	$O(v)$
Communication	$O(w \cdot v)$	$O(w \cdot v)$	$O(w)$
Assumption	RSA	GDH	BDH
Client Unlinkability	Yes	No	Yes
Server Unlinkability	Yes	Yes	Yes
Forward Security	Yes	No	No

Table 4.1: Performance Comparison of PPIT Instantiations.

In conclusion, proposed PPIT constructions offer different computation/communication complexities and additional privacy features. Also, their security holds under different computation assumptions.

4.7 Proofs

This section concludes the chapter with proofs of proposed PPIT instantiations.

4.7.1 RSA-PPIT Proofs

RSA-PPIT in Figure 4.1 is *secure, server- and client-private* under the RSA assumption on safe RSA moduli and the GDH assumption (introduced in Chapter 2.3), in the Random Oracle Model, given semantically secure symmetric encryption.

Security and Server Privacy. To ease presentation, we first prove the security and server privacy in a setting where $w = 1$, i.e., the server runs on a single input record $(\mathsf{ID}, \mathsf{D})$. Next, we show how to easily generalize to the proof for multiple records. We start by showing that no efficient $\mathcal{A}$ (acting as a client) has a *non-negligible* advantage over $1/2$ against Ch in the following game:

1. Ch executes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\tau)$ and gives pk to $\mathcal{A}$.
2. $\mathcal{A}$ invokes $\mathsf{Authorize}$ on any ID_i of its choice (a polynomial number of times) and obtains the corresponding signature σ_i.

3. $\mathcal{A}$ generates ID_0^*, ID_1^* and two equal-length data records $\mathsf{D}_0{}^*, \mathsf{D}_1{}^*$.
4. $\mathcal{A}$ participates in Transfer as a client with message μ^*.
5. Ch selects one record pair by selecting a random bit b and executes the server's part of Transfer on public input pk and private inputs $(\mathsf{ID}_b^*, \mathsf{D}_b{}^*)$ with message (Z, CT, T).
6. $\mathcal{A}$ outputs b' and wins if $b = b'$.

Let HQuery be an event that $\mathcal{A}$ ever queries H_2 on input K^*, where K^* is defined (as the combination of message μ^* sent by $\mathcal{A}$ and message Z sent by Ch), as follows: $K^* = (\mu^*)^{eR_s} \cdot (h^*)^{-2R_s} \bmod N$, where $Z = g^{eR_s}$ and $h^* = H_1(\mathsf{ID}^*)$. In other words, HQuery is an event that $\mathcal{A}$ computes (and enters into hash function H_2) the key-material K^* for the challenging protocol.
[Claim 1]. Unless HQuery happens, $\mathcal{A}$'s view of interaction with Ch on bit $b = 0$ is indistinguishable from $\mathcal{A}$'s view of the interaction with Ch on bit $b = 1$.

Since the distribution of $R = g^{ez}$ is independent from $(\mathsf{ID}_b, \mathsf{D}_b)$, it reveals no information about which of $(\mathsf{ID}_b, \mathsf{D}_b)$ is related in the protocol. Since PPIT uses a semantically secure symmetric encryption, the distribution with $b = 0$ is indistinguishable from that with $b = 1$, unless $\mathcal{A}$ computes $k^* = H_2(K^*)$, in the random oracle model, by querying H_2, i.e., HQuery.
[Claim 2]. If event HQuery happens with non-negligible probability, then $\mathcal{A}$ can be used to violate the RSA assumption.

We describe a reduction algorithm called RCh using a modified challenger algorithm. Given the RSA challenge (N, e, α), RCh sets the public key as (N, e, g) where g is a generator of QR_N. RCh simulates signatures on each ID_i by assigning $H_1(\mathsf{ID}_i)$ as ${\sigma_i}^e \bmod N$ for some random value σ_i. In this way RCh can present the certificate of ID_i as σ_i. RCh embeds α to each H_1 query, by setting $H_1(\mathsf{ID}_i) = \alpha(a_i)^e$ for random $a_i \in \mathbb{Z}_N$. Given $(H_1(\mathsf{ID}_i))^d$ for any ID_i the simulator can extract $\alpha^d = (H_1(\mathsf{ID}_i))^d / a_i$.

We describe how RCh responds to $\mathcal{A}$ during Transfer and how RCh computes $(H_1(\mathsf{ID}_i))^d$ for certain ID_i. On $\mathcal{A}$'s input message μ^*, RCh picks a random $m \leftarrow \mathbb{Z}_{N/4}$, computes $Z = g^{(1+em)}$, and sends Z, arandom encryption CT, and a random T to $\mathcal{A}$. We remark that $g^{1+em} = g^{e(d+m)}$. On the HQuery event, RCh gets $K^* = (\mu^*)^{e(d+m)}(h^*)^{-2(d+m)}$ from $\mathcal{A}$. Since RCh knows μ^*, h^*, e, and m, RCh can compute $(h^*)^{2d}$. Since $gcd(2, e) = 1$, computing $(h^*)^{2d}$ leads to computing $(h^*)^d$.

We now extend to the setting where $w > 1$, i.e., the server holds more than a single record. The game is the same as above, except the adversary challenges the protocol on two pairs of input vectors $(\overrightarrow{\mathsf{ID}^*_0}, \overrightarrow{\mathsf{D}^*_0})$, $(\overrightarrow{\mathsf{ID}^*_1}, \overrightarrow{\mathsf{D}^*_1})$, instead of $(\mathsf{ID}^*_0, \mathsf{D}^*_0)$, $(\mathsf{ID}^*_1, \mathsf{D}^*_1)$. Namely, we demonstrate that no efficient $\mathcal{A}$ (acting as a client) has a *non-negligible* advantage over $1/2$ against Ch in the following game:

1. Ch executes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\tau)$ and gives pk to $\mathcal{A}$.

2. $\mathcal{A}$ invokes $\mathsf{Authorize}$ on $\mathsf{ID}_{s:j}$ of its choice and obtains the corresponding signature σ_j.

3. $\mathcal{A}$ generates two ID vectors:
   $\overrightarrow{\mathsf{ID}^*_0} = \{\mathsf{ID}_{0j}\}_{j=1,\dots,w}$, $\overrightarrow{\mathsf{ID}^*_1} = \{\mathsf{ID}_{1j}\}_{j=1,\dots,w}$,
   and two corresponding record vectors
   $\overrightarrow{\mathsf{D}^*_0} = \{\mathsf{D}_{0j}\}_{j=1,\dots,w}$, $\overrightarrow{\mathsf{D}^*_1} = \{\mathsf{D}_{1j}\}_{j=1,\dots,w}$.

4. $\mathcal{A}$ participates in $\mathsf{Transfer}$ as a client with message μ^*.

5. Ch selects one record pair by selecting a random bit b and executes the server's part of $\mathsf{Transfer}$ on public input pk and private inputs $(\overrightarrow{\mathsf{ID}^*_b}, \overrightarrow{\mathsf{D}^*_b})$ with message (Z, CT, T).

6. $\mathcal{A}$ outputs b' and wins if $b = b'$.

We define HQuery the same event as above. By the hybrid argument, if the adversary wins the above game with a non-negligible advantage over 1/2, HQuery happens on at least one pair $(\mathsf{ID}^*_{bj}, \mathsf{D}^*_{bj})$ out of $(\overrightarrow{\mathsf{ID}^*_b}, \overrightarrow{D^*_b})$. Using this adversary, we can build a reduction algorithm to break the RSA assumption, by the same argument as described above.

Client Privacy. In the following description, we use $U \approx_S V$ to denote that distribution U is statistically close to V in the sense that the difference between these distributions is at most $O(2^\tau)$. We show that $\{h^{2d} g^{R_c}\}_{R_c \leftarrow \mathbb{Z}_{N/4}} \approx_S QR_N$. Take any $h \in \mathbb{Z}^*_N$ and compute $\sigma = h^d \bmod N$. (Note that since $N - (p'q')$ is on the order of $\sqrt{N}$, which is negligible compared to N, the distribution of h chosen in $\mathbb{Z}_N$ is statistically close to uniform in $\mathbb{Z}^*_N$.) Since multiplication by h^{2d} is a permutation in QR_N, we have

$$\{h^{2d} g^{R_c}\}_{R_c \leftarrow \mathbb{Z}_{p'q'}} \equiv QR_N.$$

Since $\mathbb{Z}_{N/4} \approx_S \mathbb{Z}_{p'q'}$, the above implies that

$$\{h^{2d} g^{R_c}\}_{R_c \leftarrow \mathbb{Z}_{N/4}} \approx_S QR_N.$$

4.7.2 Schnorr-PPIT Proofs

Schnorr-PPIT in Figure 4.2 is *secure, server- and client-private* under the GDH assumption (introduced in Chapter 2.3) in the Random Oracle Model, given semantically secure symmetric encryption.

Security and Server Privacy. To ease presentation, we first prove the security and server privacy in a setting where $w = 1$, i.e., the server runs on a single input record $(\mathsf{ID}, \mathsf{D})$. Next, we show how to easily generalize to the proof for multiple records. We start by showing that no efficient $\mathcal{A}$ (acting as a client) has a *non-negligible* advantage over $1/2$ against Ch in the same game as in RSA-PPIT security and server privacy, except that now $\mathcal{A}$ sends X^* instead of μ^*.

Let HQuery be an event that $\mathcal{A}$ ever queries H_2 on input K^*, where K^* is defined via the combination of the message X^* sent by $\mathcal{A}$ and message R sent by Ch, as follows: $K^* = (X^*)^{R_s} \cdot (y)^{cR_s} \bmod p$, where $R = g^{R_s}$ and $c = H_1(i, X^*)$.

[Claim 3]: Unless HQuery happens, $\mathcal{A}$'s view of the interaction with the challenger on bit $b = 0$ is indistinguishable from $\mathcal{A}$'s view of the interaction with the challenger on bit $b = 1$.

[Claim 4]: If event HQuery happens with non-negligible probability, then $\mathcal{A}$ can be used to break the CDH assumption.

We describe a reduction algorithm called RCh using a modified challenger algorithm. The goal of a CDH problem on $(p, q, g, y = g^a, R = g^{R_s})$ is to compute $g^{aR_s} \bmod p$. RCh takes $(p, q, g, y = g^a)$ as its public key and simulates the signatures (X_i, s_i) on each ID_i by taking random s_i, e_i and computing $X_i = g^{s_i} \cdot y^{e_i}$ and assigning $H_1(i, X_i)$ to e_i. Since the verification equation is satisfied and s_i, e_i are picked at random, this is indistinguishable from receiving real signatures. In the protocol on $\mathcal{A}$'s input X^*, RCh responds with $R = g^{R_s}$ and random encryption CT and random T.

Assume that HQuery happens, which can be detected by querying to DDH oracle on $(g, X^* \cdot y^e, g^{R_s}, Q_H)$ for every query input Q_H to H_1. Then, as in the forking lemma argument of [134], we know that $\mathcal{A}$ can be executed twice in a row with the same value $X = g^k \bmod p$ and different hash values such that $(e \neq e')$ and $\mathcal{A}$ wins both games with non-negligible probability of at least $\frac{\epsilon^2}{q_h}$, where q_H is the number of queries $\mathcal{A}$ makes to the hash function. This means, $\mathcal{A}$ can compute with non-negligible probability the values $K = g^{R_s(ea+k)} \bmod p$ and $K' = g^{R_s(e'a+k)} \bmod p$ with $e \neq e'$. Consequently, $\mathcal{A}$ can also efficiently compute g^{aR_s}:

$$(K/K')^{(e-e')^{-1}} = (g^{R_s ea - R_s e' a})^{(e-e')^{-1}} = (g^{R_s a(e-e')})^{(e-e')^{-1}} = g^{aR_s} \bmod p.$$

We now extend to the setting where $w > 1$, i.e., the server holds more than a single records. The game is the same as above, except that $\mathcal{A}$ selects two pairs of $(\overrightarrow{\mathsf{ID}_0^*}, \overrightarrow{\mathsf{D}_0^*})$, $(\overrightarrow{\mathsf{ID}_1^*}, \overrightarrow{\mathsf{D}_1^*})$, instead of $(\mathsf{ID}_0^*, \mathsf{D}_0^*)$, $(\mathsf{ID}_1^*, \mathsf{D}_1^*)$. We define HQuery the same event as above. By the hybrid argument, if the adversary wins the above game with a non-negligible advantage over 1/2, HQuery happens on at least one pair $(\mathsf{ID}_{bj}^*, \mathsf{D}_{bj}^*)$ out of $(\overrightarrow{\mathsf{ID}_b^*}, \overrightarrow{\mathsf{D}_b^*})$. Using this adversary, we can build a reduction algorithm to break the GDH assumption, by the same argument as described above.

Client Privacy. Client privacy is easy to show since $X = g^k$ for random k is independent from any ID value.

4.7.3 IBE-PPIT

IBE-PPIT in Figure 4.3 is *secure, server- and client-private* if the underlying IBE scheme, i.e., [21], is semantically secure and key-private under selective ID attack.

Security and Server Privacy. Assuming an underlying IBE system semantically secure under a chosen plaintext attack, such as [21], the resulting PPIT scheme is *secure* against malicious client. We prove this claim by contradiction. Assuming our claim is not true, then there exists a polynomial-bounded adversary $\mathcal{A}$ that wins the security game presented in Section 4.2. $\mathcal{A}$ is given $\mathsf{pk} = \mathsf{ID}$ and the IBE-encryption of D under the key pk but not the corresponding sk. If $\mathcal{A}$ decrypts D with non-negligible probability, then we can construct a polynomial-bounded adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the CPA-security of IBE. This contradicts our assumption.
Finally, *server privacy* is trivially achieved if the underlying IBE scheme is key-private.

We stress that the use of key tags to reduce the number of client's decryptions do not affect the security and privacy of the scheme assuming the security of Boneh and Franklin's IBE instantiation [21]. Key tags are nothing but bilinear maps operations as in BF-IBE. Hence, we claim that if the tags give any advantage to the adversary then there exists another polynomial-bounded adversary that breaks the security of BF-IBE, again, contradicting our assumption. As the re-use of randomness z (as described in Figure 4.3) has been proved to be CPA-secure in [23], we omit formal reductions here to ease presentation.

Client Privacy. Client privacy is trivial since the server does not receive any information from client during $\mathsf{Transfer}$.

Chapter 5

Linear-Complexity Private Set Intersection Protocols

In this chapter, we propose several Private Set Intersection (PSI) protocols, secure in the presence of semi-honest adversaries, that are appreciably more efficient than state-of-the-art. We introduce the notion of Authorized Private Set Intersection (APSI) and show how some PPIT instantiations, presented in the previous chapter, can be adapted to APSI. We then construct an improved APSI protocol and shows how to derive efficient PSI. Finally, we propose an even more efficient PSI protocol geared for scenarios where the server can perform some pre-computation and/or the client has limited computational resources.

5.1 From PPIT to PSI, via APSI

The previous chapter presented Policy-based Information Transfer (PPIT) – a cryptographic tool that lets a client obtain information from a server, in a private and policy-guided (authorized) manner. It is geared for the case where the client is interested in a single record, while the server holds many database records.

In this chapter, we focus on PSI protocols. PSI involves two participants, a server and a client, each on input a private set, and lets the client learn the set intersection, while the server learns nothing. We also introduce an *authorized* variant, that we call Authorized Private Set Intersection (APSI): in APSI, each item in client set must be authorized (i.e., signed) by some recognized and mutually trusted authority.

Another natural PSI extension is what we call *PSI with Data Transfer* or PSI-DT. In it, the server has some data associated with each item in its set, e.g., a database record. Therefore, the client, along with the set intersection, also obtains data records associated with each item in the intersection. Interestingly, we show that some previously-discussed PPIT instantiations, in fact, provide APSI-DT for the case where one participant (the client) has a set of size one.

In contrast to prior work, we do not start with constructing PSI protocols and piling on extra features later. Instead, we start from PPIT and show that RSA-PPIT (Section 4.3) can be trivially extended into inefficient PSI and APSI protocols, with and without data transfer. We then construct several efficient (and less trivial) provably secure PSI and APSI protocols that incur **linear** computation and communication overhead, and are appreciably more efficient than current state-of-the-art. Next, we construct another, even more efficient, PSI protocol geared for scenarios where the server can perform some pre-computation and/or the client is computationally weak. Protocols in this chapter achieve security in the presence of semi-honest adversaries. In Chapter 6, we extend them to offer security in the malicious adversary model.

5.2 Definitions

In this section, we present our definitions for PSI functionality, as well as for APSI and PSI with Data Transfer (PSI-DT).

Definition 5.1 (Private Set Intersection (PSI)). *It involves a* server *and a* client*, on input* $\mathcal{S} = \{s_1, \ldots, s_w\}$ *and* $\mathcal{C} = \{c_1, \ldots, c_v\}$*, respectively. It results in the client outputting* $\mathcal{S} \cap \mathcal{C}$*. PSI securely implements the functionality:* $\mathcal{F}_{\mathsf{PSI}} : ((\mathcal{S}, v), (\mathcal{C}, w)) \mapsto (\perp, \mathcal{S} \cap \mathcal{C})$.

Definition 5.2 (Authorized Private Set Intersection (APSI)). *It involves a* server *and a* client*, on input* $\mathcal{S} = \{s_1, \ldots, s_w\}$ *and* $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$*, respectively. It results in the client outputting* $\mathsf{ASI} \overset{def}{=} \{s_j \in \mathcal{S} \mid \exists\, (c_i, \sigma_i) \in \mathcal{C} \text{ s.t. } c_i = s_j \wedge \mathsf{Vrfy}_{\mathsf{pk}}(\sigma_i, c_i) = 1\}$*, where* pk *is the public key of an (offline) trusted authority (denoted as CA). APSI securely implements the functionality:* $\mathcal{F}_{\mathsf{APSI}} : ((\mathcal{S}, v), (\mathcal{C}, w)) \mapsto (\perp, \mathsf{ASI})$.

Privacy Properties. Definitions above use the standard *secure computation* formulation [77] and capture privacy guarantees provided by the functionalities. Nonetheless, for the sake of clarity, we also provide an informal (yet concise) discussion of (A)PSI privacy features.

- ***Correctness.*** At the end of the protocol, the client outputs the exact (possibly empty) intersection of the two respective sets.
- ***Server Privacy.*** The client learns no information (except the size) about the subset of server items that are NOT in the intersection of their respective sets.
- ***Client Privacy.*** No information is leaked about client set items to a malicious server, except set size.
- ***Unlinkability (Optional).*** Unlinkability guarantees that a malicious server (resp., client) cannot tell if any two protocol instances are related, i.e., executed on the same inputs by the client (resp., server).

For APSI, server privacy is amended as follows:

- ***Server Privacy (APSI).*** The client learns no information (except the size) about the subset of server items that are NOT in the intersection of their respective sets, for which the client does not hold a valid authorization.

Finally, we consider the following extension:

Definition 5.3 (**Private Set Intersection with Data Transfer (PSI-DT)).** *It involves a* server *and a* client*, on input* $\mathcal{S} = \{(s_1, data_1), \ldots, (s_w, data_w)\}$ *and* $\mathcal{C} = \{c_1, \ldots, c_v\}$*, respectively. It results in the client outputting* $\mathsf{SI\text{-}DT} \overset{def}{=} \{(s_j, data_j) \in \mathcal{S} \mid \exists\, c_i \in \mathcal{C} \; s.t. \; c_i = s_j\}$. *PSI-DT securely implements the functionality:* $\mathcal{F}_{\mathsf{PSI-DT}} : ((\mathcal{S}, v), (\mathcal{C}, w)) \mapsto (\perp, \mathsf{SI\text{-}DT})$.

Also, APSI can naturally be extended to a *Authorized Private Set Intersection with Data Transfer* (APSI-DT) variant. Since this extension mirrors its PSI counterpart, we omit it here to ease presentation.

5.3 Designing Efficient Private Set Intersection Protocols

This section presents our efficient (linear-complexity) APSI and PSI protocols.

5.3.1 Baseline: APSI from PPIT

The starting point for our design is an APSI protocol derived from RSA-PPIT, introduced in Section 4.3. Since our new protocols are loosely based on it, we specify it in Figure 5.1.

[**Common input:** $N, e, g, H_1(\cdot), H_2(\cdot)$]

Client, on input: $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$,
where $\forall i\ \sigma_i = H_1(c_i)^d \bmod N$

Server, on input: $\mathcal{S} = \{s_1, \ldots, s_w\}$

For $i = 1, \ldots, v$:
$R_{c:i} \leftarrow_r \mathbb{Z}_{N/4}$
$M_i = {\sigma_i}^2 \cdot g^{R_{c:i}}$

$$\xrightarrow{\{M_1, \ldots, M_v\}}$$

$(\hat{s}_1, \ldots, \hat{s}_w) \leftarrow \Pi(\mathcal{S})$
$R_s \leftarrow_r \mathbb{Z}_{N/4}$, $Z = g^{eR_s}$
For $i = 1, \ldots, v,\ j = 1, \ldots, w$:
$K_{s:ij} = (M_i)^{eR_s} \cdot H_1(\hat{s}_j)^{-2R_s}$
$T_{s:ij} = H_2(K_{s:ij})$

$$\xleftarrow{Z, \{T_{s:11}, \ldots, T_{s:vw}\}}$$

For $i = 1, \ldots, v$:
$K_{c:i} = Z^{R_{c:i}}$
$T_{c:i} = H_2(K_{c:i})$

Output: $\{c_i \mid c_i \in \mathcal{C}$ and $\exists\ T_{s:ij}$ s.t. $T_{s:ij} = T_{c:i}\}$

[All computation is $\bmod N$]

Figure 5.1: APSI protocol derived from RSA-PPIT.

Similar to RSA-PPIT, an (offline) trusted authority, CA, generates the RSA parameters, i.e., (N, e, d), at setup time. (N, e) are published, alongside two cryptographic hash functions (modeled as random oracles) i.e., $H_1 : \{0,1\}^* \to \mathbb{Z}_N$ (full-domain hash) and $H_2 : \{0,1\}^* \to \{0,1\}^\tau$, and and a generator g of QR_N. CA's secret key is d. In order to obtain an authorization on a given item c_i, the client interacts with CA and receives an RSA signature on c_i, i.e., $H_1(c_i)^d \bmod N$.

Finally, note that the server uses a random permutation $\Pi(\cdot)$ over set $\mathcal{S}$. The goal is to prevent the client from inferring additional information over $\mathcal{S}$ in case it has some knowledge on the *order* of items in $\mathcal{S}$.

Correctness. The APSI protocol in Figure 5.1 is correct, since: for any (σ_i, c_i) held by the client and $\hat{s}_j$ held by the server, if: (1) σ_i is a genuine CA's signature on c_i, and (2) $c_i = \hat{s}_j$:

$$K_{s:ij} = (M_i)^{eR_s} \cdot H_1(\hat{s}_j)^{-2R_s} = (\sigma_i^2 \cdot g^{R_{c:i}})^{eR_s} \cdot H_1(\hat{s}_j)^{-2R_s} =$$
$$= H_1(c_i)^{2R_s} \cdot g^{eR_s \cdot\, R_{c:i}} \cdot H_1(\hat{s}_j)^{-2R_s} = g^{eR_s \cdot\, R_{c:i}} = Z^{R_{c:i}} = K_{c:i} \qquad (5.1)$$

Thus, $T_{s:ij} = H_2(K_{s:ij}) = H_2(K_{c:i}) = T_{c:i}$.

Protocol Complexity. The protocol in Figure 5.1 incurs linear ($O(v)$) computation complexity at client side and quadratic ($O(w\cdot v)$) computation overhead at server side. Communication complexity is also quadratic ($O(w \cdot v)$). However, we can reduce the number of on-line exponentiations on the server from $O(w \cdot v)$ to $O(w + v)$. The server can compute, separately, $H_1(\hat{s}_j)^{-2R_s}$ and $(M_i)^{eR_s}$. However, the number of modular multiplications, as well as the communication overhead, would still be quadratic.

5.3.2 APSI with Linear Costs

The trivial derivation of APSI from RSA-PPIT is relatively inefficient. We now show how to use it to derive an improved protocol, presented in Figure 5.2.

[**Common input:** $N, e, g, H_1(\cdot), H_2(\cdot)$]

Client, on input: $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$, where $\forall i\ \sigma_i = H_1(c_i)^d \bmod N$

Server, on input: $\mathcal{S} = \{s_1, \ldots, s_w\}$

```
Client:
$R_c \leftarrow_r \mathbb{Z}_{N/4}$, $X = g^{R_c}$
For $i = 1, \ldots, v$ :
    $R_{c:i} \leftarrow_r \mathbb{Z}_{N/4}$
    $M_i = {\sigma_i}^2 \cdot g^{R_{c:i}}$
```

$$\xrightarrow{X, \{M_1, \ldots, M_v\}}$$

```
Server:
$(\hat{s}_1, \ldots, \hat{s}_w) \leftarrow \Pi(\mathcal{S})$
$R_s \leftarrow_r \mathbb{Z}_{N/4}$, $Z = g^{eR_s}$
For $i = 1, \ldots, v$ :
    $M'_i = (M_i)^{eR_s}$
For $j = 1, \ldots, w$ :
    $K_{s:j} = (X^e \cdot H_1(\hat{s}_j)^2)^{R_s}$
    $T_{s:j} = H_2(K_{s:j})$
```

$$\xleftarrow[\{M'_1, \ldots, M'_v\}]{Z, \{T_{s:1}, \ldots, T_{s:w}\},}$$

```
Client:
For $i = 1, \ldots, v$ :
    $K_{c:i} = M'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$
    $T_{c:i} = H_2(K_{c:i})$
```

Output: $\{c_i \mid c_i \in \mathcal{C}$ and $\exists\, T_{s:j}$ s.t. $T_{s:j} = T_{c:i}\}$

[All computation is $\bmod N$]

Figure 5.2: New APSI protocol with linear complexities.

Similar to the APSI protocol derived from RSA-PPIT, we assume that an (offline) trusted authority, CA, generates RSA parameters at setup time. (N, e) are published, alongside two cryptographic hash functions (modeled as random oracles) i.e., $H_1 : \{0,1\}^* \to \mathbb{Z}_N$ (full-domain hash) and $H_2 : \{0,1\}^* \to \{0,1\}^\tau$, and and a generator g of QR_N. CA's secret key is d. Again, the server uses a random permutation $\Pi(\cdot)$ over set $\mathcal{S}$, in order to prevent the client from inferring additional information over $\mathcal{S}$ in case it has some knowledge on the *order* of items in $\mathcal{S}$.

Correctness. The APSI protocol in Figure 5.2 is correct, since: for any (σ_i, c_i) held by the client and $\hat{s}_j$ held by the server, if: (1) σ_i is a genuine CA's signature on c_i, and (2) $c_i = \hat{s}_j$:

$$\begin{aligned} K_{c:i} =& M_i' \cdot Z^{R_c} \cdot Z^{-R_{c:i}} = (M_i)^{eR_s} \cdot g^{eR_cR_s} \cdot g^{-eR_sR_{c:i}} = ({\sigma_i}^2 \cdot g^{R_{c:i}})^{eR_s} \cdot g^{eR_cR_s} \cdot g^{-eR_sR_{c:i}} = \\ =& H_1(c_i)^{2R_s} \cdot g^{eR_cR_s} = X^{eR_s} \cdot H_1(\hat{s}_j)^{2R_s} = K_{s:j} \end{aligned} \tag{5.2}$$

Thus, $T_{s:j} = H_2(K_{s:j}) = H_2(K_{c:i}) = T_{c:i}$.

Protocol Complexity. The protocol in Figure 5.2 incurs linear computation (for both participants) and communication complexity. Specifically, the client performs $O(v)$ exponentiations and the server – $O(w+v)$. Communication is dominated by server's reply – $O(w+v)$.

Security. We intentionally omit security proofs for the APSI protocol in Figure 5.2. In fact, in Chapter 6, we will show how this APSI protocol (with semi-honest security) can be extended to achieve security in the presence of malicious adversaries (under the RSA and DDH assumption in ROM). Nonetheless, [52] presents formal proofs for the APSI protocol in Figure 5.2 (relying on the RSA assumption in ROM).

5.3.3 Deriving Efficient PSI

We now convert the above APSI protocol into PSI. In doing so, the main change is the obviated need for the RSA setting. Instead, the protocol operates in $\mathbb{Z}_p$, where p is a large prime, and selects random exponents from a subgroup of size q, where $q|p-1$. This change makes the protocol more efficient, especially, because of smaller exponents (e.g., $|q|$=160 bits).

We assume that the protocol runs on public input p, q, g (where g is a generator of the subgroup of order q), and two cryptographic hash functions (modeled as random oracles), $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^* \to \{0,1\}^\tau$. Once again, the server uses a random permutation $\Pi(\cdot)$ over set $\mathcal{S}$, in order to prevent the client from inferring additional information over $\mathcal{S}$ in case it has some knowledge on the *order* of items in $\mathcal{S}$.

The basic complexity of the resulting protocol remains the same: linear communication and computational overhead (specifically, $O(w+v)$, for the server and $O(v)$ for the client). However, if the server can pre-compute all values of the form: $H_1(\hat{s}_j)^{R_s}$, the cost of computing all $K_{s:j}$ values can be reduced to $O(w)$ multiplications (from $O(w)$ exponentiations). (The same optimization applies to the APSI protocol in Figure 5.2). The resulting PSI construct is illustrated in Figure 5.3:

[**Common input:** $p, q, g, H_1(\cdot), H_2(\cdot)$]

Client, on input: $\mathcal{C} = \{c_1, \ldots, c_v\}$ — **Server**, on input: $\mathcal{S} = \{s_1, \ldots, s_w\}$

Client:

$R_c \leftarrow_r \mathbb{Z}_q$, $X = g^{R_c}$

For $i = 1, \ldots, v$:

- $R_{c:i} \leftarrow_r \mathbb{Z}_q$
- $M_i = H_1(c_i) \cdot g^{R_{c:i}}$

$$\xrightarrow{X, \{M_1, \ldots, M_v\}}$$

Server:

$(\hat{s}_1, \ldots, \hat{s}_w) \leftarrow \Pi(\mathcal{S})$

$R_s \leftarrow_r \mathbb{Z}_q$, $Z = g^{R_s}$

For $i = 1, \ldots, v$:

- $M_i' = (M_i)^{R_s}$

For $j = 1, \ldots, w$:

- $K_{s:j} = (X \cdot H_1(\hat{s}_j))^{R_s}$
- $T_{s:j} = H_2(K_{s:j})$

$$\xleftarrow[\{M_1', \ldots, M_v'\}]{Z, \{T_{s:1}, \ldots, T_{s:w}\},}$$

Client:

For $i = 1, \ldots, v$:

- $K_{c:i} = M_i' \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$
- $T_{c:i} = H_2(K_{c:i})$

Output: $\{c_i \mid c_i \in \mathcal{C} \text{ and } \exists\, T_{s:j} \text{ s.t. } T_{s:j} = T_{c:i}\}$

[All computation is $\bmod\ p$]

Figure 5.3: PSI protocol with linear complexities.

Correctness. Similar to its APSI counterpart, it is easy to see that the PSI protocol in Figure 5.3 is correct, since: for any c_i held by the client and $\hat{s}_j$ held by the server, if $c_i = \hat{s}_j$:

$$\begin{aligned} K_{c:i} =& M_i' \cdot Z^{R_c} \cdot Z^{-R_{c:i}} = (M_i)^{R_s} \cdot g^{R_c R_s} \cdot g^{-R_s R_{c:i}} = \\ =& (H_1(c_i) \cdot g^{R_{c:i}})^{R_s} \cdot g^{R_c R_s} \cdot g^{-R_s R_{c:i}} = (X \cdot H_1(\hat{s}_j))^{R_s} = K_{s:j} \end{aligned} \tag{5.3}$$

Thus, $T_{s:j} = H_2(K_{s:j}) = H_2(K_{c:i}) = T_{c:i}$.

Security. Security proofs for the PSI protocol (with semi-honest security) in Figure 5.3 are intentionally omitted, since in Chapter 6, we will introduce a protocol variant that achieves security in the presence of malicious adversaries (under the DDH assumption in ROM). Nonetheless, formal proofs can be found in [52]. (Security arguments are based on the One-More Diffie-Hellman assumption in ROM [14]).

5.4 Efficient PSI based on RSA Blind-Signatures

Although efficient, the PSI protocol in Figure 5.3 is *sub-optimal* for application scenarios where the client runs on a resource-poor device, e.g., a PDA or a smartphone. $O(v)$ exponentiations might still represent a fairly heavy burden. Also, if server set is very large, overhead incurred (at server side) by $O(w)$ on-line modular exponentiations (or even just $O(w)$ multiplications, if pre-computation is allowed) might be substantial.

To this end, in Figure 5.4, we present yet another PSI construct, based on RSA Blind Signatures [38]. In it, the client does not perform any modular exponentiations on-line, only $O(v)$ modular multiplications. Also, server on-line workload does *not* depend on the size of its own set.

We assume that, at setup time, the server uses the RSA key generation algorithm, on input a security parameter τ, to generate (N, e, d). The protocol is then run on public input (N, e), and two cryptographic hash functions (modeled as random oracles) $H_1 : \{0,1\}^* \to \mathbb{Z}_N$ (full-domain hash) and $H_2 : \{0,1\}^* \to \{0,1\}^\tau$, chosen at setup time.

Finally, similar to protocols in Section 5.3, the server needs to use a random permutation $\Pi(\cdot)$ over set $\mathcal{S}$. Again, the goal is to prevent the client from inferring additional information over $\mathcal{S}$ in case it has some knowledge on the *order* of items in $\mathcal{S}$.

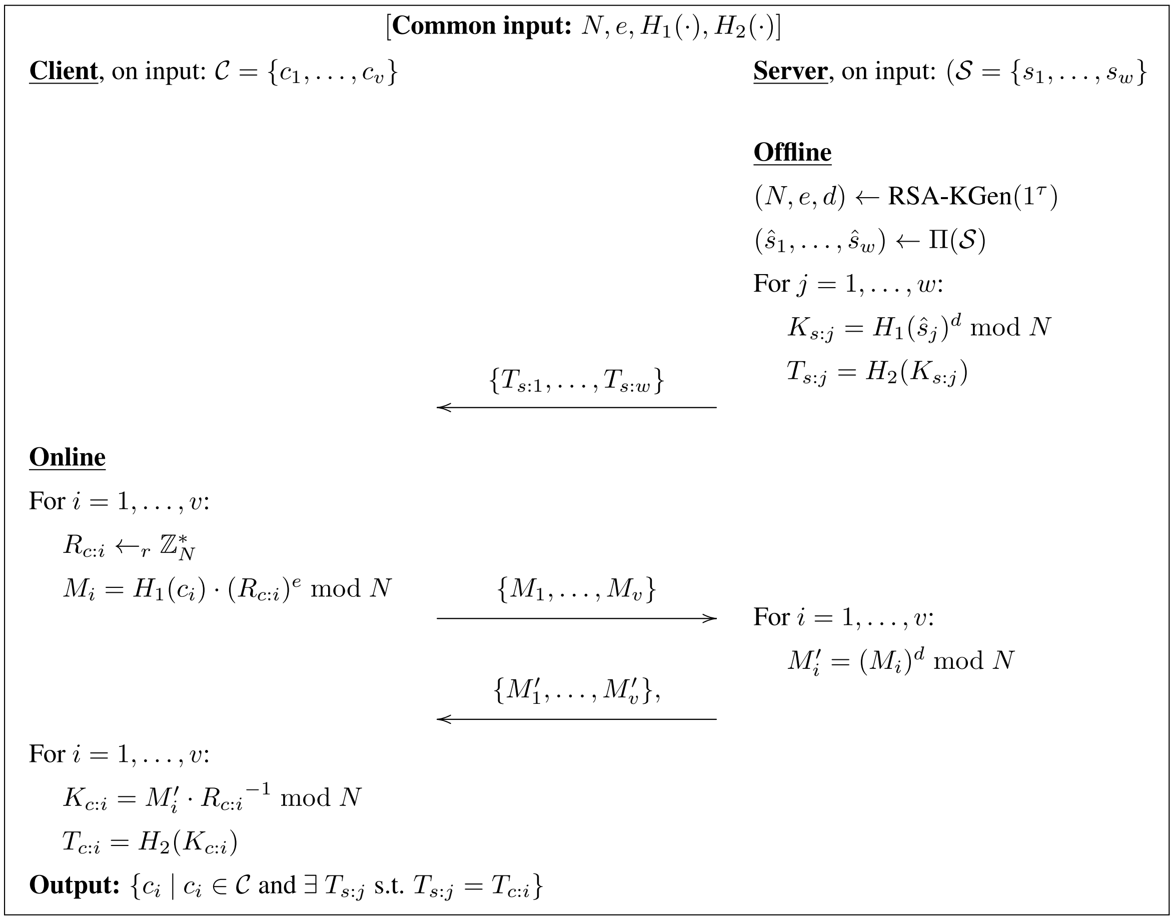
Correctness. It is easy to see that this protocol is correct, since: for any c_i held by the client and $\hat{s}_j$ held by the server, if $c_i = \hat{s}_j$, then:

$$K_{c:i} = M_i' \cdot {R_{c:i}}^{-1} = {M_i}^d \cdot {R_{c:i}}^{-1} = (H_1(c_i) \cdot {R_{c:i}}^e)^d \cdot {R_{c:i}}^{-1} =$$

$$= H_1(c_i)^d \cdot {R_{c:i}}^{ed} \cdot {R_{c:i}}^{-1} = H_1(\hat{s}_j)^d = K_{s:j}$$

Thus, $T_{s:j} = H_2(K_{s:j}) = H_2(K_{c:i}) = T_{c:i}$.

Protocol Complexity. Server's on-line computation overhead is limited to $O(v)$ exponentiations, while its pre-computation requires $O(w)$ exponentiations, owing to RSA signatures. Note that RSA keys are generated by the server. By taking advantage of the Chinese Remainder Theorem



[**Common input:** $N, e, H_1(\cdot), H_2(\cdot)$]

Client, on input: $\mathcal{C} = \{c_1, \ldots, c_v\}$

Server, on input: $(\mathcal{S} = \{s_1, \ldots, s_w\}$

Offline

$(N, e, d) \leftarrow$ RSA-KGen(1^τ)

$(\hat{s}_1, \ldots, \hat{s}_w) \leftarrow \Pi(\mathcal{S})$

For $j = 1, \ldots, w$:

$K_{s:j} = H_1(\hat{s}_j)^d \bmod N$

$T_{s:j} = H_2(K_{s:j})$

$\xleftarrow{\{T_{s:1}, \ldots, T_{s:w}\}}$

Online

For $i = 1, \ldots, v$:

$R_{c:i} \leftarrow_r \mathbb{Z}_N^*$

$M_i = H_1(c_i) \cdot (R_{c:i})^e \bmod N$

$\xrightarrow{\{M_1, \ldots, M_v\}}$

For $i = 1, \ldots, v$:

$M_i' = (M_i)^d \bmod N$

$\xleftarrow{\{M_1', \ldots, M_v'\},}$

For $i = 1, \ldots, v$:

$K_{c:i} = M_i' \cdot {R_{c:i}}^{-1} \bmod N$

$T_{c:i} = H_2(K_{c:i})$

Output: $\{c_i \mid c_i \in \mathcal{C}$ and $\exists\, T_{s:j}$ s.t. $T_{s:j} = T_{c:i}\}$

Figure 5.4: Efficient PSI protocol based on RSA Blind Signatures.

(CRT) [104], server's exponentiations can be speeded up by a factor of (approximately) 4. Client's overhead involves $O(v)$ multiplications, since, as is well-known that, e can be a small integer. Note that, although this protocol uses the RSA setting, RSA parameters are initialized *a priori* by the server. This is in contrast to the protocol in Figure 5.2 where CA sets up RSA parameters.

Protocol Linkability. Although very efficient, PSI protocol in Figure 5.4 has some drawbacks. First, it is unclear how to extend it to APSI. Second, if pre-computation is impossible, its performance becomes comparable to that of the PSI protocol in Figure 5.3, since the latter uses short exponents both at server and client side. In terms of privacy properties, this protocol lacks server unlinkability. (Recall that this feature is relevant if the protocol is run multiple times.) The server computes tags of the form $T_{s:j} = H_2(H_1(\hat{s}_j)^d)$. Consequently, running the protocol twice allows the client to observe changes in server set.

There are several ways of patching the protocol. One is for the server to select a new set of RSA parameters for each protocol instance. This would be a time-consuming extra step at the start of the protocol; albeit, with pre-computation, no extra on-line work would be required. Two additional initial messages would be necessary: one from the client – to "wake up" server, and the other – from the server to the client bearing the new RSA public key and $\{T_{s:1}, .., T_{s:w}\}$. Another simple way of providing server unlinkability is to change the hash function H_1, for each protocol instance. If we assume that the client and the server maintain either a common protocol counter (monotonically increasing and non-wrapping) or sufficiently synchronized clocks, it is easy to select/index a distinct hash function based on such unique and common values. One advantage of this approach is that we no longer need the two extra initial messages.

Proofs of PSI Protocol in Figure 5.4

We start by claiming that the use of different randomness across multiple interactions ($R_{c:i}$-s at client) trivially yields client unlinkability.

Next, proving client privacy is also straightforward. In fact, client inputs to the protocol are statistically close to random distribution. Also, privacy directly follows from the security argument of blind RSA signatures [38].

Finally, we prove server privacy by presenting a concise construction of an ideal (*adaptive*) world SIM_c from a honest-but-curious real-world client C^*, and show that the views of C^* in the real game with the real world server and in the interaction with SIM_c are indistinguishable, under the *One-More-RSA* assumption in ROM.

First, SIM_c runs $(N, e, d) \leftarrow$ RSA-Keygen(τ) and gives (N, e) to C^*. SIM_c models the hash function $H_1(\cdot)$ and $H_2(\cdot)$ as random oracles. A query to $H_1(\cdot)$ is recorded as $(q, h = H_1(q))$, a query to $H_2(\cdot)$ as $(k, h' = H_2(k))$, where q and h' are random values. Finally, SIM_c creates two empty sets A, B. During interaction, SIM_c publishes the set $T = \{t_1, \cdots, t_w\}$, where t_j is taken at random. Also, for every $M_i \in \{M_1, \cdots, M_v\}$ received from C^* (recall that $M_i = H_1(c_i) \cdot (R_{c:i})^e$), SIM_c answers according to the protocol with $(M_i)^d$.

We now describe how SIM_c answers to queries to $H_2(\cdot)$. *On query k to $H_2(\cdot)$*, SIM_c checks whether it has recorded a value h s.t. $h = k^e$ (i.e., $h^d = k$).

If $!\exists h$ s.t. $h = k^e$, SIM_c answers a random value h' and record (k, h') as mentioned above.

If $\exists h$ s.t. $h = k^e$, SIM_c can recover the q s.t. $h = H_1(q)$ and $h = k^e$. Then, it checks whether it has previously been queried on the value k.

If $\exists k$ s.t. k has already been queried, then SIM_c checks whether $q \in A$. If $q \notin A$, it means that C^* queried q to $H_1(\cdot)$ (which returned h), and also made an independent query k to $H_2(\cdot)$ s.t. $h = k^e$. In this case SIM_c aborts the protocol. However, it easy to see that this happens with negligible probability. Instead, if $q \in A$, SIM_c returns the value h' previously stored for k.

If $!\exists k$ s.t. k has already been queried, this means that SIM_c is learning one of C^*'s outputs. Hence, $A = A \cup \{q\}$. Then, SIM_c checks if $|A| > v$.

If $|A| <= v$, then SIM_c checks if $q \in \mathcal{C} \cap \mathcal{S}$ by playing the role of the client with the real world server. If $q \in \mathcal{C} \cap \mathcal{S}$, SIM_c answers to the query on k with a value $t_j \in T \backslash B$, records the answer (k, t_j) and sets $B = B \cup \{t_j\}$. If $q \notin \mathcal{C} \cap \mathcal{S}$, SIM_c answers with a random value h' and records the answer.

If $|A| > v$, then we can construct a reduction Red breaking the *One-More-RSA* assumption. The reduction Red can be constructed as follows. Red answers to C^*'s queries to $H_1(\cdot)$ with RSA challenges $(\alpha_1, \cdots, \alpha_{ch})$. During interaction, on C^*'s messages $M_i \in \{M_1, \cdots, M_v\}$, Red answers $(M_i)^d$ by querying the RSA Oracle. Finally, if the case discussed above happens, at the end of the protocol the set B will contain at least $(v + 1)$ elements, where v is the number of RSA challenges, thus violating the *One-More-RSA assumption*. As a result, we have shown that the views of C^* in the real game with the real world server and in the interaction with SIM_c are indistinguishable.

The structure of the above proof resembles the one by Jarecki and Liu in [100] (reviewed in Section 3.3.1) with security under the One-More-Gap-DH assumption in ROM. We also re-use the notion of *adaptiveness* for PSI, needed to let client adaptively make queries (i.e., client inputs do not need to be specified all at once).

5.5 Realizing PSI-DT and APSI-DT

We can easily add the *data transfer* functionality to the protocols in Figures 5.1, 5.2, 5.3, and 5.4, thus, implement APSI-DT and PSI-DT at no extra asymptotic cost. We assume that an additional secure cryptographic hash function $H_3 : \{0,1\}^* \rightarrow \{0,1\}^\tau$ is chosen during setup. In all the protocols proposed in this chapter, we then use $H_3(\cdot)$ to derive a symmetric key for a CPA-secure symmetric cipher, such as AES [45], used in the appropriate mode of operation. For every $j = 1, \ldots, w$, the server computes $k_{s:j} = H_3(K_{s:j})$ and encrypts associated data using a distinct key $k_{s:j}$. For its part, the client, for every $i = 1, \ldots, v$, computes $k_{c:i} = H_3(K_{c:i})$ and decrypts ciphertexts corresponding to the matching tag. (Note that $ks_j = kc_i$ if and only if $\hat{s}_j = c_i$ and so

$T_{s:j} = T_{c:i}$). As long as the underlying encryption scheme is CPA-secure, this extension does not affect security or privacy arguments for any protocol discussed thus far.

5.6 Performance Evaluation

In this section, we highlight the differences between prior PSI techniques (presented in Section 3.3) and protocols proposed in this chapter. We focus on asymptotic complexities for: (1) communication overhead and (2) server and client computation (in terms of "expensive" operations, such as modular exponentiations).

Let w and v denote the number of items in server and client sets, respectively. Let m be the number of bits needed to represent each item. We distinguish between *online* and *offline* operations. Protocols are compared in Table 5.1, choosing parameters such that all protocols achieve 80-bit security. The first three rows refer to APSI protocols and the last seven – to PSI. The table also includes communication overhead.

Protocol	**Model**	**Communic.**	**Pre-Comp.**	**Server Comput.**	**Client Comput.**	**Mod**
[31]	Std	$O(vw)$	-	$O(wv)$ 160-bit	$O(wv)$ 160-bit	1024
APSI Fig.5.1	ROM	$O(vw)$	$O(w)$ 1024-bit	$O(v)$ 1024-bit exps $O(vw)$ mults	$O(v)$ 1024-bit	1024
APSI Fig.5.2	ROM	$O(w{+}v)$	$O(w)$ 512-bit	$O(v)$ 512-bit	$O(v)$ 512-bit	1024
PSI in [66]	Std	$O(w{+}v)$	-	$O(w \log\log v)$ m-bit	$O(w{+}v)$ 160-bit	1024
PSI in [87]	Std	$O(w{+}v)$	-	$O(v{+}w(\log\log v))$ 160-bit	$O(w{+}v)$ 160-bit	1024
PSI in [108]	Std	$O(w{+}v)$	-	$O(wv)$ 1024-bit	$O(wv)$ 1024-bit	2048
PSI in [98]	Std	$O(w{+}v)$	$O(w)$ 1024-bit	$O(v)$ 1024-bit	$O(v)$ m-bit	2048
PSI in [100]	ROM	$O(w{+}v)$	$O(w)$ 160-bit	$O(v)$ 160-bit	$O(v)$ 160-bit	1024
PSI Fig.5.3	ROM	$O(w{+}v)$	$O(w)$ 160-bit	$O(v)$ 160-bit	$O(v)$ 160-bit	1024
PSI Fig.5.4	ROM	$O(w{+}v)$	$2{\cdot}O(w)$ 512-bit mod 512-bit	$2{\cdot}O(v)$ 512-bit mod 512-bit	$O(v)$ **mults** mod 1024-bit	1024

Table 5.1: Performance Comparison of PSI and APSI protocols secure in semi-honest model.

We highlight a few points in Table 5.1:

- Protocols in [31] and [108] provide, respectively, *mutual* APSI and PSI, *without* data transfer, whereas our techniques present one-way (A)PSI techniques with data transfer.

- Some prior work is secure in the presence of malicious adversaries and/or in the standard model. When analyzing efficiency, we consider—whenever possible—complexity incurred by instantiations in semi-honest model and in ROM.

- In APSI of Figure 5.2, we consider random exponents (i.e., R_s, R_c, and $R_{c:i}$-s) chosen from smaller groups than $\mathbb{Z}_{N/4}$ – i.e., 512-bit instead of 1024-bit exponents. Although a formal proof of security equivalence is left as part of future work, we rely on arguments from [80, 72].
- Modular exponentiations at server-side in the protocol of Figure 5.4 employ smaller exponents/moduli, using the Chinese Remainder Theorem (CRT) [104]. This is possible since the server generates the RSA parameters.

Chapter 6

Private Set Intersection with Linear Complexities Secure in the Malicious Model

In this chapter, we construct PSI and APSI protocols secure in the malicious *model under standard cryptographic assumptions, with both linear communication and computational complexities. Proposed APSI protocol is the first of its kind. Whereas, the new PSI construct is appreciably more efficient than the state-of-the-art.*

6.1 Security in the Malicious Model for Private Set Intersection

PSI protocols presented in Chapter 5 combine efficiency with provable security guarantees, albeit they only consider semi-honest adversaries. Recall (from Section 2.5) that semi-honest players are assumed to faithfully follow all protocol specifications and not to misrepresent any information related to their inputs, e.g., set size and content. However, during or after protocol execution, they might (passively) attempt to infer additional information about the other participant's input. This model is formalized by considering an ideal implementation where a Trusted Third Party (TTP) receives the inputs of both participants and outputs the result of the defined function. Security in the presence of semi-honest adversaries requires that, in the real implementation of the protocol (without a TTP), each participant does not learn more information than in the ideal implementation.

Therefore, the actual “degree” of security provided by PSI protocols secure against semi-honest adversaries may depend on the specific setting. For example, it may also be reasonable to consider only semi-honest adversaries if participants are subject to auditing and could face severe penalties for non-compliance. In semi-honest model, security focuses on privacy guarantees: the client cannot learn any information about the server’s set, beyond the intersection, while the server learns nothing about client’s set. This notion is usually referred to as *input privacy*.

However, when players are “allowed” to deviate, protocol *correctness* may be affected. Ideally, the output of an honest player should depend only on its input and implicit input used by the adversary. Whereas, it is unclear whether a malicious adversary, even without violating other player’s input privacy, can violate protocol correctness. For instance, can the server, in PSI, force the client’s output to always include the first item in set inputted by the client? If this happens, the protocol fails to guarantee correctness and, arguably, *privacy of client’s output*.

In contrast, the classic formulation of security in the malicious model involves realizing an ideal functionality, that, in turn, requires (1) simulation of any real-world attack into an ideal-world attack, and (2) that outputs of honest players should not differ in the two worlds. Therefore, security in the malicious model captures, simultaneously, correctness and simulation [62], and thus provides the same security as general two-party computation.

Security in the presence of malicious adversaries, however, does not prevent them from refusing to participate in the protocol, modifying their private input sets, or prematurely aborting the protocol. Thus, we still need mechanisms to enforce *authorization* of inputs. This motivates the need for malicious-secure protocols for APSI.

Roadmap

As discussed in Sections 3.3 several PSI and APSI protocols have been proposed, that are secure in the malicious model [108, 85, 44, 31, 27, 87, 100]. Only [98] constructed linear-complexity PSI in the malicious setting under standard assumptions, whereas, [100] is (adaptively) secure under the OneMore-Gap-DH assumption [14]. Also note that proofs in [98] require the domain of inputs to be restricted to polynomial in the security parameter. [98] also requires a Common Reference String model (CRS) — where the reference string, including a safe RSA modulus, must be generated by a mutually trusted third party.

Our starting point are linear-complexity protocols in Chapter 5, that are secure only in semi-honest model. First, we modify the APSI construct in Section 5.3.2 and obtain APSI secure in the malicious model, under the standard RSA (and DDH) assumption (in ROM). Then, we modify its PSI counterpart (from Section 5.3.3): while the linear-complexity PSI protocol in [52] is secure under the One-More-Gap-DH assumption [14] against semi-honest adversaries, our modified variant is secure in the malicious model under the standard DDH assumption (again, in ROM). We present formal proofs for all proposed protocols.

6.2 Linear-Complexity APSI Secure in the Malicious Model

We now present our protocol for secure computation of authorized set intersection. First, we review the definition of the APSI ideal functionality. APSI employs an (off-line) CA with algorithms (KGen, Sign, Vrfy). The CA generates a key-pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}$, publishes its public key pk, and, on client input c_i, it issues a signature $\sigma_i = \mathsf{Sign}_{\mathsf{sk}}(c_i)$ such that $\mathsf{Vrfy}_{\mathsf{pk}}(\sigma_i, c_i) = 1$.

Definition 6.1. *The ideal functionality $\mathcal{F}_{\mathsf{APSI}}$ of an APSI protocol between a server, on input $\mathcal{S} = \{s_1, \ldots, s_w\}$, and a client, on input $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$, is defined as:*

$$\mathcal{F}_{\mathsf{APSI}} : ((\mathcal{S}, v), (\mathcal{C}, w)) \rightarrow (\perp, \mathsf{ASI})$$

where $\mathsf{ASI} \overset{def}{=} \{s_j \in \mathcal{S} \mid \exists\, (c_i, \sigma_i) \in \mathcal{C} \text{ s.t. } c_i = s_j \wedge \mathsf{Vrfy}_{\mathsf{pk}}(\sigma_i, c_i) = 1\}$ *and* pk *is CA's public key.*

We start from the APSI protocol presented in Section 5.3.3, secure in the presence of *semi-honest* adversaries. We describe a modified version that securely implements the $\mathcal{F}_{\mathsf{APSI}}$ functionality in the presence of ***malicious*** adversaries, in ROM, under the RSA and DDH assumptions.

Resulting APSI protocol is illustrated in **Figure 6.1**. The (off-line) trusted third party authorizing client's input (CA) is realized with the following algorithms:

- $\mathsf{KGen}(1^\tau)$: On input of security parameter τ, this algorithm generates safe RSA modulus $N = pq$, where $p = 2p' + 1$, $q = 2q' + 1$ (with p', q' primes), and picks random elements g, g', s.t. $\langle -1 \rangle \times \langle g \rangle \equiv \langle -1 \rangle \times \langle g' \rangle \equiv \mathbb{Z}_N^*$. RSA exponents (e, d) are chosen in the standard way: e is a small prime and $d = e^{-1} \bmod \phi(N)$. The algorithm also fixes hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ and $H_2 : \mathbb{Z}_N^* \times \mathbb{Z}_N^* \times \{0,1\}^* \rightarrow \{0,1\}^\tau$. The secret key is (p, q, d) and the public parameters are: $N, e, g, g', H_1(\cdot), H_2(\cdot)$.

- $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$: On input of c_i, this algorithm issues an authorization $\sigma_i = H_1(c_i)^d \bmod N$.

- $\mathsf{Vrfy}_{\mathsf{pk}}(\cdot, \cdot)$: On input of (σ_i, c_i), this algorithm verifies whether ${\sigma_i}^e = H_1(c_i) \bmod N$.

Also note that the server uses a random permutation $\Pi(\cdot)$ over set $\mathcal{S}$. The goal is to prevent the client from inferring additional information over $\mathcal{S}$ in case it has some knowledge on the *order* of items in $\mathcal{S}$.

[Common input: $N, e, g, g', H_1(\cdot), H_2(\cdot)$]

Client, on input: $(\mathcal{C}, \mathcal{C}_\sigma)$, where
$\mathcal{C} = \{c_1, \ldots, c_v\}, \mathcal{C}_\sigma = \{\sigma_1, \ldots, \sigma_v\}$
$(\forall i\ 1 \leq i \leq v$: $\sigma_i = H_1(c_i)^d \bmod N)$

Server, on input: $\mathcal{S} = \{s_1, \ldots, s_w\}$

```
For $i = 1, \ldots, v$:
    $R_{c:i} \leftarrow_r \mathbb{Z}_{N/2}, (b_i, \bar{b}_i) \leftarrow_r \{0,1\} \times \{0,1\}$
    $M_i = (-1)^{b_i} \cdot \sigma_i \cdot g^{R_{c:i}}$
    $N_i = (-1)^{\bar{b}_i} \cdot H_1(c_i) \cdot (g')^{R_{c:i}}$
$\pi = $ ZK$\{R_{c:i}, i = 1, \ldots, v \mid$
        ${M_i}^{2e}/{N_i}^2 = (g^e/g')^{2R_{c:i}}\}$
```

$$\xrightarrow[\{N_1, \ldots, N_v\}, \pi]{\{M_1, \ldots, M_v\}}$$

```
If $\pi$ doesn't verify, then abort
$(\hat{s}_1, \ldots, \hat{s}_w) \leftarrow \Pi(\mathcal{S})$
$R_s \leftarrow_r \mathbb{Z}_{N/2}$, $Z = g^{2eR_s}$
For $i = 1, \ldots, v$
    $M'_i = (M_i)^{2eR_s}$
For $j = 1, \ldots, w$
    $K_{s:j} = (H_1(\hat{s}_j))^{2R_s}$
    $T_{s:j} = H_2(K_{s:j}, H_1(\hat{s}_j), \hat{s}_j)$
$\pi' = $ ZK$\{R_s \mid Z = g^{2eR_s}$
        $\forall i, M'_i = (M_i)^{2eR_s}\}$
```

$$\xleftarrow[\{T_{s:1}, \ldots, T_{s:w}\}, \pi']{Z, \{M'_1, \ldots, M'_v\}}$$

```
If $\pi'$ doesn't verify, then abort
For $i = 1, \ldots, v$:
    $K_{c:i} = M'_i \cdot Z^{-R_{c:i}}$
    $T_{c:i} = H_2(K_{c:i}, H_1(c_i), c_i)$
```

Output: $\{c_i \mid c_i \in \mathcal{C}$ and $\exists\, T_{s:j}$ s.t. $T_{s:j} = T_{c:i}\}$

[All Computation is $\bmod N$]

Figure 6.1: APSI protocol with linear complexities secure in the malicious model, in ROM, under the RSA and DDH assumptions.

Theorem 6.2.1. *If RSA and DDH problems are hard, and π, π' are zero-knowledge proofs, then the protocol in Figure 6.1 is a secure computation of $\mathcal{F}_{\mathsf{APSI}}$, in ROM.*

Proof. The proof starts with building a simulator from a malicious server and proves that the server has indistinguishable views (and outputs) when interacting with the simulator or with the real client. It then builds a simulator from a malicious client.
[*Construction of an ideal world SIM_s from a malicious real-world server S^**]
The simulator SIM_s is built as follows:

- ***Setup:*** SIM_s executes KGen and publishes public parameters N, e, g, g'.
- ***Hash queries to H_1 and H_2:*** SIM_s constructs two tables $\Upsilon_1 = (q, h_q)$ and $\Upsilon_2 = ((k, h'_q, q'), t)$ to answer, respectively, the H_1 and H_2 queries. Specifically:
  - On query q to H_1, SIM_s checks if $\exists (q, h_q) \in \Upsilon_1$: If so, it returns h_q, otherwise it responds $h_q \leftarrow_r \mathbb{Z}_N^*$, and stores (q, h_q) in Υ_1.
  - On query (k, h'_q, q') to H_2, SIM_s checks if $\exists ((k, h'_q, q'), t) \in \Upsilon_2$: If so, it returns t, otherwise it responds $t \leftarrow_r \{0,1\}^\tau$ to H_2, and stores $((k, h'_q, q'), t)$ to Υ_2.
- ***Simulation of the real-world client C and the ideal-world server $\overline{S}$:***
  1. SIM_s picks $M'_i \leftarrow_r \mathbb{Z}_N^*$, $N'_i \leftarrow_r \mathbb{Z}_N^*$ and computes $M_i = (M'_i)^2$, $N_i = (N'_i)^2$ for each $i = 1, \ldots, v$.
  2. SIM_s sends $\{M_i, N_i\}_{i=1,\ldots,v}$ and simulates the proof π.
  3. After getting $(Z, \{M'_i\}_{i=1,\ldots,v}, \{T_{s:j}\}_{j=1,\ldots,w})$, and interacting with S^* as verifier in the proof π', if the proof π' verifies, SIM_s runs the extractor algorithm for R_s. Otherwise, it aborts.
     (a) For each $T_{s:j}$, SIM_s checks if $\exists (q, h_q) \in \Upsilon_1$ and $\exists ((k, h'_q, q'), t) \in \Upsilon_2$, s.t. $q = q'$, $h_q = h'_q, k = (h_q)^{2R_s}$ and $t = T_{s:j}$. If so, add q to $\mathcal{S}$; otherwise, add a dummy item into $\mathcal{S}$.
     (b) Then SIM_s plays the role of the ideal-world server, that uses $\mathcal{S}$ to respond to ideal client $\overline{C}$'s queries.

Since the distribution of $\{M_i, N_i\}_{i=1,\ldots,v}$ sent by SIM_s is identical to the distribution produced by the real client C and the π proof system is zero-knowledge, S^*'s views when interacting with the real client C and with the simulator SIM_s are indistinguishable.

[*Output of (honest) real client C interacting with S^**]

Now we consider the output of the honest real client C interacting with S^*. By soundness of proof π', message Z and M'_i sent by S^* is $Z = g^{eR_s}$ and $M'_i = (M_i)^{eR_s}$ for $i = 1, \ldots, v$. Then, C's final output is a set containing all c_i's, such that $H_2(M'_i \cdot Z^{-R_{c:i}}, H_1(c_i), c_i) \in \{T_{s:j}\}$. In other words, for each c_i, C outputs c_i if $\exists\, j$ s.t. $H_2(M'_i \cdot Z^{-R_{c:i}}, H_1(c_i), c_i) = T_{s:j}$. Since H_2 is a random oracle, there are two possibilities:

1. S^* computes $T_{s:j}$ from $H_2((H_1(\hat{s}_j))^{2R_s}, H_1(\hat{s}_j), \hat{s}_j)$ for $\hat{s}_j = c_i$. Since SIM_s described above extracts $\hat{s}_j = c_i$ and adds $\hat{s}_j$ in $\mathcal{S}$, the ideal world $\overline{C}$ also output c_i on its input c_i.

2. S^* did not query H_2 on $(M'_i \cdot Z^{-R_{c:i}}, H_1(c_i), c_i)$ but $H_2(M'_i \cdot Z^{-R_{c:i}}, H_1(c_i), c_i)$ happens to be equal to $T_{s:j}$. This event occurs with negligible probability bounded by $v \cdot\ w \cdot\ 2^{-\tau}$.

Therefore, with probability $1 - v \cdot\ w \cdot\ 2^{-\tau}$, the real-world client C interacting with S^* and the ideal-world client $\overline{C}$ interacting with SIM_s yield identical outputs.

[*Construction of an ideal world SIM_c from a malicious real-world client C^**]

The simulator SIM_c is built as follows:

- ***Setup and hash queries to H_1 and H_2:*** Same as Setup and H_1 and H_2 responses described above in construction of SIM_s.

- ***Authorization queries:*** On input m, SIM_c responds with (m, σ) where $\sigma = H_1(m)^d$ and stores (m, σ) in another table, Υ_3.

- ***Simulation of real-world server S and ideal-world client $\overline{C}$:***

  1. After getting $\{M_i, N_i\}_{i=1,\ldots,v}$, and interacting with C^* as verifier in the proof π, SIM_c checks if proof π verifies. If not, it aborts. Otherwise, it runs the extractor algorithm for $\{R_{c:i}\}$ and computes $\pm(H_1(c_i), \sigma_i)$ s.t. $H_1(c_i) = {\sigma_i}^e$.
  2. For each $\pm(H_1(c_i), \sigma_i)$:
     - If $\nexists (q, h_q) \in \Upsilon_1$ s.t. $h_q = \pm H_1(c_i)$ then add a dummy item (δ, σ_δ) to $\mathcal{C}$ where δ and σ_δ are randomly selected from the respective domain.
     - If $\exists (q, h_q) \in \Upsilon_1$ s.t. $h_q = \pm H_1(c_i)$, but $\nexists (m, \sigma) \in \Upsilon_3$ s.t. $\sigma = \pm\sigma_i$ then output fail_1 and abort.
     - If $\exists (q, h_q) \in \Upsilon_1$ s.t. $h_q = \pm H_1(c_i)$ and $\exists (m, \sigma) \in \Upsilon_3$ s.t. $\sigma = \pm\sigma_i$, then add $(q, \pm\sigma)$ to the set $\mathcal{C}$.

3. SIM_c plays the role of the client in the ideal-world. On input $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$, SIM_c interacts with the ideal-world server $\bar{S}$ through the TTP.
4. On getting intersection $L = \{c'_1, \ldots, c'_{|L|}\}$, with $|L| \leq v$ from the ideal-world interaction, SIM_c forms $\mathcal{S}$= $\Pi\left(c'_1, \ldots, c'_{|L|}, \delta'_1, \ldots, \delta'_{w-|L|+1}\right)$, where δ''s are dummy items.
5. SIM_c picks $R_s \leftarrow_r \mathbb{Z}_{N/2}$, and computes $Z = g^{2eR_s}$ and $M'_i = (M_i)^{2eR_s}$ for $i = 1, ..., v$.
6. For each $\hat{s}_j \in \mathcal{S}$:
   - If $\hat{s}_j \in L$, compute $T_{s:j} = H_2((H_1(\hat{s}_j))^{2R_s}, H_1(\hat{s}_j), \hat{s}_j)$.
   - If $\hat{s}_j \notin L$, compute $T_{s:j} \leftarrow_r \{0,1\}^{\tau}$.
7. SIM_c returns $Z, \{M'_i\}_{i=1,\ldots,v}, \{T_{s:j}\}_{j=1,\ldots,w}$ to C^* and simulates the proof π'.

Claim 1. If event fail_1 occurs with non-negligible probability, then C^* can be used to break the RSA assumption.

We describe the reduction algorithm using a modified simulator algorithm called $\mathcal{C}h_1$ that takes an RSA challenge (N', e', z) as an input and tries to output $z^{(e')^{-1}}$. $\mathcal{C}h_1$ follows the SIM_c as described above, except:

- ***Setup:*** On input (N', e', z), $\mathcal{C}h_1$ sets $N = N'$, $e = e'$ and picks generator g, $g' \leftarrow_r \mathbb{Z}_N^*$. (Note that random g in $\mathbb{Z}_N^*$ matches that chosen by a real key generation with probability about 1/2.)

- ***Authorization queries:*** On input m, $\mathcal{C}h_1$ responds with (m, σ) with $\sigma \leftarrow_r \mathbb{Z}_N^*$, assign $H_1(m) = \sigma^e$, and records (m, σ) to Υ_3.

- ***Hash queries to*** H_1***:*** On query H_1 on q, if $\nexists(q, h_q) \in \Upsilon_1$ then $\mathcal{C}h_1$ responds $h_q = z(r_q)^e$ where $r_q \leftarrow_r \mathbb{Z}_N$, and stores (q, r_q, h_q) in Υ_1. (Since r_q is uniformly distributed in $\mathbb{Z}_N$, the distribution of h_q is also uniformly distributed in $\mathbb{Z}_N$.)

Assume that fail_1 occurs on $(H_1(c_i), \sigma_i)$. Then, $\mathcal{C}h_1$ extracts entry $(q, r_q, h_q) \in \Upsilon_1$ s.t. $h_q = H_1(c_i)$ and outputs σ_i / r_q, thus, breaking the RSA assumption.

Unless the fail_1 event occurs, the views interacting with the SIM_c and with the real protocol are different only in the computation of $T_{s:j}$ for $\hat{s}_j \in \mathcal{S}$ but $\hat{s}_j \notin L$. Let fail_2 be the event that C^* queries H_2 on $((H_1(\hat{s}_j))^{2R_s}, H_1(\hat{s}_j), \hat{s}_j)$ for $\hat{s}_j \in \mathcal{S}$ and $\hat{s}_j \notin L$.

Claim 2. If event fail_2 occurs with non-negligible probability, then C^* can be used to break the DDH assumption.

We describe reduction algorithm $\mathcal{C}h_2$ that takes a DDH challenge $(N', f, \alpha = f^a \pmod{N'}, \beta = f^b \pmod{N'}, \gamma)$ as input and outputs the DDH answer using C^*. $\mathcal{C}h_2$ follows the SIM_c algorithm as we describe above, except that:

- ***Setup:*** On input $(N', f, \alpha, \beta, \gamma)$, $\mathcal{C}h_2$ sets $N = N'$, $g = f$ and picks generator $g' \leftarrow_r \mathbb{Z}_N^*$ and odd $e \leftarrow_r \mathbb{Z}_N$.
- ***Authorization queries:*** Same as in $\mathcal{C}h_1$ simulation.
- ***Hash queries to*** H_1***:*** On query q to H_1, if $\nexists(q, h_q) \in \Upsilon_1$ then $\mathcal{C}h_2$ responds with $h_q = \beta g^{r_q}$ where $r_q \leftarrow_r \mathbb{Z}_{N/2}$, and records (q, r_q, h_q) to Υ_1. (Since r_q is random $\mathbb{Z}_N/2$, the distribution of h_q is computationally indistinguishable from the uniform distribution of $\mathbb{Z}_N^*$.)
- ***In computation for*** $Z, \{M_i\}, \{T_{s:j}\}$***:***
  - $\mathcal{C}h_2$ sets $Z = A^{2e}$ and computes $M_i' = \gamma^2(\alpha)^{2r_q+2eR_{c:i}}$ for $i = 1, \ldots, v$ (instead of picking R_s and computing $Z = g^{2eR_s}$ and $M_i' = (M_i)^{2eR_s}$).
  - For each $\hat{s}_j \in \mathcal{S}$, if $\hat{s}_j \in L$, $\mathcal{C}h_2$ computes $T_{s:j} = H_2(\gamma^2(\alpha)^{2r_q}, H_1(\hat{s}_j), \hat{s}_j)$.

Given $\alpha = g^a (= g^{R_s})$ and $\beta = g^b$, we replace g^{ab} by γ in the above simulation of M_i and $T_{s:j}$. Thus, C^*'s views when interacting with the real server S and with the simulator $\mathcal{C}h_2$ are indistinguishable under that DDH assumption. Assume that fail_2 occurs, i.e., C^* makes a query to H_2 on $((H_1(\hat{s}_j))^{2R_s}, H_1(\hat{s}_j), \hat{s}_j)$ for $\hat{s}_j \in \mathcal{S}$ but $\hat{s}_j \notin L$. $\mathcal{C}h_2$ checks if $\exists(q, r_q, h_q) \in \Upsilon_1$ and $\exists((k, h_q', q'), t) \in \Upsilon_2$ s.t. $q = q'$, $h_q = h_q'$, $k = \gamma^2(\alpha)^{2r_q}$ for each $q \in \mathcal{S}$ but $q \notin L$. If so, $\mathcal{C}h_2$ outputs True. Otherwise, $\mathcal{C}h_2$ outputs False. Thus, the DDH assumption is broken.

Therefore, since fail_1 and fail_2 events occur with negligible probability, C^*'s view in the protocol with the real-world server S and in the interaction with SIM_c is negligible.

[*The output of honest real server S interacting with C^**]

Finally, the real-world S interacting with C^* in the real protocol outputs $\perp$ and the ideal-world $\bar{S}$ interacting with SIM_c gets $\perp$. This ends proof of Theorem 6.2.1. □

The APSI protocol secure in the malicious model (in Figure 6.1) differs from the one with semi-honest security (in Section 5.3.2) in the following:

- We modify inputs to the protocol and add efficient zero-knowledge proofs to prevent client and server from deviating from the protocol and to enable extraction of inputs.

- We multiply client inputs by -1 or 1, in order to: (1) ensure that they are uniformly distributed in QR_N, and (2) simplify reduction to the RSA problem.

6.3 Deriving Linear-Complexity PSI Secure in the Malicious Model

We now present our protocol for secure computation of authorized set intersection. First, we review the definition of PSI ideal functionality.

Definition 6.2. *The ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ of a PSI between server S on input $\mathcal{S} = \{s_1, \ldots, s_w\}$ and client C on input $\mathcal{C} = \{c_1, \ldots, c_v\}$ is defined as follows:*

$$\mathcal{F}_{\mathsf{PSI}} : ((\mathcal{S}, v), (\mathcal{C}, w)) \mapsto (\perp, \mathcal{S} \cap \mathcal{C})$$

Similar to its APSI counterpart, our new PSI technique builds on the PSI protocol presented in Section 5.3.3 (secure in the presence of semi-honest adversaries). We amend it to obtain a protocol that securely implements $\mathcal{F}_{\mathsf{PSI}}$ in the presence of *malicious* adversaries, under the DDH assumptions (in ROM). We assume that, at setup time, the following public parameters are selected: p, q, g, g', g'', where p and q are primes, such that $q|p-1$ and g, g', g'' are generators of $\mathbb{Z}_q^*$, alongside two hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \times \{0,1\}^* \to \{0,1\}^\tau$.

Resulting protocol is illustrated in **Figure 6.2** and is secure under the DDH assumption in ROM. Once again, the server uses a random permutation $\Pi(\cdot)$ over set $\mathcal{S}$, in order to prevent the client from inferring additional information over $\mathcal{S}$ in case it has some knowledge on the *order* of items in $\mathcal{S}$.

Theorem 6.3.1. *If the DDH problem is hard and π, π' are zero-knowledge proofs, the protocol in Figure 6.2 is a secure computation of $\mathcal{F}_{\mathsf{PSI}}$, in ROM.*

Proof. Once again, we construct a simulator from a malicious server (resp., client) and prove that the server (resp., client) has indistinguishable views (and outputs) when interacting with the simulator or with the real client (resp., server).

[*Construction of an ideal world SIM_s from malicious real-world server S^**]

Simulator SIM_s is built as follows:

- ***Setup:*** SIM_s generates and publishes public parameters p, q, g, g', g''.

- ***Queries*** H_1 ***and*** H_2***:*** SIM_s creates two tables $\Upsilon_1 = (q, h_q)$ and $\Upsilon_2 = ((k, h'_q, q'), t)$ to answer, respectively, H_1 and H_2 queries. Specifically, [Common input: $p, q, g, g', g'', H_1(\cdot), H_2(\cdot)$]

Client, on input: $\mathcal{C} = \{c_1, \dots, c_v\}$

$(\forall\, i = 1, \dots v)\;\; PCH_i = \prod_{l \neq i}(H_1(c_i))$

$PCH = \prod_{i=1}^{v}(H_1(c_i))$

$R_c \leftarrow_r \mathbb{Z}_q,\, X = PCH \cdot (g)^{R_c}$

For $1 \leq i \leq v$

- $R_{c:i} \leftarrow_r \mathbb{Z}_q$
- $M_i = H_1(c_i) \cdot (g')^{R_{c:i}}$
- $N_i = PCH_i \cdot (g'')^{R_{c:i}}$

$\pi = \text{ZK}\{R_c, R_{c:i}, i = 1, \dots, v \mid$
$\quad X/(M_i N_i) = g^{R_c}/(g' \cdot g'')^{R_{c:i}}\}$

$$\xrightarrow[\{N_1, \dots, N_v\}, \pi]{X, \{M_1, \dots, M_v\}}$$

Server, on input: $\mathcal{S} = \{s_1, \dots, s_w\}$

If π doesn't verify, then abort

$(\hat{s}_1, \dots, \hat{s}_v) \leftarrow \Pi(\mathcal{S})$

$R_s \leftarrow_r \mathbb{Z}_q,\, Z = (g')^{R_s}$

For $i = 1, \dots, v$

- $M'_i = (M_i)^{R_s}$

For $j = 1, \dots, w$

- $K_{s:j} = (H_1(\hat{s}_j))^{2R_s}$
- $T_{s:j} = H_2(K_{s:j}, H_1(\hat{s}_j), \hat{s}_j)$

$\pi' = \text{ZK}\{R_s \mid Z = (g')^{R_s}$
$\quad \forall i, M'_i = (M_i)^{R_s}\}$

$$\xleftarrow[\{T_{s:1}, \dots, T_{s:w}\}, \pi']{Z, \{M'_1, \dots, M'_v\},}$$

If π' doesn't verify, then abort

For $i = 1, \dots, v$:

- $K_{c:i} = M'_i \cdot Z^{-R_{c:i}}$
- $T_{c:i} = H_2(K_{c:i}, H_1(c_i), c_i)$

Output: $\{c_i \mid c_i \in \mathcal{C} \text{ and } \exists\, T_{s:j} \text{ s.t. } T_{s:j} = T_{c:i}\}$

[All Computation is $\bmod\ p$]

Figure 6.2: PSI protocol with linear complexities secure in the malicious model, in ROM, under the DDH assumption.

- On query q to H_1, SIM_s checks if $\exists(q, h_q) \in \Upsilon_1$: If so, it returns h_q, otherwise it responds $h_q \leftarrow_r \mathbb{Z}_p^*$, and stores (q, h_q) in Υ_1.

– On query (k, h'_q, q') to H_2, SIM_s checks if $\exists((k, h'_q, q'), t) \in \Upsilon_2$: If so, it returns t, otherwise it responds $t \leftarrow_r \{0,1\}^\tau$ to H_2, and stores $((k, h'_q, q'), t)$ to Υ_2.

- ***Simulation of real-world client C and ideal-world server $\overline{S}$:***

  1. SIM_s picks $X \leftarrow_r \mathbb{Z}_p^*$ and $\{M_i, N_i \mid M_i \leftarrow_r \mathbb{Z}_p^*, N_i \leftarrow_r \mathbb{Z}_p^*\}$ (for $i = 1, \ldots, v$).
  2. SIM_s sends $X, \{M_i, N_i\}_{i=1,\ldots,v}$ and simulates proof π.
  3. After getting $(Z, \{M'_i\}_{i=1,\ldots,v}, \{T_{s:j}\}_{j=1,\ldots,w})$, and interacting with S^* as verifier in proof π', if π' verifies, SIM_s runs the extractor algorithm for R_s. Otherwise, it aborts.
     (a) For each $T_{s:j}$, SIM_s checks if $\exists(q, h_q) \in \Upsilon_1$ and $\exists((k, h'_q, q'), t) \in \Upsilon_2$, s.t. $q = q'$, $h_q = h'_q, k = (h_q)^{R_s}$ and $t = T_{s:j}$. If so, add q to $\mathcal{S}$; otherwise, add a dummy item into $\mathcal{S}$.
     (b) Then SIM_s plays the role of the ideal-world server, that uses $\mathcal{S}$ to respond to ideal client $\overline{C}$'s queries.

Since the distribution of $X, \{M_i, N_i\}_{i=1,\ldots,v}$ sent by SIM_s is identical to the distribution produced by the real client C and the π proof system is zero-knowledge, S^*'s views when interacting with real client C and with simulator SIM_s are indistinguishable.

[*Output of the honest real client C interacting with S^**]

Now we consider output of honest real client C interacting with S^*. By soundness of π', message Z and M'_i sent by S^* is $Z = (g')^{R_s}$ and $M'_i = (M_i)^{R_s}$ for $i = 1, \ldots, v$. Then C's final output is a set containing all c_i's such that $H_2(M'_i Z^{-R_{c:i}}, H_1(c_i), c_i) \in \{T_{s:j}\}$. In other words, for each c_i, C outputs c_i if $\exists\ j$ s.t. $H_2(M'_i Z^{-R_{c:i}}, H_1(c_i), c_i) = T_{s:j}$. Since H_2 is a random oracle, there are two possibilities:

1. S^* computes $T_{s:j}$ from $H_2((H_1(\hat{s}_j))^{2R_s}, H_1(\hat{s}_j), \hat{s}_j)$ for $\hat{s}_j = c_i$. Since SIM_s described above extracts $\hat{s}_j = c_i$ and adds $\hat{s}_j$ in $\mathcal{S}$, ideal world $\overline{C}$ also output c_i on its input c_i.
2. S^* did not query H_2 on $(M'_i Z^{-R_{c:i}}, H_1(c_i), c_i)$ but $H_2(M'_i Z^{-R_{c:i}}, H_1(c_i), c_i)$ happens to be equal to $T_{s:j}$. This event occurs with negligible probability bounded by $v \cdot\ w \cdot\ 2^{-\tau}$.

Therefore, with probability $1 - v \cdot\ w \cdot\ 2^{-\tau}$, real-world client C interacting with S^* and ideal-world client $\overline{C}$ interacting with SIM_s produce identical output.

[*Construction of ideal world SIM_c from malicious real-world client C^**]

Simulator SIM_c is built as follows:

- ***Setup and hash queries to H_1 and H_2:*** Same as Setup and H_1 and H_2 responses described above in construction of SIM_s.

- ***Simulation of real-world server S and ideal-world client $\overline{C}$:***

  1. After getting $(X, \{M_i\}, \{N_i\})$, and interacting with C^* as verifier in proof π, SIM_c checks if π verifies. If not, it aborts. Otherwise, it runs the extractor algorithm for $R_c, \{R_{c:i}\}$ and computes $H_1(c_1), \ldots, H_1(c_v)$.
  2. For each $H_1(c_i)$, if $\exists(q, h_q) \in \Upsilon_1$ s.t. $h_q = H_1(c_i)$ then add q to the set $\mathcal{C}$. Otherwise, add a dummy item to $\mathcal{C}$.
  3. SIM_c plays the role of the client in the ideal-world. On input $\mathcal{C} = \{c_1, \ldots, c_v\}$, SIM_c interacts with the ideal-world server $\bar{S}$ through the TTP.
  4. On getting intersection $L = \{c'_1, \ldots, c'_{|L|}\}$, with $|L| \le v$ from the ideal-world interaction, SIM_c forms $\mathcal{S}$= $\Pi\left(c'_1, \ldots, c'_{|L|}, \delta'_1, \ldots, \delta'_{w-|L|+1}\right)$, where δ''s are dummy items and Π is a permutation function.
  5. SIM_c picks $R_s \leftarrow_r \mathbb{Z}_q$, and computes $Z = g^{R_s}$ and $M'_i = (M_i)^{R_s}\}$ for $_{i=1,\ldots,v}$.
  6. For each $\hat{s}_j \in \mathcal{S}$:
     - If $\hat{s}_j \in L$, compute $T_{s:j} = H_2((H_1(\hat{s}_j))^{R_s}, H_1(\hat{s}_j), \hat{s}_j)$.
     - If $\hat{s}_j \notin L$, compute $T_{s:j} \leftarrow_r \{0,1\}^\tau$.
  7. SIM_c returns $Z, \{M'_i\}, \{T_{s:j}\}$ to C^* and simulates proof π'.

Let fail be the event that C^* queries H_2 on $((H_1(\hat{s}_j))^{R_s}, H_1(\hat{s}_j), \hat{s}_j)$ for $\hat{s}_j \in \mathcal{S}$ and $\hat{s}_j \notin L$. Similar to the argument in the proof of Theorem 6.2.1, if fail event does not occur, since the π' is zero-knowledge, we argue that C^*'s views in the real game with real-world server S and in the interaction with simulator SIM_c constructed above are indistinguishable .

Claim 3. If event fail occurs with non-negligible probability, then C^* can be used to break the DDH assumption.

We describe the reduction algorithm called $\mathcal{C}h$ that takes a DDH problem $(p', q', f, \alpha = f^a \pmod{p'}$, $\beta = f^b \pmod{p'}, \gamma)$ as an input and tries to output the answer using C^*. $\mathcal{C}h$ follows the SIM_c algorithm as we describe above, except that:

- ***Setup:*** On input $(p', q', f, \alpha, \beta, \gamma)$, $\mathcal{C}h_2$ sets $p = p'$, $q = q'$, $g' = f$ and picks generator g, $g'' \leftarrow_r \mathbb{Z}_q^*$.

- ***Hash queries to*** H_1***:*** On query q to H_1, if $\nexists(q, h_q) \in \Upsilon_1$ then $\mathcal{C}h_2$ responds with $h_q = \beta(g')^{r_q}$ where $r_q \leftarrow_r \mathbb{Z}_q$, and records (q, r_q, h_q) to Υ_1.
- ***In computation for*** $Z, \{M'_i\}, \{T_{s:j}\}$***:***
  - $\mathcal{C}h_2$ sets $Z = A$ and computes $M'_i = C(A)^{r_q + R_{c:i}}$.
  - For each $\hat{s}_j \in \mathcal{S}$, if $\hat{s}_j \in L$, $\mathcal{C}h_2$ computes $T_{s:j} = H_2(C(A)^{r_q}, H_1(\hat{s}_j), \hat{s}_j)$.

Using an argument similar to that in the proof of Theorem 1, C^*'s views, when interacting with real server S and with simulator $\mathcal{C}h_2$, are indistinguishable under the DDH assumption. Assume that fail occurs, i.e., C^* makes a query to H_2 on $((H_1(\hat{s}_j))^{R_s}, H_1(\hat{s}_j), \hat{s}_j)$ for $\hat{s}_j \in \mathcal{S}$ but $\hat{s}_j \notin L$. $\mathcal{C}h$ checks if $\exists(q, r_q, h_q) \in \Upsilon_1$ and $\exists((k, h'_q, q'), t) \in \Upsilon_2$ s.t. $q = q'$, $h_q = h'_q$, $k = C(A)^{r_q}$ for each $q \in \mathcal{S}$ and $q \notin L$. If so, $\mathcal{C}h$ outputs True. Otherwise, $\mathcal{C}h_2$ outputs False. Thus, $\mathcal{C}h$ solves the DDH problem.

Since fail occurs with negligible probability, C^*'s view in the protocol with the real-world server S and in interaction with SIM_c is negligible.

[*Output of honest real server S interacting with C^**]

Finally, real-world S interacting with C^* in the real protocol outputs $\perp$ and ideal-world $\bar{S}$ interacting with SIM_c also gets $\perp$. This ends the proof of Theorem 6.3.1. □

Supporting (A)PSI with Data Transfer

It is easy to add the *data transfer* functionality to the protocols in Figure 6.1 and 6.2. We assume that an additional secure cryptographic hash function $H_3 : \{0,1\}^* \to \{0,1\}^\tau$ is chosen during setup. In either case (PSI or APSI), we then use $H_3(\cdot)$ to derive a symmetric key for a CPA-secure symmetric cipher, such as AES [45], used in the appropriate mode of operation. For every $j = 1, \ldots, w$, the server computes $k_{s:j} = H_3(K_{s:j})$ and encrypts associated data using a distinct key $k_{s:j}$. For its part, the client, for every $i = 1, \ldots, v$, computes $k_{c:i} = H_3(K_{c:i})$ and decrypts ciphertexts corresponding to the matching tag. (Note that $ks_j = kc_i$ if and only if $\hat{s}_j = c_i$ and so $T_{s:j} = T_{c:i}$). As long as the underlying encryption scheme is CPA-secure, this extension does not affect security or privacy arguments for any protocol discussed thus far. Also, it leaves the complexity of the protocols unaltered.

This technique is same as the one used in Section 5.5 to support data transfer in (A)PSI protocols with semi-honest security.

6.4 Efficiency

We now analyze the efficiency of our protocols secure in the malicious model and compare them to prior results. Protocol features and estimated asymptotic complexities in Table 6.1. For each protocol, we choose choose parameters that achieve 80-bit security. We distinguish between *offline* (i.e., pre-computation) and *online* operations.

Protocol	**Model**	**Assumptions**	**Pre-Comp.**	**Server Comput.**	**Client Comput.**	**Mod**
[31]	Std	Strong RSA	-	$O(wv)$ 160-bit	$O(wv)$ 160-bit	1024
APSI Fig.6.1	ROM	RSA,DDH	$O(w)$ 1024-bit	$O(v)$ 1024-bit	$O(v)$ 1024-bit	1024
PSI in [66]	ROM	Hom. Enc.	-	$O(w \log\log v)$ 160-bit	$O(v{+}w)$ 160-bit	1024
PSI in [87]	Std	DDH	-	$O(v{+}w(\log\log v))$ 160-bit	$O(v{+}v)$ 160-bit	1024
PSI in [98]	Std	CRS,CDR	$O(w)$ 1024-bit	$O(v)$ 1024-bit	$O(v)$ m-bit	2048
PSI in [100]	ROM	OneMoreDH	$O(w)$ 160-bit	$O(v)$ 160-bit	$O(v)$ 160-bit	1024
PSI Fig.6.2	ROM	DDH	$O(w)$ 160-bit	$O(v)$ 160-bit	$O(v)$ 160-bit	1024

Table 6.1: Performance Comparison of PSI and APSI protocols in the malicious model.

Observe that the APSI protocol (in Figure 6.1) is, to the best of our knowledge, the only such construct, secure in the malicious model, with *linear* communication and computational complexity. Moreover, our PSI protocol (in Figure 6.2) achieves the same complexity (linear number of short exponentiations) as the work in [100]. However, the latter is only secure under the OneMore-DH assumption, while ours – under DDH.

Finally, from a conservative stance, we select 1024-bit random exponents in the APSI protocol of Figure 6.1. However, similar to what discussed (in Section 5.6), we could select shorter exponents (e.g., 512-bit), relying on the arguments from [80, 72].

Chapter 7

Size-Hiding Private Set Intersection

In this chapter, we introduce the concept of Size-Hiding in Private Set Intersection, where the size of the set held by one of the participants is hidden from the other. After formal definitions, we present our efficient and provably secure constructions.

7.1 Why Size Matters?

We start with motivating the need for PSI with a stronger privacy property of *hiding the size* of the set held by one entitity. Below, we present three motivating scenarios.

1. U.S. Department of Homeland Security (DHS) maintains a *dynamic* database of suspected terrorists (TWL: Terror Watch List) and, for every flight, it needs to know whether the flight passenger manifest and TWL have any names in common. Since the DHS treats TWL as secret information, and airlines are reluctant to unconditionally share passenger information, entities could employ (A)PSI to let the DHS learn names of any flight passengers that also appear in TWL.
2. U.S. Central Intelligence Agency (CIA) might have requirement to periodically (e.g., every year) check whether any of its agents have been arrested or convicted of any crimes. It thus needs to, ideally, compare its list of employees against each state's list of arrestees and/or convicted offenders. Again, the CIA and the states could run a PSI protocol to protect their respective privacy.

3. U.S. Center of Disease Control and Prevention (CDC) maintains a list of people, per city, afflicted by certain contagious diseases, e.g., the H1N1 virus. The CDC needs to monitor high or unusual concentrations of infected people in schools, since that might indicate the start of an epidemic. To this end, it periodically needs to cross-check its list with student rosters in each school district, and it can do so by employing PSI techniques, in order to comply with privacy regulations.

All examples above have some features in common: neither party can reveal its information in its entirety. What one of them is willing to reveal is limited to common information, i.e., items appearing on both parties' lists. Thus, PSI represents a potential solution to protect privacy of both parties.

Another more subtle feature common to our examples is the need to keep client input size secret. Specifically, the DHS does not reveal the number of names on the TWL. This list is dynamic (names can be added and removed frequently) and revealing its size would leak sensitive information. Likewise, by law, the CIA cannot divulge the number of its agents, for obvious reasons. Finally, the number of infected school-kids in a city (school district) is extremely sensitive: its disclosure can cause wide-spread panic and/or prompt health insurance rate hikes for that location.

Consequently, there are solid reasons to keep sizes of their inputs secret. The most common reason is that client's input size represents sensitive information. A related reason is that, given multiple interactions between the same two parties, fluctuations in input size are equally (or even more) sensitive.

An additional motivating factor is related to the amount of computation imposed on the server. In all current PSI protocols, server's computational complexity always depends on size of client set. Considering that the server obtains no output from the PSI interaction, and that client set may be significantly larger than server's (e.g., in case of TWL and passenger manifests), there seems to be an "unfair" computational burden on the server.

We note, from the outset, that there are limits to input size hiding. For instance, both parties cannot hide their respective input sizes. One obvious reason is that, in all examples above, one party learns the intersection of its input with that of the other party. The intersection itself is a list and its size leaks information about the overall input size.

Size Hiding with Current Tools?

Current PSI protocols disclose participants' input size Also, generic secure multi-party computation tools [148] (discussed in Section 1.2.1) are not applicable as they provide all players with the sizes of other players' inputs.

One trivial approach is for the client to pad its input up to a certain fixed size. However, this has several drawbacks. First and foremost, this always leaks the *upper bound* of input size. Second, if client input is a dynamic set, the fixed size must reflect the maximum possible set size (otherwise, fluctuations would leak information), which entails wasted computation and communication resources.

7.2 SHI-PSI Definitions

Informally, SHI-PSI extends PSI with an additional privacy feature that client input size must not be revealed to the server. Clearly, SHI-PSI implies PSI. We now define the SHI-PSI functionality as well as its security and privacy requirements.

Definition 7.1 (SHI-PSI.). *An interactive protocol satisfying correctness, server privacy and client privacy (per Definitions 7.2, 7.3, 7.4 below), involving client and server, on input, $\mathcal{S} = \{s_1, \cdots, s_w\}$ and $\mathcal{C} = \{c_1, \cdots, c_v\}$, respectively.*

Definition 7.2 (Correctness.). *If both participants follow the protocol on inputs $(\mathcal{S}, \mathcal{C})$, the server outputs $\perp$, and the client outputs $(w, \mathcal{S} \cap \mathcal{C})$.*

We assume semi-honest participants and use general definitions of secure computation given in [77]. Specifically, we define SHI-PSI as a secure two-party protocol realizing the functionality described above. Our client and server privacy definitions follow from those in related work [112, 66, 65, 85]. As stated by [77], in case of semi-honest participants, the general "real-versus-ideal" definition framework is ***equivalent*** to a much simpler framework that extends the formulation of honest-verifier zero-knowledge. Informally, a protocol privately computes certain functionality if whatever can be obtained from one participant's view of a protocol execution can be obtained from input and output of that participant. In other words, the view of a semi-honest participant (including $\mathcal{C}$ or $\mathcal{S}$, all messages received during execution, and the outcome of that participant's internal coin tosses), on each possible input $(\mathcal{C}, \mathcal{S})$, can be efficiently simulated considering only that participant's own input and output. This is equivalent to the following formulation:

Definition 7.3 (Client Privacy.). *For every PPT S^* that plays server's role, for every $\mathcal{S}$, and for any client input set $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)})$, two views of S^* corresponding to client's inputs: $\mathcal{C}^{(0)}$ and $\mathcal{C}^{(1)}$, are computationally indistinguishable.*

Client privacy is guaranteed if no information is leaked about its input. That is, S^* cannot distinguish between $\mathcal{C}^{(0)}$ and $\mathcal{C}^{(1)}$. S^* cannot even determine whether $|\mathcal{C}^{(0)}| \neq |\mathcal{C}^{(1)}|$. In fact, Definition 7.3 is strictly stronger than client privacy definition for PSI protocols that reveal client input size. In this case, indistinguishability would be relaxed by the constraint $|\mathcal{C}^{(0)}| = |\mathcal{C}^{(1)}|$.

Definition 7.4 (Server Privacy.). *Let $\mathsf{View}_C(\mathcal{C}, \mathcal{S})$ be a random variable representing client's view during execution of SHI-PSI with inputs $\mathcal{C}, \mathcal{S}$. There exists a PPT algorithm C^* such that:*

$$\{C^*(\mathcal{C}, \mathcal{S} \cap \mathcal{C})\}_{(\mathcal{C},\mathcal{S})} \overset{c}{\equiv} \{\mathsf{View}_C(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$$

In other words, on each possible pair of inputs $(\mathcal{C}, \mathcal{S})$, client's view can be efficiently simulated by C^* on input: $\mathcal{C}$ and $\mathcal{S} \cap \mathcal{C}$. Thus, as in [77], we claim that the two distributions implicitly defined above are computationally indistinguishable.

Remark. As mentioned earlier, we consider security in the presence of semi-honest participants, This models precisely the class of adversaries considered in our applications. For instance, in one of the examples above, DHS and airlines have no incentive to deviate from protocol specifications, because they might be subject to auditing and could face severe penalties for non-compliance. Nonetheless, airline personnel, system administrators, or other malicious insiders might seek to surreptitiously obtain information about contents or size of the DHS Terror Watch List (TWL).

7.3 SHI-PSI Construction

We now present our SHI-PSI protocol. Its two main building blocks are: (1) RSA accumulators [16], and (2) *unpredictable* function $f_{X,\phi(N)}(y) = (X^{1/y \bmod \phi(N)}) \bmod N$ (under the RSA assumption on safe moduli).[1]

Specifically, the client computes a global witness for its input $\mathcal{C} = \{c_1, \cdots, c_v\}$, in the form of an RSA accumulator: $(g^{\prod_{i=1}^{v} H_1(c_i)}) \bmod N$, where g is a generator of QR_N and $H_1(\cdot)$ is

[1]A function (family) $f_k(\cdot)$ is an $(t, q_f, \epsilon)-$unpredictable if, for any t-time algorithm $\mathcal{A}$ and any auxiliary information z, it holds that: $Pr[(x^*, f_k(x^*) \leftarrow_r \mathcal{A}^{f_k(\cdot)}(z)) \wedge x^* \notin \mathcal{Q}] \leq \epsilon$ where $\mathcal{A}$ makes at most q_f queries to $f_k(\cdot)$, and $\mathcal{Q}$ is the set of queries [100].

a full-domain hash function [15]. Then, the client securely blinds the accumulator with a random exponent and sends the result (denoted as X) to the server. The latter learns no information about client input. For each item $s_j \in \mathcal{S}$, the server computes unpredictable function f over client message X. Server then applies a one-way function (in practice, a suitable cryptographic hash function) to each output of f. The results form a set of so-called *tags*, one for each s_j. These tags are then returned to the client for matching (details below). The outer hash is crucial, since server privacy is based on the fact that, in ROM, a hash of an unpredictable function is a PRF.

Note that $H_1(\cdot)$ is a standard random oracle that does not have to output large primes. Also, we obviate the technical issue of computing the inverse of $H_1(s_j)$ "in the exponent" by selecting the RSA modulus N as a product of safe primes to ensure that the order of X is itself a product of large and unknown primes (see proof for details).

Client learns the set intersection as well as w since it can only match tags corresponding to the items in the intersection. The intuition is that client computation of $g^{(\prod_{l \neq i} H_1(c_l))}$ leads to it finding a matching tag if and only if $c_i \in \mathcal{S} \cap \mathcal{C}$.

7.3.1 Protocol Description

We present the initial SHI-PSI protocol in **Figure 7.1**. Common input is extracted from the output of $\mathsf{RSA\text{-}Gen}(1^\tau)$, reviewed in Section 2.3, for a security parameter τ. Specifically, common input is $N = pq$ where $p = 2p' + 1$ and $q = 2q' + 1$, a generator g of QR_N, as well as two hash functions, $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_N$ (full-domain hash) and $H_2 : \mathbb{Z}_N \rightarrow \{0,1\}^\tau$. Primes p' and q' are known exclusively by the server. Client must treat its exponents as large integers. We emphasize that arithmetic operations in the exponent are performed in $\mathbb{Z}^*_{p'q'}$. (In particular, squaring is a permutation of QR_N, in our setting.) Finally, we use $\Pi(\cdot)$ to indicate a random permutation over a set of values in $\{0,1\}^\tau$.

Theorem 7.3.1. *Under the RSA assumption on safe moduli, the protocol in Figure 7.1 is a server- and client-private SHI-PSI, as per Definition 7.1, in the Random Oracle Model (ROM).*

Proof. We show that the protocol satisfies *correctness, client privacy, and server privacy*, defined in Section 7.2. We assume that all server elements are distinct. Hash functions $H_1(\cdot)$ and $H_2(\cdot)$ are modeled as random oracles.

Correctness. $\forall c_i \in \mathcal{S} \cap \mathcal{C}, \exists\, s_j \in \mathcal{S}\ s.t.\ s_j = c_i$. Hence, $H_1(s_j) = H_1(c_i)$ and

$$K_{s:j} = X^{R_s \cdot 1/H_1(s_j)} = g^{R_c R_s PCH(1/H_1(s_j))} = g^{R_c \cdot PCH_i} = K_{c:i} \tag{7.1}$$

[Common input: $N, g, H_1(\cdot), H_2(\cdot)$ – Server's input (p', q')]

Client, on input: $\mathcal{C} = \{c_1, \ldots, c_v\}$ | **Server**, on input: $\mathcal{S} = \{s_1, \ldots, s_w\}$)

$PCH \overset{\text{def}}{=} \prod_{i=1}^{v} H_1(c_i)$

For $i = 1, \ldots, v$:

$\quad PCH_i \overset{\text{def}}{=} \prod_{l \neq i} H_1(c_l)$

$R_c \leftarrow_r \{1, \ldots, N^2\}$

$X = g^{PCH \cdot R_c} \bmod N$

$\xrightarrow{\quad X \quad}$

$R_s \leftarrow_r \{0, \ldots, p'q' - 1\}$

$Z = (g^{R_s}) \bmod N$

For $j = 1, \ldots, w$:

$\quad K_{s:j} = (X^{R_s \cdot (1/H_1(s_j))}) \bmod N$

$\quad t_j = H_2(K_{s:j})$

$\{T_{s:1}, \ldots, T_{s:w}\} = \Pi(\{t_1, \ldots, t_w\})$

$\xleftarrow{\quad Z, \{T_{s:1}, \ldots, T_{s:w}\}, \quad}$

For $i = 1, \ldots, v$:

$\quad K_{c:i} = (Z^{R_c \cdot PCH_i}) \bmod N$

$\quad T_{c:i} = H_2(K_{c:i})$

Output: $\{c_i \mid c_i \in \mathcal{C} \text{ and } \exists\, T_{s:j} \text{ s.t. } T_{s:j} = T_{c:i}\}$

Figure 7.1: Proposed SHI-PSI protocol.

Consequently, $T_{c:i} = H_2(K_{c:i}) = H_2(K_{s:j}) = T_{s:j}$; thus, the client learns: $c_i \in \mathcal{S} \cap \mathcal{C}$.

Client Privacy. Since client's only message to the server is $X = g^{(PCH \cdot R_c)} \bmod N$, the distribution of X is essentially equivalent to that of random elements in QR_N, which is a cyclic group of order $p'q'$. Since PCH and $p'q'$ are relatively prime (with overwhelming probability), we assume that $g^{PCH} \bmod N$ is a generator of QR_N. Moreover, R_c is chosen uniformly at random from $\{1, \ldots, N^2\}$. Thus, if $R_c = r_1 p'q' + r_2$ with $r_2 \in \{0, \ldots, p'q' - 1\}$, we have that the distribution of r_2 is *statistically* indistinguishable from the uniform distribution on $\{0, \ldots, p'q' - 1\}$ and r_1 and r_2 are essentially independent (see, e.g., [43]). Therefore, $X = g^{PCH \cdot R_c} \bmod N$ is essentially distributed as a random quadratic residue independent of *PCH even if factorization of N is known*.

Server Privacy. To show that client's view can be efficiently simulated by a PPT algorithm, we follow a hybrid argument: The entire client's view is gradually transformed by replacing values (received by the client) that are **outside** the set intersection, with elements chosen uniformly and independently at random. It then suffices to show that this progressive substitution cannot be detected by any efficient algorithm.

Let $I = \mathcal{C} \cap \mathcal{S}$, and $|I| = t$. For any $(\mathcal{C}, \mathcal{S})$, we show that two distributions:

$$\mathcal{D}_0 = \left\{ (R_c, T) \ : \ R_c \leftarrow_r \{1, \dots, N^2\}, T = \Pi\left(H_2(X^{R_s(1/H_1(s_{j1}))}), \cdots, H_2(X^{R_s(1/H_1(s_{jw}))}) \right) \right\}$$

and

$$\mathcal{D}_{w-t} = \left\{ (R_c, T) \ : \ R_c \leftarrow_r \{1, \dots, N^2\}, T = \Pi\left(H_2(X^{R_s(1/H_1(s_{j1}))}), \cdots, r_{t+1}, \cdots, r_w \right) \right\},$$

are computationally indistinguishable, where $(H_1(s_{j1}), \cdots, H_1(s_{jt})) \in I$ and values in $(r_{t+1}, \cdots, r_w)$ are chosen uniformly and independently at random from $\{0,1\}^{\tau_2}$ (i.e., the co-domain of the random oracle $H_2(\cdot)$).

Our proof follows the standard hybrid argument: Let $z = w - t$. We define a series of intermediate distributions $\mathcal{D}_i$, for $0 < i < z$, where T is constructed by replacing the first i outputs of items NOT in I with random values in the co-domain of $H_2(\cdot)$.

After fixing index i and probabilistic polynomial-time distinguisher D, we define:

$$\epsilon(\tau) = |\Pr[D = 1|\mathcal{D}_{i+1}] - \Pr[D = 1|\mathcal{D}_i]|$$

Our claim is that $\epsilon(\tau)$ is negligible in τ. Let us assume that this claim is false. The only difference between $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ is the way T is defined. Specifically, $(i+1)$-st item of T not in I is $H_2(X^{R_s(1/H_1(s_l))})$ for $\mathcal{D}_i$ and a random value for $\mathcal{D}_{i+1}$.

Since $H_2(\cdot)$ is a random oracle, distinguisher D must compute $X^{R_s(1/H_1(s_l))} = g^{R_sR_cPCH/H_1(s_l)}$ for $H_1(s_l) \notin I$. Then, we can build an efficient algorithm $\mathcal{A}$ that, given a challenge (N, e, y), returns $y^{1/e} \bmod N$. (We assume that y is chosen uniformly at random from QR_N. Thus, the order of y is $p'q'$.) The simulation proceeds as follows: First, $\mathcal{A}$ sets $g = y$ and, by programming the random oracle $H_1(\cdot)$, $\mathcal{A}$ assigns random values to outputs of $H_1(\cdot)$ and computes $d = \gcd(R_sR_cPCH, H_1(s_l))$, for some integers e and b with $H_1(s_l) = ed$ and $R_sR_cPCH = bd$. Since $H_2(\cdot)$ is a random oracle, $\mathcal{A}$ sees $g^{R_sR_cPCH/H_1(s_l)} = g^{b/e}$. Given that $(g^{b/e})^e = g^b$ and $\gcd(e, b) = 1$, $\mathcal{A}$ can use the extended Euclidean algorithm to compute $g^{1/e} = y^{1/e}$ via the well-known *Shamir's trick*.[2] Thus, under the RSA assumption on safe moduli, formulated for a random exponent, $\epsilon(\tau)$ is negligible in τ.□ □

[2]This is similar to the reduction in [70]. However, in contrast to Theorem 5 in [70], our reduction is not based on the strong RSA assumption, but on the standard RSA assumption in ROM. This is because e is generated independently of base y and, thus, e is effectively provided as input to the adversary. In fact, the signature scheme in [70] is actually secure under the standard RSA assumption in ROM; this was confirmed via private communication [69].

We stress that exponents in our scheme do not have to be prime, unlike related reductions, e.g., [141, 16, 119, 70]. In fact, the client cannot compute $g^{R_s R_c PCH/H_1(s_l)}$, for $l \in \{1, \ldots, w\}$, unless $R_c PCH/H_1(s_l)$ is an integer. (Recall that R_c is generated honestly). Clearly, if $H_1(s_l) \notin I$, $R_c PCH/H_1(s_l)$ is, – with negligible probability – an integer as long as $H_1(s_l)$ is sufficiently large: random oracles are indeed *division intractable*, as shown in [70, 42] (in particular, [42] presents an algorithm for finding division collisions sub-exponential in τ_1, the digest size).

Security of our construction assumes both semi-honest players and the Random Oracle Model. Nevertheless, generic 2PC techniques, following traditional definitions that also apply to malicious adversaries, do not achieve size-hiding of client input. As noted in [77], the program of each participant (in a protocol for computing the desired functionality) depends on the length of other participant's input. One intuitive argument against the feasibility of input size-hiding protocols secure in the malicious model is that proving well-formed-ness of client input is only possible by considering each client input set element separately (e.g., via some ZK proofs). Thus, combined proofs would have to reveal the upper bound on client input size.

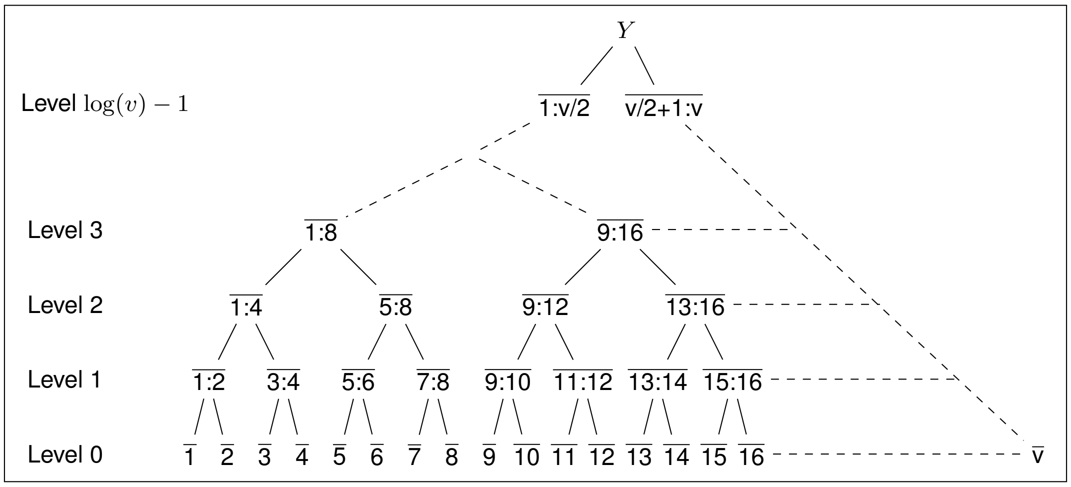
In conclusion, it is an interesting open problem to design PSI protocols (and in general secure two-party computation schemes) that hide the size of at client set and with security in the malicious model. On the other hand, it might be feasible to obtain SHI-PSI constructions that provide security in the presence of malicious *servers*.

7.3.2 Protocol Complexity

We now analyze complexity of protocol presented in Figure 7.1. In each interaction, the server needs to compute $O(w)$ exponentiations, hence, its workload is **independent of client set size.** Client work includes $O(v)$ exponentiations, needed for the computation of $(g^{PCH}) \bmod N$ since PCH is the non-modular product of v values. Additionally, the client computes $K_{c:i} = Z^{R_c \cdot PCH_i} \bmod N$ for each item: every such operation requires $O(v)$ exponentiations, thus, client complexity amounts to $O(v^2)$ exponentiations.[3] Communication complexity in each interaction is dominated by $O(w)$ outputs of $H_2(\cdot)$ sent from the server to the client in the second message. (The first message involves the transmission of a single $\log(N)$-bit value).

We now discuss a simple technique to **reduce client computation**. As discussed above, the naïve computation of $K_{c:i}$ leads to $O(v^2)$ exponentiations. However, this can be reduced to

[3]If the client knew the factorization of N, it could compute PCH and PCH_i-s using multiplication $\bmod\ \phi(N)$, thus, reducing complexity of each exponentiation. However, as discussed earlier, the fact that the client does not know $\phi(N)$ is crucial to server privacy.



Y
Level $\log(v) - 1$
$\overline{\text{1:v/2}}$
$\overline{\text{v/2+1:v}}$
Level 3
$\overline{\text{1:8}}$
$\overline{\text{9:16}}$
Level 2
$\overline{\text{1:4}}$
$\overline{\text{5:8}}$
$\overline{\text{9:12}}$
$\overline{\text{13:16}}$
Level 1
$\overline{\text{1:2}}$ $\overline{\text{3:4}}$ $\overline{\text{5:6}}$ $\overline{\text{7:8}}$ $\overline{\text{9:10}}$ $\overline{\text{11:12}}$ $\overline{\text{13:14}}$ $\overline{\text{15:16}}$
Level 0
$\overline{1}$ $\overline{2}$ $\overline{3}$ $\overline{4}$ $\overline{5}$ $\overline{6}$ $\overline{7}$ $\overline{8}$ $\overline{9}$ $\overline{10}$ $\overline{11}$ $\overline{12}$ $\overline{13}$ $\overline{14}$ $\overline{15}$ $\overline{16}$ $\overline{\text{v}}$

Figure 7.2: Tree-based strategy to reduce client computation.

$O(v \log(v))$ via dynamic programming. Our intuition is as follows: For any (i, j), $K_{c:i}$ and $K_{c:j}$ only differ by one exponent, since $PCH_i = \prod_{l \neq i} H_1(c_l)$, whereas, $PCH_j = \prod_{l \neq j} H_1(c_l)$.

In Figure 7.2, we illustrate this technique using a tree. We define $Y = Z^{R_c} \bmod N$, and $\overline{\text{i:j}} = Y^{\left(\prod_{l \notin [i,j]} H_1(c_l)\right)} \bmod N$. The leaves in the tree contain values $K_{c:i}$, for $1 \leq i \leq v$, e.g.:

$$\overline{\text{i}} = Y^{\left(\prod_{l \neq i} H_1(c_l)\right)} \bmod N = Z^{R_c \cdot PCH_i} \bmod N = K_{c:i}$$

We now obtain total number of exponentiations needed to compute all these values. Note that, from a node with value $\overline{\text{i:j}}$, one can obtain the children, $\overline{\text{i:h}}$ and $\overline{\text{h+1:j}}$, as follows:

$$\begin{aligned} \overline{\text{i:h}} &= \left(\overline{\text{i:j}}\right)^{\left(\prod_{l=h+1}^{j} H_1(c_l)\right)} \bmod N \\ \overline{\text{h+1:j}} &= \left(\overline{\text{i:j}}\right)^{\left(\prod_{l=i}^{h} H_1(c_l)\right)} \bmod N \end{aligned}$$

Since $h = \dfrac{i + (j - i + 1)}{2}$, each of these two operations involves exactly $\dfrac{j - i + 1}{2}$ exponentiations.

At level 0, there are v values, each obtained with a single exponentiation from the parents at level 1. At level 1, there are $v/2$ values, each obtained with 2 exponentiations from nodes at level 2. In general, at level i, there are $v/2^i$ values, each obtained with 2^i exponentiations from nodes at level i+1.

Thus, client overhead can be estimated as:

$$\text{\# exponentiations} = \sum_{i=0}^{\log(v)-1} \left(2^i \frac{v}{2^i}\right) = v \log(v).$$

7.4 Extensions

In this section, we discuss some extensions to the SHI-PSI protocol of Section 7.3.

7.4.1 Linear-Complexity SHI-PSI

In many scenarios, participants engage in multiple interactions, and it is important to hide (from client) any changes in server input. This feature is sometimes referred to as *unlinkability*: the client cannot determine whether any two server interactions are related, i.e., executed on the same input (e.g., see unlinkability definitions in Section 4.6.1 and Section 5.2).

[Common input: $N, g, H_1(\cdot), H_2(\cdot)$ – Server's input (p', q')]

Client, on input: $\mathcal{C} = \{c_1, \ldots, c_v\}$

Server, on input: $\mathcal{S} = \{s_1, \ldots, s_w\}$

Offline

For $i = 1, \ldots, v$:
 $PCH_i \overset{\text{def}}{=} \prod_{l \neq i} H_1(c_l)$
 $a_i = g^{PCH_i} \bmod N$
$PCH \overset{\text{def}}{=} \prod_{i=1}^{v} H_1(c_i)$
$A = g^{PCH} \bmod N$

Online

$R_c \leftarrow_r \{1, \ldots, N^2\}$
$X = A^{R_c} \bmod N$

$$\xrightarrow{X}$$

For $j = 1, \ldots, w$:
 $K_{s:j} = X^{1/H_1(s_j)} \bmod N$
 $t_j = H_2(K_{s:j})$
$\{T_{s:1}, \ldots, T_{s:w}\} = \Pi(\{t_1, \ldots, t_w\})$

For $i = 1, \ldots, v$:
 $K_{c:i} = (a_i)^{R_c} \bmod N$
 $T_{c:i} = H_2(K_{c:i})$

$$\xleftarrow{\{T_{s:1}, \ldots, T_{s:w}\},}$$

Output: $\{c_i \mid c_i \in \mathcal{C} \text{ and } \exists\, T_{s:j} \text{ s.t. } T_{s:j} = T_{c:i}\}$

Figure 7.3: New (linear-complexity) SHI-PSI protocol.

The SHI-PSI protocol presented in Section 7.3 clearly guarantees unlinkability. Server tags are "unlinkable" across multiple interactions, since the server computes a new random R_s, thus, a new $Z \in QR_N$, in each execution. However, it is worth considering scenarios that motivate sacrificing unlinkability for better efficiency.

To this end, we present a modified SHI-PSI protocol that reduces the number of client on-line exponentiations. The main intuition is that removing R_s allows the client to pre-compute the exponentiations involving (long) PCH_i values. We show the resulting protocol in **Figure 7.3**. (Notation as well as common and private inputs are same as protocol in Section 7.3.)

Note that security arguments in Theorem 7.3.1 also apply to this protocol variant. $X = g^{PCH \cdot R_c}$, similar to X^{R_s} in protocol of Figure 7.1, is also uniformly distributed in QR_N, for $R_c \leftarrow_r \{1, \ldots, N^2\}$.

Correctness. Observe that $\forall c_i \in \mathcal{S} \cap \mathcal{C}, \exists\, j$ s.t. $c_i = s_j$. Hence, $H_1(c_i) = H_1(s_j)$ and:

$$K_{s:j} = X^{1/H_1(s_j)} = g^{R_c PCH_j (1/H_1(s_j))} = g^{R_c \cdot PCH_i} = K_{c:i}$$

Consequently, $T_{c:i} = H_2(K_{c:i}) = H_2(K_{s:j}) = T_{s:j}$, and the client learns $c_i \in \mathcal{S} \cap \mathcal{C}$.

Computational and Communication Complexity. The amended SHI-PSI construct in Figure 7.3 incurs the following computational complexity: Server overhead is unaltered from Figure 7.1, i.e., $O(w)$ exponentiations. However, the client performs $O(v \log(v))$ exponentiations *off-line*, and only $O(v)$ exponentiations *on-line*. Communication overhead is the same as in the protocol of Figure 7.1.

7.4.2 SHI-PSI with Data Transfer

It is easy to add the *data transfer* functionality to the protocols in Figure 7.1 and 7.3, and implement a "SHI-PSI with Data Transfer" variant. Following the same intuition discussed in Section 5.5 and Section 6.3, we assume that an additional secure cryptographic hash function $H_3 : \{0,1\}^* \to \{0,1\}^\tau$ is chosen during setup. In all aforementioned protocols, we can use H_3 to derive a symmetric key for a CPA-secure symmetric cipher, such as AES [45], used in the appropriate mode of operation. For every $j = 1, \ldots, w$, the server computes $k_{s:j} = H_3(K_{s:j})$ and encrypts associated data using a distinct key $k_{s:j}$. For its part, the client, for every $i = 1, \ldots, v$, computes $k_{c:i} = H_3(K_{c:i})$ and decrypts ciphertexts corresponding to the matching tag. (Note that $ks_j = kc_i$ if and only iff $s_j = c_i$ and so $T_{s:j} = T_{c:i}$). As long as the underlying encryption scheme is CPA-secure, this extension does not affect security or privacy arguments for any protocol discussed thus far.

Part II
Practical Aspects of Privacy-Preserving Sharing of Sensitive Information

Chapter 8

Building a Toolkit for Privacy-preserving Sharing of Sensitive Information

In this chapter, we present the design and implementation of a toolkit for Privacy-Preserving Sharing of Sensitive Information (PPSSI), geared for database querying applications. We use Private Set Intersection as the main building block and address several interesting challenges.

8.1 Introduction

In previous chapters, we presented several efficient and provably secure protocols for privately sharing sensitive information. The next step is to implement and experiment with them. In this chapter, we describe the design of a Toolkit for Privacy-Preserving Sharing of Sensitive Information (PPSSI). Such a toolkit functions as a *privacy shield* to protect interacting parties from disclosing more than the required minimum of sensitive information. While our main building blocks are PSI techniques, the process of realizing an efficient and secure toolkit for PPSSI is not at all trivial. We model PPSSI in the context of database-querying, involving a *server*, in possession of a private database, and a *client*, performing (privacy-preserving) disjunctive equality queries.

First, we consider a strawman approach, obtained with a naïve adaptation of PSI techniques. In the process, we identify some challenges that arise when realizing PPSSI from PSI, i.e., handling repetitions in input sets (multi-sets) and securely storing database indices. Next, we propose a novel method for database encryption, that addresses aforementioned challenges.

8.2 Preliminaries

8.2.1 Syntax & Notation

As mentioned above, we model PPSSI in the context of simple database querying: A server maintains a database, DB, and multiple clients pose disjunctive SQL queries.

- Server's DB contains w records with m attributes $(attr_1, \cdots, attr_m)$, i.e.,

  DB $= \{R_j\}_{1 \le j \le w}$ for $R_j = \{val_{j,l}\}_{1 \le l \le m}$, (where $val_{j,l}$ is R_j's value for attribute $attr_l$).

- A client poses disjunctive SQL queries, such as:

  $$\text{SELECT * FROM DB WHERE } (attr_1^* = val_1^* \text{ OR } \cdots \text{ OR } attr_v^* = val_v^*) \quad (8.1)$$

PPSSI guarantees that the client, upon query execution, gets all records in DB satisfying *where* clause, and nothing else. The server learns nothing about any $\{attr_i^*, val_i^*\}_{1 \le i \le v}$. We assume that the database schema (format) is known to the client.

An alternative version enables the concept of *authorized queries*, where the client is required to obtain query authorizations from a (mutually trusted) offline ***Certification Authority (CA)*** prior to interacting with the server. That is, the client outputs records matching its query (e.g., in Equation 8.1) if and only if it holds pertinent authorizations, (e.g., for $(attr_i^*, val_i^*)$).

Our notation is reflected in Table 8.1. We use $H_1(\cdot), H_2(\cdot), H_3(\cdot), H_4(\cdot)$ to denote different hash functions.

8.2.2 Privacy Requirements

We now define PPSSI (informal) privacy requirements. We consider both semi-honest adversaries and malicious adversaries.

- *Correctness.* Upon query execution, the client outputs all records in DB that satisfy the *where* clause.

$attr_l$	lth attribute in the database schema	$ctr_{j,l}$	number of times where $val_{j',l} = val_{j,l}, \forall j' <= j$
R_j	jth record in the database	$tag_{j,l}$	tag for $attr_l, val_{j,l}$
$val_{j,l}$	value in R_j corresponding to $attr_l$	$k'_{j,l}$	key used to encrypt k_j
k_j	key used to encrypt R_j	$k''_{j,l}$	key used to encrypt index j
er_j	encryption of R_j	$ek_{j,l}$	encryption of key k_j
$tk_{j,l}$	token evaluated over $attr_l, val_{j,l}$	$eind_{j,l}$	encryption of index j

Table 8.1: Notation used in Chapter 8.

- *Server Privacy.* The client learns no information about any record in DB that does not satisfy the *where* clause.
- *Server Privacy (Authorized Queries).* Same as above. In addition, the client learns no information about any record satisfying the *where* $(attr_i^* = val_i^*)$ clause, unless the $(attr_i^*, val_i^*)$ query is authorized by the CA.
- *Client Privacy.* The server learns nothing about any client query parameters, i.e., all $attr_i^*$ and val_i^*.
- *Client Privacy (Authorized Queries).* Same as above. In addition, the server learns no information about client's authorizations.
- *Client Unlinkability.* The server cannot determine (with probability non-negligibly exceeding $1/2$) whether any two client queries are related.
- *Server Unlinkability.* For any two queries, the client cannot determine (with probability non-negligibly exceeding $1/2$) whether any record in the server's database has changed, except for the records that are learned (by the client) as a result of both queries.
- *Forward Security (Authorized Queries).* The client cannot violate server privacy with regard to prior interactions, using authorizations obtained later.

Unlinkability keeps one participant from noticing changes in other participant's input. In particular, unless server unlinkability is guaranteed, the client can always detect whether the server updates its database between two interactions. Unlinkability also minimizes the risk of privacy leaks. For instance, without client unlinkability, the server learns that the client's queries are the same in two interactions, thus, if one of these queries is leaked, the other query would be immediately exposed.

8.2.3 Pre-Distribution in PSI

PSI constitutes the main building block of our PPSSI toolkit. (We have formally defined PSI functionality in Section 5.2).

We distinguish between (A)PSI-DT protocols based on whether or not they support **pre-distribution**:

(A)PSI-DT *with* pre-distribution: In this variant, the server can "pre-process" its input set, independently from client input to the protocol. This way, the server can pre-distribute its (processed) set items before protocol execution. Both pre-processing and pre-distribution can be done offline, once for all possible clients, thus, server's online complexity does not depend on server input size.

(A)PSI-DT *without* pre-distribution: In this variant, server inputs to the protocol depend on both item in its sets and client's input to the protocol, thus, pre-distribution is impossible.
Pre-distribution precludes server unlinkability, since server input is assumed to be fixed. Also, in the context of authorized protocols with pre-distribution, forward security cannot be guaranteed.

8.2.4 Related Primitives

Searchable Encryption. Song, Wagner, and Perrig [145] introduce Symmetric Searchable Encryption (SSE), allowing a client to store on an untrusted server messages encrypted using a symmetric-key cipher. Later, the client can search for specific keywords by giving the server a trapdoor that does not reveal keywords or plaintexts. Boneh et al. [20] later extended SSE to the public-key setting, i.e., anyone can use client's public key to encrypt and route messages through an untrusted server (e.g., a mail server). The client can then generate search tokens, based on its private key, to let the server identify messages including specific keywords.

Privacy-Preserving Database Querying (PPDQ). Some PPDQ techniques are similar to SSE: a client encrypts its private data, outsources it to an untrusted service provider (while not maintaining copies), and queries the service provider at will. However, in addition to simple equality predicates supported by SSE, certain techniques [83, 92, 17] support general SQL operations. Again, this setting is different from ours: data, although stored by the server, belongs to the client, thus, there is no privacy restriction with respect to the client. Moreover, these techniques do not provide provably secure guarantees, since they are based on statistical (probabilistic) methods.

Another line of work is closely related to Private Information Retrieval (PIR), discussed in Section 3.3.4. Olumofin and Goldberg [131] propose an extension from block-based PIR to SQL-enabled PIR. As opposed to PPSSI, however, server database is public. Moreover, it requires data to be replicated over several non-colluding servers.

Kantarcioĝlu and Clifton [101] consider a scenario where the client matches classification rules against server's database. However, they assume that client's rules are fixed and known to the server. Murugesan et al. [122] also allow "fuzzy" matching. However, this approach requires a number of (expensive) cryptographic operations quadratic in the size of participants' inputs.

Other PPDQ-related results, such as [137, 41], require mutually trusted and non-colluding entities.

8.3 A Strawman Approach

We now attempt to construct PPSSI using a straightforward instantiation of PSI-DT protocols (or APSI-DT, for authorized queries). We outline this *strawman* approach below and show its security limitations.

For each record, the hash of every attribute-value pair $(attr_l, val_{j,l})$ is treated as a set element, and R_j – its associated data. Server "set" is then: $\mathcal{S} = \{(H_1(attr_l, val_{j,l}), R_j)\}_{1 \le l \le m, 1 \le j \le w}$. Client "set" is: $\mathcal{C} = \{H_1(attr_i^*, val_i^*)\}_{1 \le i \le v}$, i.e., elements corresponding to the *where* clause in Equation 8.1. Optionally, if authorized queries are enforced, $\mathcal{C}$ is accompanied by signatures σ_i over $H_1(attr_i^*, val_i^*)$, following the APSI-DT syntax. Participants engage in an (A)PSI-DT interaction; at the end of it, the client obtains all records matching its query. However, the strawman approach has two security issues:

Issue 1: Multi-Sets. While most databases include duplicate values (e.g., "gender=female"), PSI-DT and APSI-DT definitions assume no duplicates.[1] If server set contains duplicate values, corresponding messages (PRF values computed over the duplicate values) to the client would be identical and the client would learn all patterns and distribution frequencies. This raises a serious concern, as actual values can be often inferred from their frequencies. For example, consider a

[1] Some PSI constructs (e.g., [108]) support multi-sets, however, their performance is not optimal as they incur quadratic computational overhead (in the size of the sets), as opposed to recent and efficient (A)PSI-DT protocols with linear complexity (e.g., those in Chapters 5 and 6, or in [100]). Also, they support neither *data transfer* nor *authorization*.

large database where one attribute reflects "employee blood type": since blood type frequencies are well-known for general population, distributions for this attribute would essentially reveal the plaintext.

Issue 2: Data Pointers. To enable querying by any attribute, each record, R_j, must be separately encrypted m times, i.e., once for each attribute. As this would result in high storage/bandwidth overhead, one could encrypt each R_j with a unique symmetric key k_j and then using k_j (instead of R_j) as data associated with $H_1(attr_l, val_{j,l})$. Although this would reduce the overhead, it would trigger another issue: in order to use the key – rather than the actual record – as the associated "data" in the (A)PSI-DT protocol, we would need to store a pointer to the encrypted record alongside each $H_1(attr_l, val_{j,l})$. This would allow the client to identify all $H_1(attr_l, val_{j,l})$ corresponding to a given encrypted record by simply identifying all $H_1(attr_l, val_{j,l})$ with associated data pointers equal to the given records. This information leak would be even more severe if one combines it with the previous "attack" on multi-sets: given two encrypted records, the client could establish their similarity based on the number of equal attributes.

8.4 PPSSI Toolkit

We now present the construction of our PPSSI toolkit. Similar to the strawman approach, it aims at enabling privacy-preserving database querying using any secure (A)PSI-DT instantiation; however, it addresses aforementioned challenges by proposing a novel database-encryption technique. It uses (A)PSI-DT *without* pre-distribution to guarantee server unlinkability and forward security.

High-level operation of PPSSI is illustrated in Figure 8.1. It works with any secure (A)PSI-DT technique: different (A)PSI-DT constructions yield distinct instantiations of the `Token` function (see details below).

8.4.1 The `Token` Function

In step 1, we let the client and the server engage in the *oblivious* computation of `Token` function. As a result, the client obtains $tk_i = \texttt{Token}(c_i)$, where $c_i = H_1(attr_i^*, val_i^*)$. (Note that the server learns nothing about c_i or tk_i, since `Token` function is computed using an (adapted) (A)PSI-DT protocol.

Following a thorough experimental analysis (presented in Appendix A), we select the PSI-

- Client's input: $\{c_i, \sigma_i\}_{1 \le i \le v}$, where: $c_i = H_1(attr_i^*, val_i^*)$. σ_i is only used for APSI-DT protocols.
- Server's input: $\{s_{j,l}\}_{1 \le j \le w, 1 \le l \le m}, \{R_j\}_{1 \le j \le w}$, where: $s_{j,l} = H_1(attr_l, val_{j,l})$

1. Client $\xLeftrightarrow{\text{Obliviously compute:}\{tk_i \leftarrow \texttt{Token}(c_i)\}_{1 \le i \le v}}$ Server
2. Server: $\mathbf{EDB} \leftarrow \texttt{EncryptDatabase}(\texttt{Token}(\cdot), \{R_j\}_{1 \le j \le w})$
3. Server $\xrightarrow{\mathbf{EDB}}$ Client
4. Client: $\forall_{1 \le i \le v} \mathbf{R_i} \leftarrow \texttt{Lookup}(tk_i, \mathbf{EDB})$, Output $\mathbf{R_1} \cup \cdots \cup \mathbf{R_v}$.

Figure 8.1: Outline of our PPSSI construction.

DT technique introduced in Section 5.3.3 – denoted as **DT10-1** – as well as its APSI-DT counterpart (for authorized queries) presented in Section 5.3.2 – denoted as **DT10-APSI**. Both protocols are secure against semi-honest adversaries. As discussed in Chapter 6, with same asymptotic complexity, they can be extended to attain security against malicious adversaries.

Table 8.2 describes the definition of the `Token` function, using DT10-1 and DT10-APSI, over a value x, on client's private input Rc and server's private input Rs.

Instantiation	**Public Params**	**Private Params**		`Token` **definition**
		Server	**Client**	
DT10-1 (Sec. 5.3.3)	$p, q, g, H_1(\cdot)$	Rc	Rs	$\texttt{Token}(x) = (g^{Rc} \cdot H_1(x))^{Rs} \bmod p$
DT10-APSI (Sec. 5.3.2)	$N, e, g, H_1(\cdot)$	Rc	Rs	$\texttt{Token}(x) = (g^{eRc} \cdot H_1(x)^2)^{Rs} \bmod N$

Table 8.2: `Token` definition for DT10-1 and DT10-APSI.

In DT10-1, public parameters include $p, q, g, H_1(\cdot)$, where p is a large prime, g a generator of a subgroup of order q (s.t., $q|p-1$), and $H_1(\cdot)$ is a cryptographic hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Rc and Rs are random values in $\mathbb{Z}_q$. In DT10-APSI, public parameters include $N, e, g, H_1(\cdot)$, where (N, e) is CA's pk, corresponding to sk d; g is a generator of QR_N and $H_1(\cdot)$ is a full-domain hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_N$.

The `Token` function is used twice in our PPSSI construction: in step 1, it is evaluated by the client on input c_i ($1 \le i \le v$) and in step 2, it is evaluated by the server during database encryption (discussed in Section 8.4.2).

In Figure 8.2 and Figure 8.3, we present the details of `Token` instantiation, using, respectively, DT10-1 and DT10-APSI.

Server's evaluation of `Token` over its own inputs (in Algorithm 1, presented below) can

Client's computation of $\texttt{Token}(x)$

1. Client: $Rc \leftarrow_r \mathbb{Z}_q$, $Rx \leftarrow_r \mathbb{Z}_q$
   $X = g^{Rc}$, $M = H_1(x) \cdot g^{Rx}$
2. Client $\xrightarrow{(X,M)}$ Server
3. Server: $R_s \leftarrow_r \mathbb{Z}_q$
   $Z \leftarrow g^{Rs}, M' = M^{Rs}$
4. Server $\xrightarrow{(Z,M)}$ Client
5. Client: $\texttt{Token}(x) = M' \cdot Z^{Rc} \cdot Z^{-Rx}$

Server's computation of $\texttt{Token}(x)$

Server: $\texttt{Token}(x) = (g^{Rc} \cdot H_1(x))^{Rs}$

(All operations are $\bmod\ p$.)

Figure 8.2: Oblivious computation of Token using DT10-1.

Client's computation of $\texttt{Token}(x)$

1. Client: $Rc \leftarrow_r \mathbb{Z}_{N/4}$, $Rx \leftarrow_r \mathbb{Z}_{N/4}$,
   $X = g^{Rc}$, $M = \sigma^2 \cdot g^{Rx}$,
   where $\sigma = H_1(x)^d$, issued by CA.
2. Client $\xrightarrow{(X,M)}$ Server
3. Server: $R_s \leftarrow_r \mathbb{Z}_{N/4}$
   $Z \leftarrow g^{eRs}, M' = M^{eRs}$
4. Server $\xrightarrow{(Z,M)}$ Client
5. Client: $\texttt{Token}(x) = M' \cdot Z^{Rc} \cdot Z^{-Rx}$

Server's computation of $\texttt{Token}(x)$

Server: $\texttt{Token}(x) = (g^{eRc} \cdot H_1(x)^2)^{Rs}$

(All operations are $\bmod\ N$.)

Figure 8.3: Oblivious computation of Token using DT10-APSI.

be performed only after step 1 of Figure 8.1. The server needs to obtain from the client the value $X = g^{Rc}$, that is sent as part of the oblivious computation protocol. Neither the input nor the output of the function is revealed to the client during server's evaluation of Token in Algorithm 1.

8.4.2 Database Encryption with counters

In step 2 of Figure 8.1, the server runs EncryptDatabase procedure (described in Algorithm 1) and creates $\mathbf{EDB}$, that is transferred to the client in step 3.

In contrast to the strawman approach, we modify the "encryption" technique: rather than (directly) using a symmetric-key encryption scheme, the EncryptDatabase procedure is invoked. It is illustrated inAlgorithm 1. It takes as input the definition of the Token function, and server's record set. It consists of two "phases": (1) *Record-level* and (2) *Lookup-Table* encryptions.

Record-level encryption is relatively trivial (lines 1–6): first, the server shuffles record locations; then, it pads each R_j up to a fixed maximum record size, picks a random symmetric key k_j, and encrypts R_j as $er_j = \mathsf{Enc}_{k_j}(R_j)$.

Lookup-Table (LTable) encryption (lines 8–15) pertains to attribute name and value pairs. It enables efficient lookup and record decryption. In step 8, the server hashes an attribute-value pair

Algorithm 1: `EncryptDatabase` Procedure.

```
input  : Function $\texttt{Token}(\cdot)$ and record set $\{R_j\}_{1\leq j\leq w}$
output: Encrypted Database $\mathbf{EDB}$
1:  Shuffle $\{R_j\}_{1\leq j\leq w}$
2:  $maxlen \leftarrow$ max length among all $R_j$
3:  for $1 \leq j \leq w$ do
4:      Pad $R_j$ to $maxlen$;
5:      $k_j \leftarrow_r \{0,1\}^{128}$;
6:      $er_j \leftarrow \mathsf{Enc}_{k_j}(R_j)$;
7:      for $1 \leq l \leq m$ do
8:          $hs_{j,l} \leftarrow H_1(attr_l, val_{j,l})$;
9:          $tk_{j,l} \leftarrow \texttt{Token}(hs_{j,l})$;
10:         $tag_{j,l} \leftarrow H_2(tk_{j,l}||ctr_{j,l})$;
11:         $k'_{j,l} \leftarrow H_3(tk_{j,l}||ctr_{j,l})$;
12:         $k''_{j,l} \leftarrow H_4(tk_{j,l}||ctr_{j,l})$;
13:         $ek_{j,l} \leftarrow \mathsf{Enc}_{k'_{j,l}}(k_j)$;
14:         $eind_{j,l} \leftarrow \mathsf{Enc}_{k''_{j,l}}(j)$;
15:         $\mathbf{LTable}_{j,l} \leftarrow (tag_{j,l}, ek_{j,l}, eind_{j,l})$;
16:     end for
17: end for
18: Shuffle $\mathbf{LTable}$ with respect to $j$ and $l$;
19: $\mathbf{EDB} \leftarrow \{\mathbf{LTable}, \{er_j\}_{1\leq j\leq w}\}$;
```

and uses the result as input to `Token` function in step 9. In step 10, we use the concatenation of `Token` output and a counter, $ctr_{j,l}$, in order to compute the tag $tag_{j,l}$, later used as a lookup tag during client query. We use $ctr_{j,l}$ to denote the index of duplicate value for the l-th attribute. In other words, $ctr_{j,l}$ is the counter of occurrences of $val_{j',l} = val_{j,l}, \forall j' <= j$. For example, the third occurrence of value "Smith" for attribute "Last Name" will have the counter equal to 3. The counter guarantees that duplicate $(attr, val)$ pairs correspond to different tags, thus addressing *Issue 1*. Next, the server computes $k'_{j,l} = H_3(tk_{j,l}||ctr_{j,l})$ and $k''_{j,l} = H_4(tk_{j,l}||ctr_{j,l})$. Note that $k'_{j,l}$ is used for encrypting symmetric key k_j. Whereas, $k''_{j,l}$ is used for encrypting the index of R_j. In step 13, the server encrypts k_j as $ek_{j,l} = \mathsf{Enc}_{k'_{j,l}}(k_j)$. Then, the server encrypts $eind_{j,l} = \mathsf{Enc}_{k''_{j,l}}(j)$.

The encryption of index (data pointer) guarantees that the client cannot link two tags belonging to the same record, thus addressing *Issue 2*. In step 15, the server inserts each $tag_{j,l}$, $ek_{j,l}$ and $eind_{j,l}$ into LTable, which is $\{tag_{j,l}, ek_{j,l}, eind_{j,l}\}_{1\leq j\leq w, 1\leq l\leq m}$. Next, the server shuffles LTable (step 18). The resulting encrypted database, $\mathbf{EDB}$, is composed of LTable and $\{er_j\}_{1\leq j\leq w}$ (step 19).

8.4.3 Lookup with counters

Algorithm 2: `Lookup` Procedure.

```
input  : Search token $tk$ and encrypted database $\mathbf{EDB} = \{\mathbf{LTable}, \{er_j\}_{1\leq j\leq w}\}$
output: Matching record set $\mathbf{R}$
 1: $ctr \leftarrow 1$;
 2: while $\exists tag_{j,l} \in \mathbf{LTable}\ s.t.\ tag_{j,l} = H_2(tk||ctr)$ do
 3:     $k'' \leftarrow H_4(tk||ctr)$;
 4:     $j' \leftarrow \mathsf{Dec}_{k''}(eind_{j,l})$;
 5:     $k' \leftarrow H_3(tk||ctr)$;
 6:     $k \leftarrow \mathsf{Dec}_{k'}(ek_{j,l})$;
 7:     $R_j \leftarrow \mathsf{Dec}_k(er_{j'})$;
 8:     $\mathbf{R} \leftarrow \mathbf{R} \cup R_j$;
 9:     $ctr \leftarrow ctr + 1$;
10: end while
```

In the last step in Figure 8.1 (step 4), the client runs `Lookup` procedure (illustrated in Algorithm 2), using the tk_i tokens over $\mathbf{EDB}$.

We now discuss `Lookup` procedure in Algorithm 2. It is used by the client to obtain the query result, i.e., to search $\mathbf{EDB}$ for all records that match client's search tokens.

In step 1, the client initializes a counter to 1. Next, it searches $\mathbf{LTable}$ for tag $tag_{j,l} = H_2(tk||counter)$. If there is a match, the client attempts to recover the record associated with $tag_{j,l}$. It computes $k'' = H_4(tk||ctr)$ and tries to decrypt: $j' = \mathsf{Dec}_{k''}(eind_{j,l})$. Note that $er_{j'}$ now corresponds to the associated record. To decrypt $er_{j'}$, the client first recovers the key k used to encrypt $er_{j'}$, by computing $k' = H_3(tk||ctr)$ and obtaining $k = \mathsf{Dec}_{k'}(ek_{j,l})$. Finally, the client obtains $R_j = \mathsf{Dec}_k(er_{j'})$.

There are several ways for the client to store $\mathbf{LTable}$. Hash table storage is most efficient as it only requires constant lookup time.

8.4.4 Discussion

Query Example. We now demonstrate the correctness of our query procedure. We consider the case where the the client searches for records matching "gender = male". We assume that server's database includes the attribute "gender" with two occurrences of value "male". In Algorithm 1, the server generates the same tk (step 9) for the two occurrences of ("gender", "male"). However, for the first occurrence, $tag = H_2(tk||1), k' = H_3(tk||1), k'' = H_4(tk||1)$ while, for the second occurrence, $tag = H_2(tk||2), k' = H_3(tk||2), k'' = H_4(tk||2)$.
If the client searches for records matching "gender = male", it first derives tk (step 1 of Figure 8.1). Next, it matches $H_2(tk||1)$ in $\mathbf{LTable}$, derives keys $k' = H_3(tk||1), k'' = H_4(tk||1)$, and recovers the index in step 4 and the record in step 7 of Algorithm 2. It also looks for $H_2(tk||2)$ and performs the same operations, except that $k' = H_3(tk||2), k'' = H_4(tk||2)$. Finally, the client looks for $H_2(tk||3)$: finding no match, it terminates.

Addressing Issue 1 and 2. Our construction discussed above addresses *Issue 1* and *Issue 2*, raised in Section 8.3.

- *Multi-sets:* The use of counters during database encryption makes each $tag_{j,l}$ (resp., $ek_{j,l}$, $eind_{j,l}$) distinct in $\mathbf{LTable}$, thus hiding plaintext patterns.
- *Data Pointers:* Storing $eind_{j,l}$ (rather than j) in $\mathbf{LTable}$, prevents the server from exposing the relationship between an entry $\mathbf{LTable}_{j,l}$ and its associated record R_j.

Security. PPSSI security immediately stems from that of the underlying (A)PSI-DT protocols, used during `Token` computation. However, additional details as well as formal adversarial games and proofs (for both semi-honest and malicious security) can be found in [50].

Removing the need for online server. Although it only needs to perform oblivious computation of tokens, we require the server to be online. Inspired by [84] and [62], we can replace the online server with a tamper-proof hardware component (e.g., a smartcard), dedicated to computing `Token` function. The server only needs to program its secret key into the smartcard. This way, after handing the smartcard to the client, the server can go offline. The smartcard is assumed to enforce a limit on the number of `Token` invocations.

8.4.5 Performance Evaluation

We now evaluate the performance of the PPSSI toolkit. Throughout all experiments, we use 2048-bit moduli and records of fixed $2KB$ length. Testing software was written in C++ using OpenSSL (ver. 1.0), GMP (ver. 5.01) libraries. Experiments were performed on a Ubuntu 9.10 desktop platform with Intel Xeon E5420 CPU (2.5GHz and 6MB cache) and 8GB RAM.

Figure 8.4 shows the computational overhead for the oblivious computation of `Token` function, for every possible (A)PSI-DT instantiation. The cost always increases linearly with client's query size. Figure 8.5 evaluates performance of the Lookup-Table encryption, performed by the server. This operation includes server's computation of `Token` function over its own input. Again, runtime increases linearly with the product of the number of records (w) and the number of attributes (m). Figure 8.6 shows the cost of the Record-level encryption. This only depends on the number of records. Compared to Lookup-table encryption, the Record-level encryption incurs a negligible overhead.

The cost of `Lookup` procedures (Algorithm 2) is negligible (in the order of $10\mu s$) and we omit its measurement details.

We conclude that, as all operations have linear complexity, our constructions scale efficiently for larger databases and query sets.

<u>PPSSI vs PIR</u>

We compare the efficiency of PPSSI to that of Symmetric PIR (SPIR, reviewed in Section 3.3.4). Recall that PPSSI provides privacy guarantees resembling those of SPIR. Both hide client's access patterns to the server and also protect privacy of server's data (with respect to records not matching the queries). However, one possible drawback of PPSSI is that communication overhead is *linear* in the size of the database, whereas, SPIR incurs *sub-linear* communication overhead. However, (1) SPIR does not immediately support keyword search, and (2) SPIR introduces a significantly higher computation overhead, that negates its advantage in communication complexity. To support latter claim, we compare overall performance of PPSSI with that of Gentry and Ramzan's single-database PIR (GR-PIR) [73]. To the best of our knowledge, it is currently the most efficient single-database SPIR technique. Assuming a database with n records, it incurs $O(k + \bar{d})$ communication (where $k \leq \log n$ and $\bar{d}$ is the bit-length of each record), and $O(n)$ computation overhead.

For the comparison, we used a database with $w = 1024$ records and $m = 5$ attributes. Each record is of size $2KB$. We assume the client's query size is $v = 1024$ and and 10 (1%) records

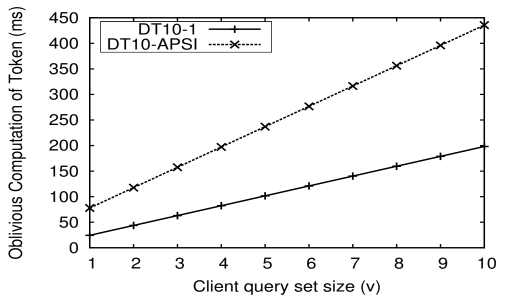


Figure 8.4: `Token` Oblivious Computation.

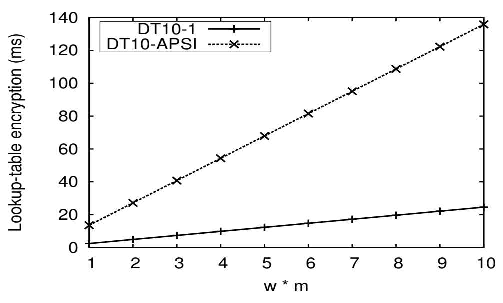


Figure 8.5: Lookup-Table Encryption (line 8-15 of Algorithm 1).

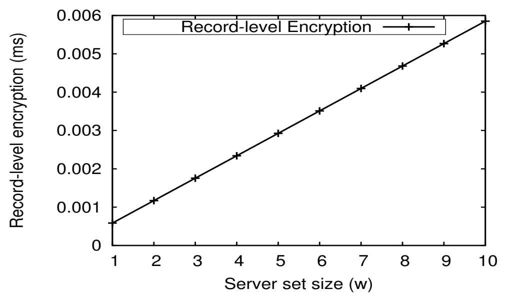


Figure 8.6: Record-level Encryption (line 1-6 of Algorithm 1).

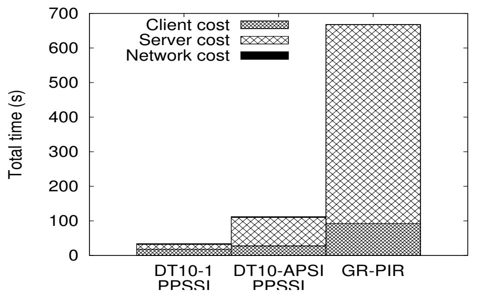


Figure 8.7: Comparison between our PPSSI construction and GR-PIR [73].

matching the query (v_m). From a conservative stance, we used a relatively (10Mbps) connection between client and server. All moduli were 2048 bits.

Results shown in Figure 8.7 and confirm that PPSSI is significantly more efficient than GR-PIR. We break down the overall cost into client, server and network transmission components. For all schemes, transmission cost (at the top stack in each bar) is negligible compared to client and server cost.

On the other hand, linear communication overhead still prompts some concerns regarding the scalability of our techniques to very large databases. Nonetheless, our preliminary results in [50] suggest that one may minimize bandwidth overhead by using a (semi-trusted) third party or a piece of trusted hardware.

Chapter 9

Private Interest Sharing and Activity Scheduling for Smartphones

In this chapter, we explore privacy-preserving techniques geared for mobile applications where sensitive information is shared between smartphone users.

9.1 Introduction

In this last chapter, we turn to the *mobile environment*. Equipped with relatively powerful processors and fairly large memory and storage capabilities, smartphones can nowadays accommodate increasingly complex interactive applications. As a result, the growing amount of sensitive information shared by smartphone users raises serious privacy concerns and motivates the need for appropriate privacy-preserving mechanisms.

We focus on the design of collaborative applications involving participants – with limited reciprocal trust – willing to share *sensitive information from their smartphones* and use it to (cooperatively) perform operations without endangering their privacy. Within this context, prior work has failed in combining provably secure guarantees with (realistically) efficient techniques (as discussed in the remainder of this chapter).

9.1.1 Motivation

The increasing dependence on anywhere-anytime availability of information has made smartphones commensurably more influential in people's computing ecosystems. Many tasks once performed exclusively on PCs have now branched out to phones and have successfully accommodated device constraints, such as display, connectivity, and power [102]. Smartphone applications are progressively enabling sophisticated interactions among people. Nowadays, users share information, for instance, for real-time social networking [130], to discover nearby friends (e.g., using Google Latitude, Foursquare, Gowalla, etc.), to exchange recommendations [91], or even for work-related purposes [103].

Beneath the proliferation of information exchange lays an extremely rich source of data: mobile phones, typically accompanying users 24/7, learn significant amounts of information about their owners and their environment. Business models behind many of today's free web applications, such as social network websites or search engines, heavily rely on data *in the clear*, collected from the users and later used to offer context-based services and recommendations. However, as the amount (and sensitiveness) of shared information increases, so do related privacy concerns.

In this chapter, we present a novel architecture geared for privacy-sensitive applications where personal information is shared among a group of users and decisions are made based on given optimization criteria. Specifically, we focus on two application scenarios: (i) privacy-preserving interest sharing, i.e., discovering shared interests without leaking users' private information, and (ii) private scheduling, i.e., determining common availabilities and location preferences that minimize associate costs, without exposing any sensitive information. In doing so, the main challenge is to protect privacy using cryptographic (provably secure) techniques while respecting computational and bandwidth constraints typical of smartphones.

Previous work has attempted to address this problem by using distributed cryptographic primitives, such as PSI, discussed in Chapters 5 and 6. While some of these techniques are quite efficient for the two-party case, the multi-party variants are not practical enough for large-scale deployment. For instance, the most efficient multi-party PSI, [108], incurs computational complexity – specifically, a number of long modular exponentiations – *quadratic* in the size of participants' inputs. Alternatively, statistical methods for protecting privacy have also been used [93]. However, they do not achieve provable privacy guarantees and generally require a fully-trusted party to produce privacy-preserving statistical digests. Such a trust assumption is often unrealistic, since all sensitive information is concentrated at this party. Beyond its obvious privacy implications, this modus operandi imposes liability issues with regard to participants' private data and concerns about possible data loss, subpoena, etc.

9.1.2 Roadmap

Motivated by the arguments above, we introduce an appropriate architecture that involves several smartphone users and a semi-trusted server (Section 9.2). This entity is never trusted with participants' data itself – i.e., it learns no information about users' inputs or outputs. We envision that this entity may be implemented by public services, (e.g., Nokia Ovi [128] or Amazon EC2 [2]), without disclosing any data in the clear.

We address the problem of letting smartphone users share their *interests*, while maintaining their privacy (Section 9.3). We overview several application examples, such as discovering matching locations, routes, availabilities, etc., without exposing any other information beyond the matching interests, or calling "polls" to find most popular preferences, without revealing anything about other preferences or single user's choices. We present a generic efficient cryptographic technique that addresses all these scenarios.

Next, we consider assigning *costs* to the various interests, and we focus on minimizing the overall aggregated cost in a privacy-preserving way. We consider, as an example, the problem of Private Scheduling, where a group of smartphone users want to schedule a meeting while preserving the privacy of their calendars (Section 9.4). We are stimulated by ongoing projects aiming at automatically filling mobile phones' calendars with users' activities (e.g., based on emails, text messages, routes). Also, widespread calendar applications, such as Apple iCal, Microsoft Outlook, and Google Calendar, currently provide software counterparts for smartphones. We propose three protocols that target different scenarios, corresponding to different efficiency, privacy, and system requirements.

Finally, we analyze the performance of proposed constructs (Section 9.8) and discuss related work (Section 9.9).

9.2 Preliminaries

In this section, we model our system for privacy-preserving computation on mobile devices. We describe players, privacy properties, and trust assumptions.

9.2.1 Players

Our system consists of:

- N smartphone users, $P_1, \ldots, P_N$, called *participants*. Participants $P_1, \ldots, P_N$ hold private information, denoted, resp., with $x_1, \ldots, x_N$. Throughout the rest of this chapter, we assume that $N > 2$.
- A public *server*, S, that generates no input.

Participant P_1 is called *Initiator*, to denote that she is starting the interaction. Depending on the application, P_1 might be performing additional operations compared to other participants.

9.2.2 Server-Assisted Privacy-preserving Computation

We assume that participants, $P_1, \ldots, P_N$, wish to jointly compute a function $f(x_1, \ldots, x_N)$, without revealing any information about their private input, $(x_1, \ldots, x_N)$, beyond what can be deduced from the output of the function. Note that one or all participants may receive the output of the computation, depending on the application. Server S assists the participants in the function computation. However, it learns no information about either $(x_1, \ldots, x_N)$ or $f(x_1, \ldots, x_N)$. In other words, S receives and processes only encryptions of participants' inputs. Specifically, we define the following privacy requirements:

- ***Participant's Privacy w.r.t. Server:*** Privacy of each participant P_i, running on input x_i, is guaranteed w.r.t. server S if no information about x_i is leaked to S. Privacy is considered as the probabilistic advantage that S gains from obtaining encrypted inputs toward learning participants' actual inputs. Formally, we say that the algorithm A is *Private w.r.t. Server* if no polynomially bounded adversary $\mathcal{A}$ can win the following game with probability non-negligibly over 1/2. The game is between $\mathcal{A}$ (playing the role of the server) and a challenger Ch:

  1. Ch executes setup operations and computes public parameters (if any).
  2. $\mathcal{A}$, on input the public parameters, selects two inputs (w_0, w_1).
  3. Ch picks a random bit $b \leftarrow_r \{0, 1\}$ and interacts with $\mathcal{A}$ by following the computation on behalf of a participant, on input the public parameters and private input w_b.
  4. $\mathcal{A}$ outputs b' and wins if $b' = b$.

We anticipate that, since S only receives encrypted inputs, privacy will be reduced to the security of the encryption algorithm.

- ***Participant's Privacy w.r.t. Other Participants:*** Privacy of each participant P_i, running on input x_i, is guaranteed w.r.t. other participants $P_j(\,\forall j \neq i)$, if no information about x_i is leaked to P_j, beyond what can be inferred from $f(x_1, \ldots, x_N)$. Privacy is considered as the probabilistic advantage that P_j gains from the protocol execution in learning P_i's input. Specifically, we define the advantage of P_j of identifying the input of any P_i, i.e., w_i (for $P_i \neq P_j$), in the algorithm A as

  $$Adv_{P_j}(A; P_i) = \left| Pr_{P_j}[w_i' = w_i] - \frac{1}{2} \right|$$

  where w_i' is P_j's guess of P_i's input w_i.

  We say that the algorithm A is *Private w.r.t. Other Participants* if no polynomially bounded adversary $\mathcal{A}$, playing the role of P_j in A, has a non-negligible advantage $Adv_{P_j}(A; P_i)$ for any $P_i \neq P_j$.

9.2.3 Trust Assumptions

We assume participants to be *semi-honest*, i.e., they faithfully follow protocol specifications and do not misrepresent any information on their inputs, as per Goldreich's definitions [77]. However, during or after protocol execution, they try to infer additional information about other participants' inputs. Also, we assume that participants have a minimum degree of knowledge among each other, e.g., they belong to the same organization or community. Thus, they do not have any incentive to deviate from the protocol.

The server is assumed to be semi-trusted, i.e., it follows protocol specifications and does not collude with any participants. It is not trusted with any private data.

9.3 Sharing Interests with Privacy

We consider the following application examples:

1. A smartphone application provides users with the possibility of calling *privacy-preserving polls*: a user can request to a community of users to indicate their preferences on a topic of interest, in order to jointly determine the most popular choice. Nothing should be revealed about a single user's preferences or other choices beyond the most popular one.

2. Members of a virtual community (e.g., AC Milan fans) want to share their location only to discover if there is at least a given number of members around the same location (e.g., 10 fans in the same pub to watch a given match).

3. A group of UC Irvine commuters want to find out common routes to join a carpooling program, i.e., they agree on sharing rides from their homes (or from a meeting location) to a common destination, e.g., as a work center. Nonetheless, information about their routes should be revealed only for matching commutes.

4. A group of Nokia employees are willing to share their availabilities, e.g., to discover times and locations at which at least 75% of them are available (if any), but do not want to reveal whether or not they are free in any other timeslots.

The four examples above exhibit some similar features: users are not willing to reveal their information wholesale, rather, they want to find only matching interests.

Motivated by these examples, we introduce the problem of ***Privacy-preserving Interest Sharing*** (PIS). We intentionally describe PIS as a broad notion and we present a generic efficient and provably private cryptographic construct that targets a diversified set of scenarios.

PIS employs a semi-trusted server – introduced in Section 9.2 – to assist the computation. However, the server is never trusted with any user data (as opposed to a non privacy-preserving approach requiring all users to submit their interests and let the server find the matches). An additional trivial approach would require users to share a symmetric key, use a (deterministic) symmetric-key encryption scheme, and let the server match ciphertexts. However, establishing and managing secret keys is not viable in several settings, like in scenarios 1-3 above. For instance, in (1) the user calling the poll does not even know who is participating. Moreover, if the key is compromised, all interests of all participants would be revealed.

Informal Definition of PIS. We assume N participants, $P_1, \ldots, P_N$, where P_1 acts as the *Initiator* of the protocol. Each participant P_i (for $i \in [1, N]$) holds a list of interests of size m_i. Without loss of generality, we assume that $\forall i, m_i = m$ (i.e., all lists are equal size) to ease presentation. The *interests* of each participant P_i are denoted as $\{\omega_{i:1}, \ldots, \omega_{i:m}\}$ – i.e., participants' inputs in the PIS protocol. The goal of PIS is to find the interests that are shared among at least a ***threshold of ϑ participants***. Participants' privacy needs to be guaranteed by ensuring that, after protocol execution, the participants do not learn anything beyond matching interests. Also, server S learns no information about any participant's input. (Recall privacy definitions from Section 9.2.2).

9.3.1 PIS Protocol Specification

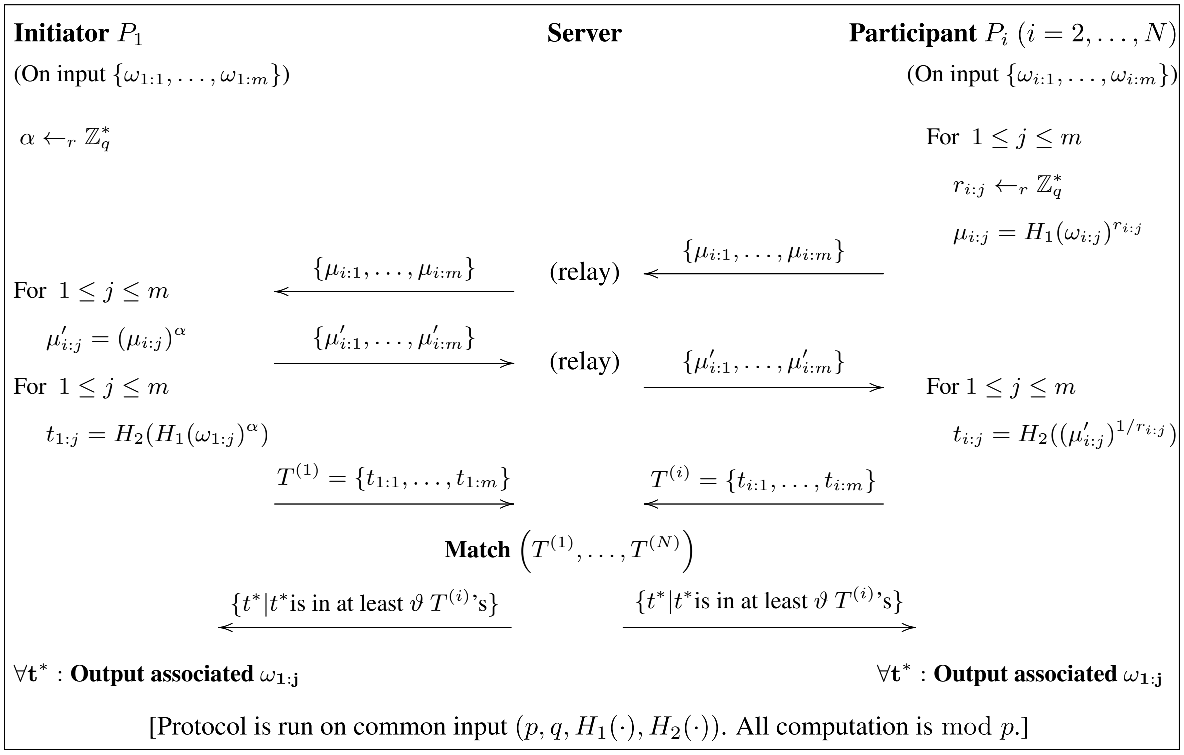

Initiator P_1
(On input $\{\omega_{1:1}, \ldots, \omega_{1:m}\}$)
Server
Participant P_i $(i = 2, \ldots, N)$
(On input $\{\omega_{i:1}, \ldots, \omega_{i:m}\}$)
$\alpha \leftarrow_r \mathbb{Z}_q^*$
For $1 \leq j \leq m$
$r_{i:j} \leftarrow_r \mathbb{Z}_q^*$
$\mu_{i:j} = H_1(\omega_{i:j})^{r_{i:j}}$
$\{\mu_{i:1}, \ldots, \mu_{i:m}\}$
(relay)
$\{\mu_{i:1}, \ldots, \mu_{i:m}\}$
For $1 \leq j \leq m$
$\mu'_{i:j} = (\mu_{i:j})^\alpha$
$\{\mu'_{i:1}, \ldots, \mu'_{i:m}\}$
(relay)
$\{\mu'_{i:1}, \ldots, \mu'_{i:m}\}$
For $1 \leq j \leq m$
$t_{1:j} = H_2(H_1(\omega_{1:j})^\alpha)$
For $1 \leq j \leq m$
$t_{i:j} = H_2((\mu'_{i:j})^{1/r_{i:j}})$
$T^{(1)} = \{t_{1:1}, \ldots, t_{1:m}\}$
$T^{(i)} = \{t_{i:1}, \ldots, t_{i:m}\}$
Match $\left(T^{(1)}, \ldots, T^{(N)}\right)$
$\{t^* | t^*$is in at least ϑ $T^{(i)}$'s$\}$
$\{t^* | t^*$is in at least ϑ $T^{(i)}$'s$\}$
$\forall \mathbf{t}^*$: **Output associated** $\omega_{\mathbf{1:j}}$
$\forall \mathbf{t}^*$: **Output associated** $\omega_{\mathbf{1:j}}$
[Protocol is run on common input $(p, q, H_1(\cdot), H_2(\cdot))$. All computation is $\bmod\ p$.]

Figure 9.1: Our Private Interest Sharing (PIS) Protocol.

Our PIS construction is illustrated in Figure 9.1 and works as follows:

Setup

During setup, the server, S, publishes public parameters (p, q, g, H_1, H_2), where: p, q are prime numbers s.t. $q|p-1$, g is a generator of the subgroup of size q, $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$, and $H_2 : \{0,1\}^* \to \{0,1\}^\tau$ (given a security parameter τ) are cryptographic (i.e., collision-resistant) hash functions. The Initiator, P_1, privately picks a random $\alpha \leftarrow_r \mathbb{Z}_q^*$. (Note that all computations below are performed $\bmod\ p$.)

Interaction

In the interactive phase of the protocol, each participant P_i (for $i \in [2, N]$), on input $\{\omega_{i:1}, \ldots, \omega_{i:m}\}$, for each $j \in [1, m]$: (1) picks $r_{i:j} \leftarrow_r \mathbb{Z}_q^*$, (2) computes and sends S: $\mu_{i:j} = H_1(\omega_{i:j})^{r_{i:j}}$.

Next, S forwards all received $\mu_{i:j}$'s to P_1, who, in turn, responds to participants (via S) with $\mu'_{i:j} = (\mu_{i:j})^{\alpha}$.

Finally, each participant P_i (for $i \in [2, N]$), upon receiving $\mu'_{i:j}$ ($j \in [1, m]$), computes and sends S:

$$T^{(i)} = \left\{ t_{i:j} \mid t_{i:j} = H_2 \left[(\mu'_{i:j})^{1/r_{i:j}} \right] \right\}_{j \in [1,m]}$$

Observe that $t_{i:j} = H_2[H_1(\omega_{i:j})^{\alpha}]$. Also, note that, as opposed to $\mu_{i:j}$'s, S does *not* forward $t_{i:j}$'s to any participant.

Matching

During the matching phase, the Initiator P_1, for $j \in [1, m]$ sends S:

$$T^{(1)} = \{t_{1:j} \mid t_{1:j} = H_2 \left[H_1(\omega_{1:j})^{\alpha}\right]\}_{j \in [1,m]}$$

Next, S identifies all the items t^* that appear in *at least* ϑ different $T^{(i)}$ sets, and outputs them to the original participants that contributed them.

Finally, these participants learn (threshold) interest matching by associating t^* to values $\omega_{i:j}$ producing it.

Remark. PIS can (straightforwardly) be applied to the application examples discussed above. For instance, a user may call for a poll on the best bars in the city, e.g., using a social networking application on her smartphones. Every participant in the poll would engage in a PIS computation as described above. At the end of the poll, the server, e.g., the social network provider, outputs the value t^* that appears most times to the participants, who can then reconstruct the most "popular" bar.

Complexity of PIS. The computational complexity of the protocol amounts to $(N \cdot m)$ exponentiations for the Initiator, whereas, all other participants perform $(2m)$ modular exponentiations. We pick p to be 1024-bit long, and q of size 160-bit (with no loss of security). Thus, using short exponents (160-bit), modular exponentiations in the protocol are very efficient. Communication overhead for the server and the Initiator amounts to $(N \cdot m)$ group elements (i.e., 1024-bit) and hash values (i.e., 160-bit using SHA-1 [59]), whereas, for the other participants, the overhead amounts to m group elements and hash values.

9.3.2 Privacy of PIS

Our PIS construction provides provable-privacy guarantees. Specifically, *Privacy w.r.t. Server* is guaranteed as the server S only receives outputs of the one-way functions $H_1(\cdot), H_2(\cdot)$, whose inputs cannot be "forged" unless S knows either α (secret to P_1) or some $r_{i:j}$ (secret to P_i's). Thus, if an adversary $\mathcal{A}$ violates Privacy w.r.t. Server, then we can construct another adversary that violates the collision resistance of the hash functions $H_1(\cdot), H_2(\cdot)$.

Next, *Privacy w.r.t. Other Participants* immediately stems from security arguments of the Private Set Intersection technique in [100], proven secure under the One-More-DH assumption [14], on which our PIS protocol is based. In other words, if any participant P_j has a non-negligible advantage $Adv_{P_j}(A)$ (defined in Section 9.2.2), then we can construct an attack to the Private Set Intersection protocol in [100].

Recall, however, that [100] only provides a two-party protocol, while our variant extends to multiple parties. We minimize overall overhead using the semi-trusted public server: in fact, available multi-party PSI techniques [108] require several rounds of computation and computational complexity at least quadratic in the size of participants' inputs.

9.4 Private Scheduling

In this section, we explore the concept of *Private Scheduling* for smartphones. Recall example (4) from Section 9.3: a group of employees want to schedule a meeting and select a timeslot such that at least a given number of users are free. We now go a step further: instead of assigning a binary value to time periods or to proposed locations (i.e., available/busy, suitable/unsuitable), we consider *non-binary* costs. For instance, the smartphone can calculate the carbon footprint or the gas cost required to reach a given destination, or how much the user is *tied* to a busy timeslot. Such a flexibility is particularly appealing in the mobile environment, where users carry their device anytime and anywhere. Thus, smartphones can infer their preferences, habits, routes, and assist them in determining availabilities and preferred locations. Therefore, we assume that a cost, between 0 and c_{max}, is assigned to each timeslot and/or location. (In the rest of this chapter, we refer to "timeslots" only, while referring to timeslots and/or locations.)

Users' calendars potentially contain a high volume of sensitive information. Exposed availabilities could be misused to infer affiliation, religion, culture, or correlated to other users. Hence, our goal is to allow users to find the most suitable timeslot – i.e., the one with the minimum sum of costs – while learning *nothing* about single users' availabilities. Our techniques employ the semi-trusted server introduced in Section 9.2 to aggregate users' encrypted inputs. One user, denoted as the *Initiator*, initiates the protocol. She accepts a slightly increased computational overhead – a reasonable assumption, considering she is the one willing to schedule the meeting. In return, only the Initiator obtains the outcome of the protocol.

The Private Scheduling Problem. Private Scheduling involves N different participants, $P_1, \ldots, P_N$. P_1 is the *Initiator* of the protocol. Each P_i (for $i \in [1, N]$) maintains a private calendar, divided into m timeslots. Typical timeslot granularities are 30 or 60 minutes (however, one can tune it according to users' preferences). Each P_i assigns a cost $c_{i:j}$ ($0 \leq c_{i:j} \leq c_{max}$), for each timeslot $j \in [1, m]$ (e.g., ranging from 0 to 10).

Definition 9.1. *(Aggregated Cost.) For a given timeslot j, the aggregated cost $ac_j = \sum_{i=1}^{N} c_{i:j}$ denotes the sum of all participants' cost.*

Definition 9.2. *(Threshold.) A threshold value, ϑ, depending on c_{max} and N, denotes the maximum acceptable aggregated cost to consider a timeslot to be* suitable. *We consider $\vartheta = f(c_{max}, N)$. A typical value could be $\vartheta = \frac{c_{max}}{2} \cdot N$.*

The goal of Private Scheduling is to output to the Initiator all timeslots with aggregated costs smaller than ϑ (if any).

9.5 PrivSched-v1

We now present our first technique for Private Scheduling, *PrivSched-v1*. Before, we provide some technical background.

Preamble. PrivSched-v1 relies on the Paillier Cryptosystem [133], a public-key probabilistic encryption scheme that provides additive homomorphism – i.e., the product of two ciphertexts decrypts to the sum of the corresponding plaintexts. We refer to Chapter 2.2 for a detailed description. Following the intuition of [55], additively homomorphic cryptosystems, such as Paillier, can be used to compute *homomorphic minimization* (or maximization), i.e., one can find the minimum of some integers while operating on ciphertexts only, thus, without learning any information on those integers. We extend this technique to obtain the *homomorphic argmin*, i.e., to additionally find which integer corresponds to the minimum. We use a *tagging* system based on powers of 2. This extension is, to the best of our knowledge, the first attempt in this direction, thus, it can be of independent interest.[1]

We encode integers in a unary system: to represent an integer X, we repeat X times encryptions of 0's. We denote this encoding technique as *vector-based representation (vbr)*:

$$x \underset{\textbf{vbr}}{\longrightarrow} \overrightarrow{X} = [\underbrace{E(0), \ldots, E(0)}_{x\ times}, E(1), E(z), \ldots, E(z)]$$

$E(\cdot)$ denotes encryption using Paillier, and z a random number in the Paillier setting. (This is used for padding).

Then, we raise each element of the vbr to the value of a tag – a power of 2. (The tagging is performed on ciphertexts, rather than plaintexts, as it will become clear later in the chapter). After tagging, $E(0)$ remains $E(0)$, while $E(1)$ becomes $E(tag)$: this allows to identify which value corresponds to the minimum, after the homomorphic minimization is performed. ($E(\cdot)$ denotes encryption using Paillier, and z a random number in the Paillier setting).

Vector $\overrightarrow{X}$ has to be large enough to contain each possible domain value. Also, since the Paillier cryptosystem is probabilistic, the elements $E(\cdot)$ (and the vectors too) are mutually computationally indistinguishable and do not reveal any information about plaintext values.

9.5.1 PrivSched-v1 Protocol Specification

To compute the homomorphic argmin, we use a tagging system over a vector-based representation, performed by each participant. First, the Initiator creates (and transfer to the server S) a vector-based representation of her costs, for each timeslots. Then, S sequentially asks other participants to update vectors with their own costs. (Recall that vectors do not reveal any information about the underlying inputs). Finally, S computes (and transfer to the Initiator) the homomorphic argmin and the Initiator learns the suitable timeslots (if any) upon decryption.

One crucial goal is to minimize computation overhead on the smartphones. Note that vbr's are relatively short, as we deal with small integers if small costs are chosen (e.g., $c_{max} = 10$). Nonetheless, we still want to minimize the number of exponentiations to compute the vbr. To this end, we compute single encryptions of $0, 1$, z, and a random $rand = E(0)$, where encryption of 0 is performed using a random number w, chosen with the same size as the Paillier modulus.

[1]E.g., the computation of homomorphic argmin may be useful for privacy-preserving data aggregation in sensor networks or urban sensing systems [144].

We then re-randomize the first element in vbr with a multiplication by $rand$. Next, we update $rand \leftarrow_r (rand)^{exp}$, where exp is a relatively small random exponent, and we continue with the next element. We describe the details of PrivSched-v1 below. The protocol is also illustrated in Figure 9.2.

Initialization

First, the Initiator P_1 generates Paillier public and private keys, denoted with pk_1 and sk_1, respectively. (In the rest of this section, all encryptions/decryptions are always performed using these keys, thus, to ease presentation, we omit them in our notation. If we need to specify the randomness used by the encryption algorithm, we use the following notation: $E(M, R)$ to denote encryption of M under pk_1 using the random value R).

Next, P_1 computes, for each time slot $j \in [1, m]$, the vbr $\overrightarrow{v_j}$:

$$\overrightarrow{v_j} = [\underbrace{E(0), \ldots, E(0)}_{c_{1:j}}, \underbrace{E(1), E(z), \ldots, E(z)}_{\vartheta - c_{1:j}}]$$

Finally, P_1 sends $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$, along with the identities of the other participants, to S.

Aggregation

After receiving the initial input from P_1, the server S sequentially forwards $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$ to each participant involved in the protocol.

Next, each P_i (for $i \in [2, N]$) adds her cost $c_{i:j}$ to each vector $\overrightarrow{v_j}$ (for $j \in [1, m]$) by shifting the elements of each vector $c_{i:j}$ positions right, and replacing them by $E(0)$:

$$\overrightarrow{v_j} \leftarrow_r \overrightarrow{v_j} >> c_{i:j} \overset{\text{def}}{=} [\underbrace{E(0), \ldots, E(0)}_{c_{i:j}}, v_{j,1}, \ldots, v_{j,\vartheta - c_{i:j}}]$$

To mask her modifications, P_i re-randomizes the vectors $\overrightarrow{v_j}$'s by multiplying the generic element $v_{j,k}$ by a random $E(0)$. Finally, she sends the updated $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$ back to S.

This phase is repeated, sequentially, for each participant, $P_2, \ldots, P_N$: at the end S obtains the final $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$ where, for $j \in [1, m]$:

$$\overrightarrow{v_j} = [\underbrace{E(0), \ldots, E(0)}_{ac_j}, \underbrace{E(1), E(z), \ldots, E(z)}_{\vartheta - ac_j}]$$

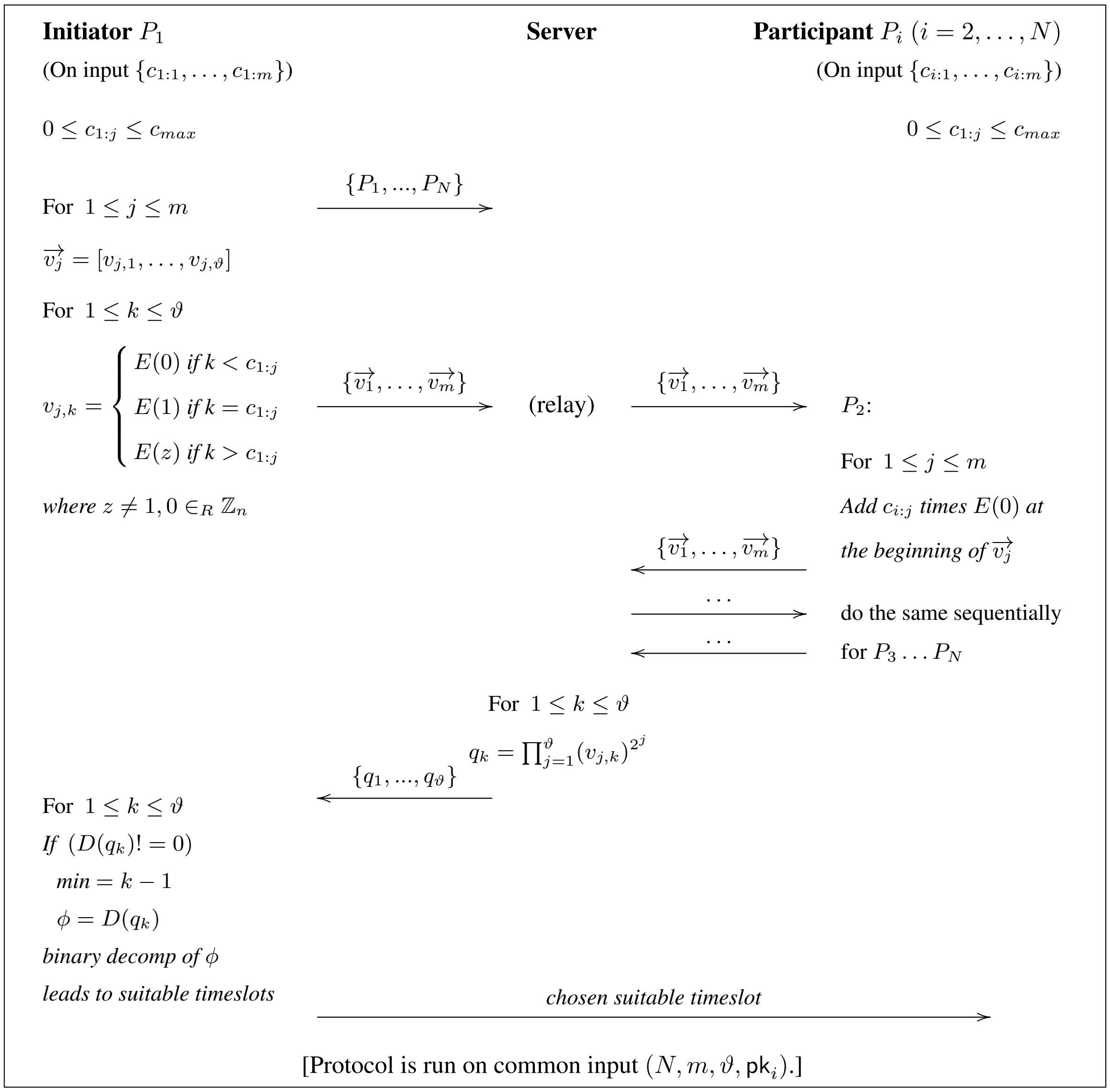

Initiator P_1
Server
Participant P_i $(i = 2, \ldots, N)$
(On input $\{c_{1:1}, \ldots, c_{1:m}\}$)
(On input $\{c_{i:1}, \ldots, c_{i:m}\}$)
$0 \leq c_{1:j} \leq c_{max}$
$0 \leq c_{1:j} \leq c_{max}$
For $1 \leq j \leq m$
$\{P_1, ..., P_N\}$
$\overrightarrow{v_j} = [v_{j,1}, \ldots, v_{j,\vartheta}]$
For $1 \leq k \leq \vartheta$
$v_{j,k} = \begin{cases} E(0) \textit{ if } k < c_{1:j} \\ E(1) \textit{ if } k = c_{1:j} \\ E(z) \textit{ if } k > c_{1:j} \end{cases}$
$\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$
(relay)
$\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$
P_2:
where $z \neq 1, 0 \in_R \mathbb{Z}_n$
For $1 \leq j \leq m$
Add $c_{i:j}$ *times* $E(0)$ *at*
the beginning of $\overrightarrow{v_j}$
$\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$
$\ldots$
do the same sequentially
$\ldots$
for $P_3 \ldots P_N$
For $1 \leq k \leq \vartheta$
$q_k = \prod_{j=1}^{\vartheta} (v_{j,k})^{2^j}$
$\{q_1, ..., q_\vartheta\}$
For $1 \leq k \leq \vartheta$
If $(D(q_k)! = 0)$
min $= k - 1$
$\phi = D(q_k)$
binary decomp of ϕ
leads to suitable timeslots
chosen suitable timeslot
[Protocol is run on common input $(N, m, \vartheta, \mathsf{pk}_i)$.]

Figure 9.2: Our PrivSched-v1 Protocol.

Minimization

Upon receiving the final $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$, S computes the homomorphic argmin: first, S raises each element of $\overrightarrow{v_j}$ to 2^j; then, it computes a vector $\overrightarrow{q}$. (The sum of all tags should not exceed the size of the Paillier modulus):

$$\begin{aligned}
\overrightarrow{v'_j} &= (\overrightarrow{v_j})^{2^j} = [(v_{j,1})^{2^j}, (v_{j,2})^{2^j}, ..., (v_{j,\vartheta})^{2^j}] \\
\overrightarrow{q} &= [q_1, q_2, \ldots, q_\vartheta] \overset{\text{def}}{=} [\prod_{j=1}^{m} v'_{j,1}, \ldots, \prod_{j=1}^{m} v'_{j,\vartheta}] = \\
&= [\underbrace{E(0), \ldots, E(0)}_{min}, q_{min+1}, \ldots, q_\vartheta]
\end{aligned}$$

Next, S sends $\overrightarrow{q}$ to the Initiator P_1, that decrypts each element of $\overrightarrow{q}$ using sk_1. The minimum aggregated cost corresponds to the number of consecutive 0's in the first positions of $\overrightarrow{q}$. Also, q_{min+1} decrypts to the sum of tags corresponding to the timeslot(s) producing the minimum aggregated cost. We denote this sum with ϕ. P_1 retrieves the index of this timeslot by observing which bits are equal to 1 in the binary decomposition of ϕ. P_1 may additionally retrieve the 2^{nd} minimum timeslot by subtracting $(\phi \cdot z)$ from the non-null decrypted elements of $\overrightarrow{q}$. Iterating this method leads to retrieval of all timeslots with aggregated cost less than ϑ.

Observe that ϑ is a system parameter and can be tuned to meet different requirements. Smaller values of ϑ result into a smaller $\overrightarrow{q}$ vector: this would reduce computations performed by participants and by the server, as well as the total bandwidth overhead. Also, the knowledge of P_1 on aggregated cost values will be limited to fewer timeslots, while the likelihood that the protocol execution terminates with no suitable timeslot would be increased. Therefore, an appropriate choice of ϑ depends on the specific setting and should be agreed on by the participants.

At the end of the protocol, only P_1 learns the timeslots with aggregated cost smaller than the threshold, and takes appropriate actions to schedule a meeting. Standard encryption techniques can be used by P_1 to multi-cast the meeting invitation to the other participants.

Complexity of PrivSched-v1. During each protocol execution, the Initiator performs 4 Paillier encryptions: $E(1)$, $E(0)$, $E(z)$ and $rand = E(0, w)$, where w is a random value chosen with the same size as the Paillier modulus. To create vector $\overrightarrow{v_1}$, the Initiator selects the encryptions $E(0)$, $E(1)$ or $E(z)$, and multiplies them by a different $rand$ to perform re-randomization. Thus, the Initiator performs $(m \cdot \vartheta)$ multiplications and small exponentiations (to create $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$), and at most ϑ decryptions (to retrieve suitable timeslots). Alternatively, to create the vector, a pool of pre-computed $E(0)$'s can be used: in this case, the Initiator performs 3 encryptions and $(m \cdot \vartheta)$ multiplications with pre-computed $E(0)$'s. All other participants perform 2 encryptions ($E(0)$ and $rand$), and $(m \cdot \vartheta)$ multiplications and small exponentiations (to update the vectors).

If pre-computations are used, they perform 1 encryption and $(m \cdot \vartheta)$ multiplications. The server performs $(m \cdot \vartheta)$ exponentiations for the tagging and $(m \cdot \vartheta)$ multiplications to create vector $\overrightarrow{q}$. The communication overhead amounts to $(m \cdot \vartheta)$ ciphertexts (i.e., 2048-bit each) for all participants. Additionally, the Initiator receives ϑ ciphertexts (in $\overrightarrow{q}$).

9.5.2 Privacy of PrivSched-v1

All vectors are encrypted under the Initiator's public key, thus, neither the participants nor the server can violate the privacy requirements described in Section 9.2.2, by virtue of the CPA-security of the Paillier cryptosystem [133]. In other words, if *Privacy w.r.t. Server* is not guaranteed, then one can construct an adversary violating the Decisional Composite Residuosity assumption [133].

Then, *Privacy w.r.t. Other Participants* is straightforwardly guaranteed, since participants only get (from the server) the minimum of the aggregated costs and no other information about other participants' inputs.

Given that the server and the Initiator do not collude, the server computes the minimization, *blindly*, i.e., over encrypted data. Collusion between the server and the Initiator may lead to violate other participants' privacy, while a collusion between the server and other participants would be irrelevant, as they could not decrypt. However, if $N - 1$ participants colluded with the server against the Initiator, they could recover (potentially) valuable information from the output of the protocol. Collusions can be thwarted using Threshold Cryptography [140] and, specifically, the threshold version of Paillier cryptosystem, presented in [63]. Recall that in a (t, N)-threshold cryptosystem, the private key to decrypt is shared between all the participants. Hence, to decrypt a ciphertext, t over N participants should agree and execute some computations to jointly perform the decryption. Using a threshold version of Paillier cryptosystem ensures that the Initiator, even if colluding with the server, cannot maliciously decrypt more information than she should. Note, however, that at this stage our protocol implementations do not address collusion between participants.

9.6 PrivSched-v2

A potential slow-down in PrivSched-v1 may result from *each* participant computing or updating the vector-based representation (vbr) of her costs. This increases the communication overhead and requires each participant to compute operations sequentially, one after the other. Therefore, we introduce a new protocol variant, called *PrivSched-v2*. Participants encrypt directly their costs without using the vbr, thus, we can perform the aggregation in parallel, instead of sequentially, since we no longer use the vbr at each participant's side. This modification reduces server's waiting time and improves the overall performance. The server relies on a mapping, pre-computed by the Initiator, to transform each aggregated cost into its vbr and perform the homomorphic argmin. Again, our improvements might be of independent interest in the context of homomorphic computation.

9.6.1 PrivSched-v2 Protocol Specification

We present our modified protocol – namely, PrivSched-v2 – below. The PrivSched-v2 protocol is also illustrated in Figure 9.3.

Setup

First, each participant P_i computes public/private keypairs $(\mathsf{pk}_i, \mathsf{sk}_i)$. Public keys, pk_i, are distributed, before protocol execution, using the server.

The Initiator P_1 computes a mapping, MAP, and sends it to the server S. S will use it during aggregation to transform each aggregated cost into the corresponding vbr. Assuming N_{max} is the maximum number of participants, $\vartheta_{max} = f(c_{max}, N_{max})$, and (a_1, y_1) are random values in the Paillier setting generated by P_1, MAP is pre-computed by P_1 as follows:

$$\begin{aligned}
E(0, a_1) &\longrightarrow [E(1), E(z), E(z), E(z), \ldots, E(z)] \\
E(1, a_1 y_1) &\longrightarrow [\underbrace{E(0)}_{1}, E(1), E(z), E(z), \ldots, E(z)] \\
\ldots &\longrightarrow \ldots \\
E(\vartheta_{max}, a_1 y_1^{\vartheta_{max}}) &\longrightarrow [\underbrace{E(0), E(0), E(0), E(0), \ldots, E(0)}_{\vartheta_{max}}]
\end{aligned}$$

Note that y_1 and a_1 randomize the mapping and prevent S from learning any private information. y_1 is raised to the value of the aggregated costs in order to randomize the costs differently. Since $(y_1)^0 = 1$, we add the random value a_1 to address the special case where an aggregated cost equals 0. In addition, a_1 simplifies future updates: P_i would only need to change this random value to re-randomize the mapping without performing exponentiations again. Finally, we let P_1 shuffle the mapping to randomize the position of each aggregated cost. (Recall that encryptions are again computed using pk_1).

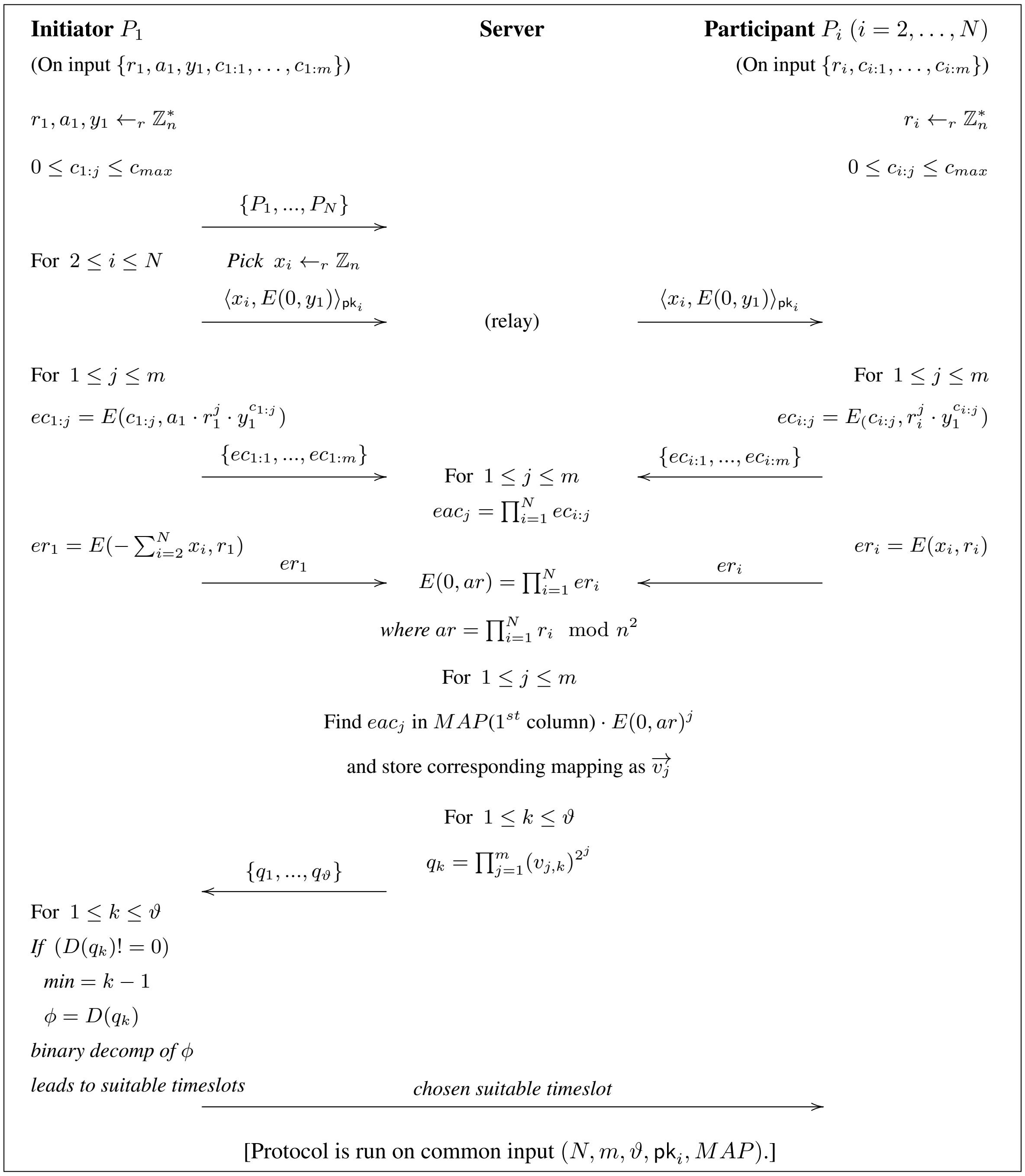

Initiator P_1
Server
Participant P_i $(i = 2, \ldots, N)$
(On input $\{r_1, a_1, y_1, c_{1:1}, \ldots, c_{1:m}\}$)
(On input $\{r_i, c_{i:1}, \ldots, c_{i:m}\}$)
$r_1, a_1, y_1 \leftarrow_r \mathbb{Z}_n^*$
$r_i \leftarrow_r \mathbb{Z}_n^*$
$0 \leq c_{1:j} \leq c_{max}$
$0 \leq c_{i:j} \leq c_{max}$
$\{P_1, ..., P_N\}$
For $2 \leq i \leq N$
Pick $x_i \leftarrow_r \mathbb{Z}_n$
$\langle x_i, E(0, y_1)\rangle_{\mathsf{pk}_i}$
(relay)
$\langle x_i, E(0, y_1)\rangle_{\mathsf{pk}_i}$
For $1 \leq j \leq m$
For $1 \leq j \leq m$
$ec_{1:j} = E(c_{1:j}, a_1 \cdot r_1^j \cdot y_1^{c_{1:j}})$
$ec_{i:j} = E_(c_{i:j}, r_i^j \cdot y_1^{c_{i:j}})$
$\{ec_{1:1}, ..., ec_{1:m}\}$
For $1 \leq j \leq m$
$\{ec_{i:1}, ..., ec_{i:m}\}$
$eac_j = \prod_{i=1}^N ec_{i:j}$
$er_1 = E(-\sum_{i=2}^N x_i, r_1)$
$er_i = E(x_i, r_i)$
er_1
$E(0, ar) = \prod_{i=1}^N er_i$
er_i
where $ar = \prod_{i=1}^N r_i \mod n^2$
For $1 \leq j \leq m$
Find eac_j in $MAP(1^{st}$ column$) \cdot E(0, ar)^j$
and store corresponding mapping as $\overrightarrow{v_j}$
For $1 \leq k \leq \vartheta$
$\{q_1, ..., q_\vartheta\}$
$q_k = \prod_{j=1}^m (v_{j,k})^{2^j}$
For $1 \leq k \leq \vartheta$
If $(D(q_k)! = 0)$
min $= k - 1$
$\phi = D(q_k)$
binary decomp of ϕ
leads to suitable timeslots
chosen suitable timeslot
[Protocol is run on common input $(N, m, \vartheta, \mathsf{pk}_i, MAP)$.]

Figure 9.3: Our PrivSched-v2 Protocol.

<u>Initialization</u>

First, P_1 picks a random r_1, then, for each time slot $j \in [1, m]$, computes and sends S the value $ec_{1:j}$:

$$ec_{1:j} = E\left(c_{1:j},\ a_1 \cdot (r_1)^j \cdot (y_1)^{c_{1:j}}\right) =$$
$$= E(c_{1:j}, a_1) \cdot (E(0, r_1))^j \cdot (E(0, y_1))^{c_{1:j}}$$

Next, P_1 picks $N-1$ random values, $(x_2, \dots, x_N)$. For each $i \in [2, N]$, P_1 encrypts $\langle x_i, E(0, y_1) \rangle$ under the public key pk_i of each participant P_i and sends them to S, that forwards to the corresponding participant. (We need this encryption to hide these values from the server).

Finally, P_1 computes $er_1 = E(-(\sum_{i=2}^{N} x_i), r_1)$ and sends it to S.

Aggregation

First, each participant P_i generates a random value r_i in the Paillier setting generated by P_1. Then, for each timeslot $j \in [1, m]$, P_i encrypts her cost $c_{i:j}$, using as randomness the decrypted $E(0, y_1)$:

$$\begin{aligned} ec_{i:j} &= E\left(c_{i:j},\ r_i \cdot (y_1)^{c_{i:j}}\right) \\ &= E(c_{i:j}, 1) \cdot (E(0, r_i))^j \cdot (E(0, y_1))^{c_{i:j}} \end{aligned}$$

Then, P_i sends S $\{ec_{i:1}, \dots, ec_{i:m}\}$ and $E(x_i, r_i)$.

Next, S computes the (encrypted) aggregated cost of each timeslot $j \in [1, m]$, using Paillier's homomorphism:

$$eac_j \overset{\text{def}}{=} \prod_{i=1}^{N} ec_{i:j} = E\left(ac_j,\ (ar)^j \cdot a_1 \cdot (y_1)^{ac_j}\right)$$

where $ar \overset{\text{def}}{=} \prod_{i=1}^{N} r_i \bmod n^2$ (where n is the public Paillier modulus of pk_1). Finally, S reconstructs:

$$E(0, ar) = E(-(\sum_{i=2}^{N} x_i), r_1) \cdot E(x_2, r_2) \cdot \ldots \cdot E(x_N, r_N)$$

Minimization

In this phase, the server S computes the (encrypted) minimum aggregated cost and sends it to P_1. To this end, S first transforms each encrypted aggregated cost (eac_j) into its vbr. Next, S computes the vbr using the mapping MAP and the value $E(0, ar)$, namely, for each timeslot $j \in [1, m]$, S:

(i) Multiplies the 1st column of MAP by $E(0, ar)$ and gets: (recall that the position of each aggregated cost is shuffled in the mapping stored by the server).

$$\begin{aligned}
E(0, ar^j a_1) &\longrightarrow [E(1), E(z), E(z), \ldots, E(z)] \\
E(1, ar^j a_1 \cdot y_1) &\longrightarrow [\underbrace{E(0)}_{1}, E(1), E(z), \ldots, E(z)] \\
E(2, ar^j a_1 \cdot y_1^2) &\longrightarrow [\underbrace{E(0), E(0)}_{2}, E(1), E(z), \ldots, E(z)] \\
\ldots &\longrightarrow \ldots \\
E(\vartheta_{max}, ar^j a_1 \cdot {y_1}^{\vartheta_{max}}) &\longrightarrow [\underbrace{E(0), E(0), E(0), \ldots, E(0)}_{\vartheta_{max}}]
\end{aligned}$$

(ii) Finds eac_j in MAP and stores the right side of the mapping as $\overrightarrow{v_j}$.

(iii) Increments j and goes back to (i).

Then, S starts the homomorphic argmin using vectors $\overrightarrow{v_j}$, i.e., the vbr of each aggregated cost, using the following tagging technique. The server raises each element of $\overrightarrow{v_j}$ to 2^j (for $j \in [1, m]$):

$$\overrightarrow{v_j'} = (\overrightarrow{v_j})^{2^j} = [(v_{j,1})^{2^j}, (v_{j,2})^{2^j}, ..., (v_{j,\vartheta})^{2^j}]$$

Next, the server computes the vector $\overrightarrow{q}$ and sends it to P_1:

$$\begin{aligned}
\overrightarrow{q} = [q_1, q_2, \ldots, q_\vartheta] &\stackrel{\text{def}}{=} \left[\prod_{j=1}^{m} v'_{j,1}, \ldots, \prod_{j=1}^{m} v'_{j,\vartheta}\right] \\
&= [\underbrace{E(0), \ldots, E(0)}_{min}, q_{min+1}, \ldots, q_\vartheta]
\end{aligned}$$

Finally, P_1 decrypts each element of $\overrightarrow{q}$ using sk_1. As in *PrivSched-v1*, the minimum aggregated cost corresponds to the number of consecutive 0's in the first positions of $\overrightarrow{q}$. q_{min+1} decrypts to the sum of tags corresponding to the timeslot(s) producing the minimum aggregated cost. Again, we denote this sum with ϕ. P_1 retrieves the index of this timeslot by observing which bits are equal to 1 in the binary decomposition of ϕ. P_1 may additionally retrieve the 2^{nd} minimum timeslot by subtracting $(\phi \cdot z)$ from the non-null decrypted elements of $\overrightarrow{q}$. Iterating this method leads to retrieval of all timeslots with aggregated cost smaller than ϑ.

At the end of the protocol, only P_1 learns the timeslots with aggregated cost smaller than the threshold, and takes appropriate actions to schedule the meeting. Again, standard encryption techniques can be used by P_1 to multi-cast meeting invitation to the other participants.

9.6.2 Complexity of PrivSched-v2

During each protocol execution, the Initiator performs, in the worst case, $(c_{max} + 3)$ Paillier encryptions and $(c_{max} + 3m)$ multiplications to create $ec_{1:j}$ for $j \in [1, m]$. In addition, the Initiator needs $N - 1$ Paillier encryptions to protect $\langle x_i, E(0, y_1)\rangle$ and at most ϑ decryptions to retrieve suitable timeslots. Each participant performs one decryptions to get $\langle x_i, E(0, y_1)\rangle$, and $(c_{max} + 2)$ Paillier encryptions plus $(c_{max} + 3 \cdot m)$ multiplications to create encrypted costs. The server performs $(m \cdot \vartheta)$ exponentiations for tagging and $(m \cdot \vartheta)$ mults to create $\overrightarrow{q}$. The communication overhead amounts to m ciphertexts (i.e., 2048-bit each) for all participants. Additionally, the Initiator receives ϑ ciphertexts (in $\overrightarrow{q}$).

9.6.3 Privacy of PrivSched-v2

Participants only receive $E(0, y_1)$ and $E(x_i)$, encrypted under the Initiator's public key, thus, similar to PrivSched-v1, neither participants nor the server can violate the privacy requirements described in Section 9.2.2, by virtue of the CPA-security of the Paillier cryptosystem [133]. The Initiator gets the vector $\overrightarrow{q}$ containing only suitable timeslots (i.e., whose aggregated cost is smaller than ϑ). Considerations about possible collusion are the same as in PrivSched-v1, thus, we do not repeat them here.

Since equal aggregated costs may appear at the same line of the mapping, the server could detect repeated aggregated costs. Considering that participants provide weekly or monthly calendar, the highest costs is probably used for nights, weekends, holidays, and busy timeslots are most likely the ones appearing the most. Therefore, the server could infer timeslots with the highest cost counting the number of collisions in the mapping. One possible countermeasure is to use a secret permutation of the timeslots known only by the participants. This way, information obtained by the server is obfuscated and she cannot get any information about date/time of repeated timeslots. Removing nights, weekends, or holidays, will also modify the distribution of busy timeslots which will tend to be closer to the most available ones, thus, reducing the probability of correctly guessing which timeslots correspond to the highest cost.

9.7 Symmetric-Key Private Scheduling: S-PrivSched

The PrivSched-v1 and v2 protocols involve a limited yet non-negligible number of public-key cryptographic operations. Although our experimental analysis (presented in Section 9.8) provides encouraging performance results for PrivSched-v1 and v2, we now present yet another construction, *S-PrivSched* (Symmetric PrivSched), that only involves symmetric-key operations, and reduces computational and communication overheads. Our intuition is the following: Participants establish a shared secret, e.g., the Initiator broadcasts it using an "out-of-band" channel, such as SMS, or she encrypts it using the public key of each of the other participants. Then, we use an additively homomorphic symmetric-key cryptosystem (e.g., the one proposed in [34]) to perform cost aggregation.

Preamble. S-PrivSched uses the cryptosystem in [34], a tailored modification of the Vernam cipher [146] to allow plaintext addition to be done in the ciphertext domain, proven to achieve security (more precisely, indistinguishability) against chosen plaintext attacks (IND-CPA). Below, we review its algorithms:

- *Encryption:*
  1. Represent the message msg as an integer $m \in [0, M-1]$, where M is the modulus. (See below).
  2. Let k be randomly generated keystream, where $k \in [0, M-1]$.
  3. Compute $ct = Enc_k(m) = (m + k) \bmod M$.
- *Decryption:* $Dec_k(ct) = (ct - k) \bmod M = m$
- *Addition of ciphertexts*: Let $ct_1 = Enc_{k_1}(m_1)$, $ct_2 = Enc_{k_2}(m_2)$. Aggregated ciphertext: $ct = ct_1 + ct_2 \bmod M = Enc_k(m_1 + m_2)$ (where $k = k_1 + k_2$).

Observe that M needs to be sufficiently large: $0 \leq msg < M$, or $\sum_{i=1}^{N} m_i < M$ if N ciphertexts are added.

Although above we assume k to be randomly generated at all times, one can generate a single master key K and derive successive keys as the output of an appropriate pseudorandom function [78], or the output of a length-preserving hash function (as shown in [34]).

9.7.1 S-PrivSched Protocol Specification

We now present S-PrivSched (also illustrated in Figure 9.4). We use the same system model introduced for the PrivSched-v1 and v2 constructs, thus, we do not repeat it here.

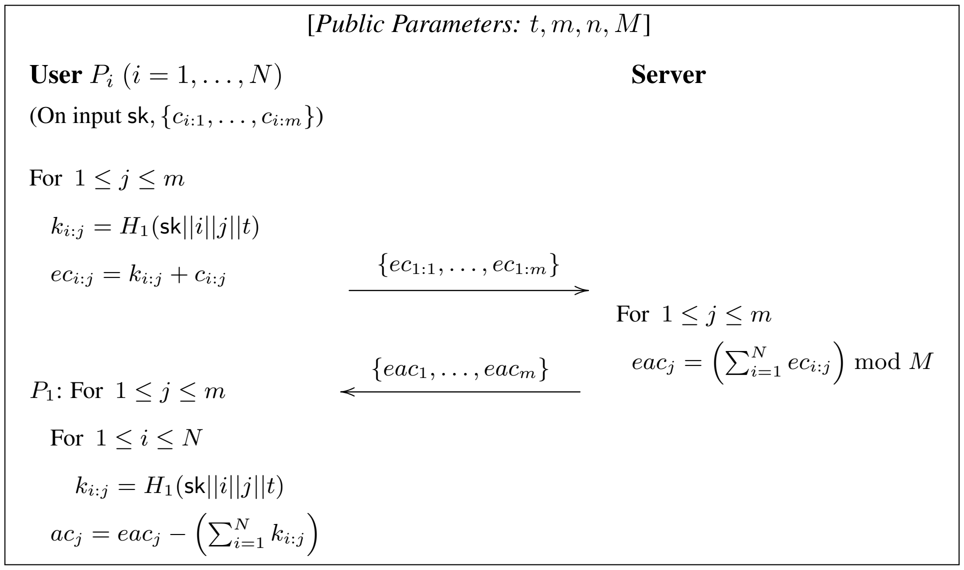

[*Public Parameters:* t, m, n, M]
User P_i $(i = 1, \ldots, N)$
(On input sk, $\{c_{i:1}, \ldots, c_{i:m}\}$)
For $1 \le j \le m$
$k_{i:j} = H_1(\mathsf{sk}||i||j||t)$
$ec_{i:j} = k_{i:j} + c_{i:j}$
$\{ec_{1:1}, \ldots, ec_{1:m}\}$
Server
For $1 \le j \le m$
$eac_j = \left(\sum_{i=1}^{N} ec_{i:j}\right) \bmod M$
$\{eac_1, \ldots, eac_m\}$
P_1: For $1 \le j \le m$
For $1 \le i \le N$
$k_{i:j} = H_1(\mathsf{sk}||i||j||t)$
$ac_j = eac_j - \left(\sum_{i=1}^{N} k_{i:j}\right)$

Figure 9.4: Our S-PrivSched Protocol.

Setup

The Initiator, P_1, selects $M > (c_{max} \cdot N_{max})$, where c_{max} is the maximum cost participants may associate to a timeslot and N_{max} the maximum number of participants.

P_1 also selects a random $\mathsf{sk} \in [0, M-1]$ and broadcasts sk and a nonce t to participants $P_2, \cdots, P_N$ over an out-of-band channel. In this version of the protocol, we assume that sk is sent over SMS, thus, we assume the cellular network operator does not eavesdrop SMS traffic and collude with the (scheduling) service provider. One can also relax this assumption by using the public keys of all other participants to encrypt sk and share it over insecure channels. However, this comes at the cost of $N-1$ additional asymmetric encryptions.

Initialization

Each participant (Initiator included), P_i (for $i \in [1, N]$), for each timeslot $j \in [1, m]$, computes:

- $k_{i:j} = H_1(\mathsf{sk}||i||j)$
- $ec_{i:j} = k_{i:j} + c_{i:j}$

and sends $\{ec_{i:1}, \ldots, ec_{i:m}\}$ to the server S.

Aggregation

Upon receiving $\{ec_{i:j}\}$ $i \in [1, N], j \in [1, m]$, S aggregates the costs, i.e.:

- Computes $eac_j = (\sum_{i=1}^{N} ec_{i:j}) \bmod M \; \forall j \in [1, m]$
- Sends $\{eac_1, \ldots, eac_m\}$ to P_1

Minimization

P_1 obtains the aggregated costs upon decryption:

- $k_{i:j} = H_1(\mathsf{sk}||i||j) \; \forall i \in [1, N], j \in [1, m]$
- $ac_j = ec_j - (\sum_{i=1}^{N} k_{i:j}) \; \forall j \in [1, m]$

Finally, P_1 obtains aggregated costs for each timeslot: she computes the timeslot(s) with minimum cost and takes appropriate actions to schedule the meeting.

Complexity of S-PrivSched. All the participants only perform symmetric key operations, specifically, $(N \cdot m)$ decryptions/subtractions (Initiator) and m encryptions/additions (participants). Communication overhead amounts to m ciphertexts for each participant and $N \cdot m$ for the server.

9.7.2 Privacy of S-PrivSched

Privacy of S-PrivSched stems from the security of the cryptosystem in [34]. *Privacy w.r.t. the Server S* is guaranteed as S only receives ciphertexts. Also, *Privacy w.r.t. Other Participants* is straightforward since participants never receive other participants' costs. Also, P_1 only obtains aggregated costs. Compared to Paillier-based PrivSched constructs, however, P_1 obtains aggregated costs for *all* timeslots. In fact, we do not know how to let the server compute the minimization using the homomorphic symmetric-key cryptosystem. Considerations and possible countermeasures about potential collusion are somewhat similar to PrivSched-v1, thus, we do not repeat them here.

9.8 Performance Analysis

All proposed protocols were implemented on Nokia N900 smartphones. We developed a prototype application for Private Scheduling, instantiating PrivSched-v1 and v2, S-PrivSched, and we used PIS to implement a threshold binary version of Private Scheduling (see scenario (4) in Section 9.3). Recall that the four algorithms provide (slightly) different privacy properties (reviewed below) and require somewhat different system settings (e.g., key management). This leads to differing computational and communication costs. Thus, our goal is not to compare their performance, rather, to assess their practicality for real-world deployment and to provide an indication of the overhead experienced by users.

In our prototypes, we used the Qt framework [129] and the open-source cryptographic libraries libpaillier [18] and libgmp [64], on several Nokia N900 devices (equipped with a 600 MHz ARM processor and 256 MB of RAM). We also used a Dell PC with 2.27 GHz CPU (16 cores) and 50 GB of RAM to instantiate the semi-trusted server. We ran tests with an increasing number of participants (ranging from 2 to 8) and a fixed number of timeslots, i.e., $7 \cdot 24 = 168$ to cover one week with one-hour granularity, with randomly-generated calendars. We define $c_{max} = 10$ and $\vartheta = \frac{c_{max}}{2} \cdot N = 5N$. In our tests, we used a local 802.11 Wi-Fi network.

For every algorithm, we measured the *processing time* of the server, the Initiator, and of a single participant. The latter is computed as the average of processing times of all participants, excluding the Initiator. We also evaluated the *communication overhead*. Results are averaged over 100 iterations. We also ran tests for a baseline protocol providing no privacy (i.e., calendars were transmitted to the server, that computed the minimization in the clear).

Results are plotted in Figure 9.5, 9.6, and 9.7. The top row measures bytes exchanged (using logarithmic scale), bottom row – processing times in milliseconds. Confidence intervals were very small and omitted for visibility. (Standard deviation was smaller than 282 bytes and $280ms$).

We make the following remarks:

1. As expected, PrivSched-v1, compared to its most similar counterpart (PrivSched-v2), incurs an increased bandwidth overhead, that grows with the number of participants. Therefore, we recommend its use only in settings where participants do not want the server to learn that some timeslots may have equal aggregated costs.

2. PrivSched-v2 incurs a reasonable overhead and scales efficiently even when the number of participants is increasing. Nonetheless, recall that the mapping creates a small privacy degradation. Also, some limited information has to be pre-exchanged among the participants.

3. S-PrivSched incurs very low computational and communication overhead – almost negligible

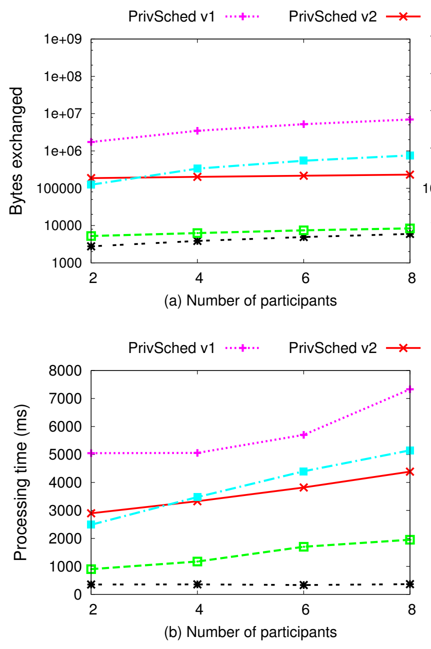


Figure 9.5: Initiator

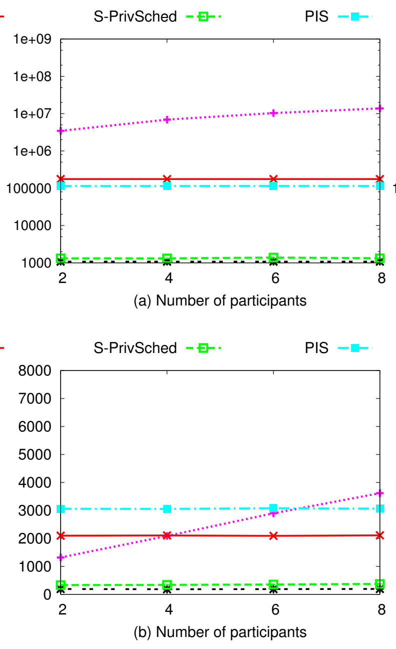


Figure 9.6: Each participant

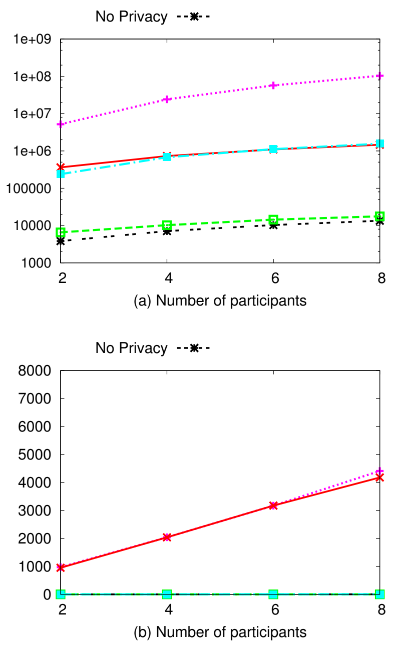


Figure 9.7: Server

compared to the baseline protocol with no privacy. However, it trades-off some of the privacy properties, as it reveals aggregated costs of *all* timeslots to the Initiator, and requires the distribution of a shared secret (e.g., via SMS).

4. PIS addresses several different applications beyond scheduling; nonetheless, it is worth observing that it is practical enough for actual deployment. The computational and communication overheads are independent of the number of participants, and incur a constant overhead, except for the Initiator and the server – a reasonable assumption considering that the Initiator is the one willing to start off the protocol.

We conclude that our implementations, though achieving different privacy properties in different system settings, are practical enough for deployment. Even the most computationally demanding protocols, such as PrivSched-v1-2, only require a few seconds in most realistic settings.

9.9 Related Primitives

The specific problems and applications investigated in this chapter bear some resemblance with a few related primitives, that we review below. In doing so, we first discuss related cryptographic constructs addressing multi-party computation problems. Next, we analyze techniques employing third-party services, and, finally, protocols for private scheduling and privacy-preserving interest sharing.

Related Cryptographic Primitives. *Secure multi-party Computation* (SMC) [79] allows several players, each equipped with a private input, to compute the value of a public function f over all inputs. Players only learn the output of f, and nothing else (beyond what revealed by the computation). Generic SMC would indeed solve all the problems we consider, however, it is well-known that MPC involves several rounds of computations, as well as computational and communication costs far too high to be deployed on smartphones, while special-purpose protocols are generally much more efficient.[2] *Multi-party Private Set Intersection* [108] allows several players to privately compute the intersection of their private sets, such that they learn nothing beyond the intersection. Thus far, however, only techniques limited to the two-party set intersection have achieved practical efficiency. Whereas, constructs for multiple players require a number of long exponentiations quadratic in the size of the sets [108] – well beyond the requirements of our smartphone setting. Further, PSI constructs cannot be used for the non-binary private scheduling problem. Finally, note that, to the best of our knowledge, there is no known construct for a private *threshold* set-intersection problem, that would be relevantly close to PIS, but only for threshold set-union [108].

Server-assisted Computations. Over the last years, semi-trusted third parties have been employed to assist privacy-preserving computations. We do not consider naïve approaches using fully trusted third parties (whereto all players surrender their inputs), or requiring the existence of specific hardware or devices, such as a Trusted Platform Modules. (Whereas, semi-trusted parties are only assumed not to collude with other players). Following Beaver's intuition [10], [26] introduces a semi-trusted third party to obliviously compare two numbers (e.g., to solve the millionaire's problem [148]). [53] uses the same intuition to solve the scalar product problem. Note, however, that these techniques are only designed for two parties and it is not clear how to adapt them to a multi-party setting. Also, to the best of our knowledge, there is no work exploring such intuition in the context of smartphone applications.

Private Discovery of Common Contacts. Another related problem occurs when two unfamiliar users want to privately discover their common contacts, e.g., reveal to each other only the contacts that they share. For instance, a smartphone user would like to interact with other users in physical proximity (e.g., in a bar or on the subway), given that they have some common friends on a given

[2]For instance, the communication complexity of MPC grows quadratically with the number of participants.

social network, e.g., Facebook. To this end, our preliminary results in [51] introduce the concept of Private Contact Discovery and propose a cryptographic primitive involving two users, on input their contact lists, that outputs only the list of mutual contacts (if any). The protocol prevents users from claiming unwarranted friendships by means of contact certification.

Private Scheduling. To the best of our knowledge, there is no protocol targeting the private scheduling problem in the setting of smartphone users. Prior work includes distributed constraint satisfaction in a fully-distributed approach [152, 149, 110, 106, 89]. These techniques incur high computation and communication overhead and are unpractical for mobile environments. Finally, a protocol for ***binary*** Private Scheduling (based on homomorphic encryption) has been presented for smartphone users, also relying on a semi-trusted server [19].

Private Interest Sharing. Besides primitives discussed above, several techniques have focused on problems similar to Private Interest Sharing. The work in [151] proposes protocols for the *nearby-friend* problem, i.e., to let two users learn whether their distance is smaller than a given radius. It uses homomorphic encryption [133] to compute algebraic operations over encrypted data. One of the proposed protocols, Louis, relies on a third user to assist computation and reduce overhead, i.e., it acts as a semi-trusted party. Thus, Louis appears somewhat similar to applying PIS to the nearby-friend problem. However, it is not clear how to extend Louis to a multi-party setting, e.g., to learn whether (at least) ϑ friends are nearby. Next, although in Louis input locations are two-dimension coordinates, while we only consider locations mapped to tags or cells, Louis involves more communication steps (four vs. two in PIS) and more expensive cryptographic operations (Paillier encryptions vs 160-bit exponentiations). Protocols that do not involve a semi-trusted party, such as Lester [151], incur much higher overhead and do not scale to multiple users, even if locations are mapped to cells, such as in Pierre [151], Wilfrid [150], and NFP [37]. Finally, the work in [126] proposes private testing of proximity, however, for only two participants. Also, the position paper in [135] discusses how Location-Based Social Applications (LBSAs) should process friends' location coordinates only in their encrypted form. Furthermore, some recent results addressed location privacy concerns in proximity-based services [117]. Finally, some of the applications envisioned in the context of PIS or Private Scheduling (e.g., polls) also resemble *e-voting* problems [81]. In reality, however, e-voting involves several different entities with specific roles and has very different requirements.

Chapter 10

Conclusion and Open Problems

This dissertation motivated the need for efficient privacy-preserving sharing of sensitive information and addressed some important problems in the field. Its main contributions are:

1. PSI protocols that are appreciably more efficient than state-of-the-art. In particular, one PSI protocol is specifically geared for limited-resource devices.

2. PSI variants with stronger privacy properties – APSI and SHI-PSI.

3. A toolkit for practical privacy-preserving sharing of sensitive information, that enables private database querying using any efficient PSI instantiation.

4. Efficient cryptographic protocols for privacy protection in cooperative smartphone applications.

We conclude this dissertation by highlighting some open problems and items for future work:

Efficient Group Private Set Intersection. In this dissertation, we studied PSI protocols, secure under different assumptions and adversarial models. The traditional PSI formulation only includes two participants, server and client. However, it is not clear how to efficiently extend such techniques to scenarios where a group of n participants (with $n > 2$) wish to privately compute the intersection of their respective sets (without using a trusted or semi-trusted party). Prior work [108] proposed a protocol for *multi-party* PSI, however, its computational complexity is quadratic in the size of input sets. Therefore, multi-party PSI protocols with linear complexities still remains a challenging topic for further research.

Multiple Certification Authorities in Authorized Private Set Intersection. In Chapter 5, we introduced the concept of APSI in order to prevent clients from using frivolous input sets and to ensure that they only obtain duly authorized information. In our APSI protocols, inputs are certified by a Certification Authority (CA). Efficiently support of multiple CAs remains to be explored.

Size-Hiding Private Set Intersection Secure in Malicious Model. Chapter 7 proposed the first PSI construct with the *size-hiding* property, i.e., the size of client's set is (unconditionally) hidden from the other participant. Proposed protocols are efficient and provably secure under standard assumptions, however, only in the presence of semi-honest adversary. It is not clear whether it is possible to design Size-Hiding PSI protocols with malicious security, thus, motivating the need for further research.

Lowering Bandwidth Overhead in Database Querying. In Chapter 8, we introduced a toolkit for privacy-preserving sharing of sensitive information, with application to private database querying. Proposed techniques combine provable security with reasonable efficiency, however, they incur a computational and communication overhead linear in the size of the database. On the one hand, linear complexity is a strict lower bound for computation: in order to guarantee perfect query privacy, the database needs to "touch" every single record. On the other hand, linear communication overhead is impractical in the context of very large databases. In [50], our preliminary results suggest this overhead can be lowered by using a (semi-trusted) third party or a piece of trusted hardware.

Private Testing of Genomic Information. Recent advances in DNA sequencing technologies have put ubiquitous availability of fully-sequenced human genomes within reach. Common genomic applications and tests performed *in vitro* today will soon be conducted computationally, using digitized genomes. New applications will be developed as genome-enabled medicine becomes increasingly preventive and personalized, however, prompting significant privacy challenges associated with the possible loss, theft, or misuse of genomic data. As a result, one interesting research direction is to design appropriate cryptographic tools for genomic privacy protection. Our preliminary results [7] show how to enable privacy in genomic applications, e.g., paternity tests, genetic and personalized medicine testing.

Bibliography

[1] A. Acquisti. *Digital privacy: theory, technologies, and practices*. Auerbach Pub, 2008.

[2] Amazon. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2`, 2010.

[3] A. Amsterdam. Perspectives on the Fourth Amendment. *Minnesota Law Review*, 58:349, 1974.

[4] Assembly, U.N.G. Universal Declaration of Human Rights. *Resolution adopted by the General Assembly*, 10:12, 1948.

[5] G. Ateniese, E. De Cristofaro, and G. Tsudik. (If) Size Matters: Size-Hiding Private Set Intersection. In *PKC*, pages 156–173, 2011.

[6] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *TCC*, pages 137–156, 2007.

[7] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes. In *CCS*, 2011.

[8] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret handshakes from pairing-based key agreements. In *S&P*, pages 180–196, 2003.

[9] D. Baumer, J. Earp, and J. Poindexter. Internet privacy law: a comparison between the united states and the european union. *Computers & Security*, 23(5):400–412, 2004.

[10] D. Beaver. Commodity-based cryptography. In *STOC*, 1997.

[11] A. Beimel, Y. Ishai, E. Kushilevitz, and J. Raymond. Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval. In *FOCS*, pages 261–270, 2002.

[12] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO*, pages 108–125, 2009.

[13] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-Privacy in Public-Key Encryption. In *ASIACRYPT*, pages 566–582, 2002.

[14] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2003.

[15] M. Bellare and P. Rogaway. The exact security of digital signatures-How to sign with RSA and Rabin. In *EUROCRYPT*, pages 399–416, 1996.

[16] J. Benaloh and M. De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *EUROCRYPT*, pages 274–285, 1994.

[17] E. Bertino, J. Byun, and N. Li. Privacy-preserving database systems. *Foundations of Security Analysis and Design*, 2005.

[18] J. Bethancourt. Paillier Library. `http://acsc.cs.utexas.edu/libpaillier/`, 2010.

[19] I. Bilogrevic, M. Jadliwala, J.-P. Hubaux, I. Aad, and V. Niemi. Privacy-Preserving Activity Scheduling on Mobile Devices. In *CODASPY*, pages 261–272, 2011.

[20] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key Encryption with Keyword Search. In *EUROCRYPT*, pages 506–522, 2004.

[21] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[22] X. Boyen and B. Waters. Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles). In *CRYPTO*, pages 290–307, 2006.

[23] R. Bradshaw, J. Holt, and K. Seamons. Concealing complex policies with hidden credentials. In *CCS*, pages 146–157, 2004.

[24] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000.

[25] G. Brassard, C. Crépeau, and J. Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, pages 234–238, 1987.

[26] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *CCS*, pages 120–127, 1999.

[27] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy. Blind and Anonymous Identity-Based Encryption and Authorised Private Searches on Public Key Encrypted Data. In *PKC*, pages 196–214, 2009.

[28] J. Camenisch and A. Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT*, pages 93–118, 2001.

[29] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT*, pages 573–590, 2007.

[30] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[31] J. Camenisch and G. Zaverucha. Private Intersection of Certified Sets. In *Financial Cryptography and Data Security*, pages 108–127, 2009.

[32] R. Canetti, U. Friege, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. Technical report, MIT, 1996.

[33] Caslon Analytics. Consumer Data Losses. `http://www.caslon.com.au/datalossnote.htm`.

[34] C. Castelluccia, A. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(3):20, 2009.

[35] C. Castelluccia, S. Jarecki, and G. Tsudik. Secret Handshakes from CA-Oblivious Encryption. In *ASIACRYPT*, pages 293–307, 2004.

[36] A. Cavoukian. *Privacy by design*. Stanford Law School, 2010.

[37] S. Chatterjee, K. Karabina, and A. Menezes. A New Protocol for the Nearby Friend Problem. *Cryptography and Coding*, 2009.

[38] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1983.

[39] B. Chor, N. Gilboa, and M. Naor. Private Information Retrieval by Keywords. *Manuscript*, 1998.

[40] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of ACM*, 45(6):965–981, 1998.

[41] S. Chow, J. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *NDSS*, 2009.

[42] J. Coron and D. Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In *EUROCRYPT*, pages 91–101, 2000.

[43] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):185, 2000.

[44] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient Robust Private Set Intersection. In *ACNS*, pages 125–142, 2009.

[45] J. Daeman and V. Rijmen. The Rijndael Block Cipher. *AES Proposal*, 1999.

[46] E. De Cristofaro, A. Durussel, and I. Aad. Reclaiming Privacy for Smartphone Applications. In *PERCOM*, 2011.

[47] E. De Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-preserving policy-based information transfer. In *PETS*, 2009.

[48] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *ASIACRYPT*, pages 213–231, 2010.

[49] E. De Cristofaro, Y. Lu, and G. Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In *TRUST*, pages 239–253, 2011.

[50] E. De Cristofaro, Y. Lu, and G. Tsudik. Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information (Extended Version). *Cryptology ePrint Archive*, 2011. `http://eprint.iacr.org/2011/113`.

[51] E. De Cristofaro, M. Manulis, and B. Poettering. Private Discovery of Common Social Contacts. In *ACNS*, pages 147–165, 2011.

[52] E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols with Linear Complexity. In *Financial Cryptography and Data Security*, pages 143–159, 2010.

[53] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW*, pages 127–135, 2002.

[54] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on Information Theory*, 31(4):469–472, 1985.

[55] L. Ertaul and Vaidehi. Computing Aggregation Function Minimum/Maximum using Homomorphic Encryption Schemes in Wireless Sensor Networks. In *ICWN*, pages 186–192, 2007.

[56] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[57] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS*, pages 211–222, 2003.

[58] Federal Bureau of Investigation. Terrorist Screening Center. `http://www.fbi.gov/terrorinfo/counterrorism/tsc.htm`.

[59] Federal Information Processing Standards. Secure Hash Standard. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`, 2002.

[60] H. Federrath. Privacy Enhanced Technologies: Methods, markets, misuse. *Trust, Privacy and Security in Digital Business*, pages 1–9, 2005.

[61] A. Fiat and M. Naor. Broadcast Encryption. In *CRYPTO*, pages 480–491, 1993.

[62] M. Fischlin, B. Pinkas, A. Sadeghi, T. Schneider, and I. Visconti. Secure set intersection with untrusted hardware tokens. In *CT-RSA*, pages 1–16, 2011.

[63] P. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography*, pages 90–104. Springer, 2001.

[64] Free Software Foundation. The GNU MP Bignum Library. `http://gmplib.org/`.

[65] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[66] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[67] J. Freudiger. *When Whereabouts is No Longer Thereabouts: Location Privacy in Wireless Networks*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2011.

[68] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30, 1997.

[69] R. Gennaro. Private Communication. 2010.

[70] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT*, pages 123–139, 1999.

[71] R. Gennaro, C. Hazay, and J. Sorensen. Text Search Protocols with Simulation Based Security. In *PKC*, pages 332–350, 2010.

[72] R. Gennaro, H. Krawczyk, and T. Rabin. Okamoto-Tanaka revisited: Fully Authenticated Diffie-Hellman with Minimal Overhead. In *ACNS*, pages 309–328, 2010.

[73] C. Gentry and Z. Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In *ICALP*, pages 803–815, 2005.

[74] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *STOC*, pages 151–160, 1998.

[75] I. Goldberg. Privacy-enhancing technologies for the Internet, II: Five years later. In *PET*, pages 1–12, 2002.

[76] I. Goldberg, D. Wagner, and E. Brewer. Privacy-enhancing technologies for the Internet. In *COMPCON*, pages 103–109, 1997.

[77] O. Goldreich. *Foundations of Cryptography*. Cambridge Univ. Press, 2004.

[78] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[79] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, pages 218–229, 1987.

[80] O. Goldreich and V. Rosen. On the security of modular exponentiation with application to the construction of pseudorandom generators. *Journal of Cryptology*, 16(2):71–93, 2003.

[81] D. Gritzalis. *Secure electronic voting*. Springer, 2003.

[82] S. Gürses, C. Troncoso, and C. Diaz. Engineering privacy by design. In *CPDP*, 2011.

[83] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, pages 216–227, 2002.

[84] C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *CCS*, pages 491–500, 2008.

[85] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[86] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer, 2010.

[87] C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC*, pages 312–331, 2010.

[88] C. Hazay and T. Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, pages 195–212, 2010.

[89] T. Herlea, J. Claessens, B. Preneel, G. Neven, F. Piessens, and B. De Decker. On securely scheduling a meeting. In *SEC*, pages 183–198, 2001.

[90] J. Holt and K. Seamons. Reconciling CA-oblivious encryption, hidden credentials, OSBE, and secret handshakes. Internet Security Research Lab Technical Report, Brigham Young University, 2006.

[91] J. Hong, E. Suh, and S. Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4), 2009.

[92] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB*, pages 720–731, 2004.

[93] B. Hore, J. Wickramasuriya, S. Mehrotra, N. Venkatasubramanian, and D. Massaguer. Privacy-preserving event detection in pervasive spaces. In *PERCOM*, pages 1–10, 2009.

[94] B. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce*, pages 78–86, 1999.

[95] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.

[96] S. Jarecki, J. Kim, and G. Tsudik. Beyond Secret Handshakes: Affiliation-Hiding Authenticated Key Exchange. In *CT-RSA*, pages 352–369, 2008.

[97] S. Jarecki and X. Liu. Affiliation-Hiding Envelope and Authentication Schemes with Efficient Support for Multiple Credentials. In *ICALP*, pages 715–726, 2008.

[98] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, pages 577–594, 2009.

[99] S. Jarecki and X. Liu. Private mutual authentication and conditional oblivious transfer. *CRYPTO*, pages 90–107, 2009.

[100] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, pages 418–435, 2010.

[101] M. Kantarcioĝlu and C. Clifton. Assuring privacy when big brother is watching. In *DMKD*, pages 88–93, 2003.

[102] A. Karlson, S. Iqbal, B. Meyers, G. Ramos, K. Lee, and J. Tang. Mobile taskflow in context: a screenshot study of smartphone usage. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2009–2018, 2010.

[103] A. Karlson, B. Meyers, A. Jacobs, P. Johns, and S. Kane. Working Overtime: Patterns of Smartphone and PC Usage in the Day of an Information Worker. In *PERVASIVE*, pages 398–405, 2009.

[104] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman & Hall/CRC, 2008.

[105] J. Katz and L. Malka. Secure Text Processing with Applications to Private DNA Matching. In *CCS*, pages 485–492, 2010.

[106] B. Kellermann and R. Bohme. Privacy-enhanced event scheduling. In *CSE*, pages 52–59, 2009.

[107] J. Kilian. Founding Cryptography on Oblivious Transfer. In *STOC*, pages 20–31, 1988.

[108] L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[109] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[110] T. Léauté and B. Faltings. Privacy-preserving multi-agent constraint satisfaction. In *CSE*, pages 17–25, 2009.

[111] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *PODC*, pages 182–189, 2003.

[112] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. In *CRYPTO*, pages 36–54, 2000.

[113] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[114] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *ISC*, pages 314–328, 2005.

[115] H. Lipmaa. Oblivious Transfer or Private Information Retrieval Links. `http://research.cyber.ee/~lipmaa/crypto/link/protocols/oblivious.php`, 2009.

[116] M. Manulis, B. Poettering, and G. Tsudik. Affiliation-Hiding Key Exchange with Untrusted Group Authorities. In *ACNS*, pages 402–419, 2010.

[117] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia. Privacy-aware proximity based services. In *MDM*, pages 31–40, 2009.

[118] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC, 1997.

[119] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[120] S. Micali, M. O. Rabin, and J. Kilian. Zero-Knowledge Sets. In *FOCS*, pages 80–91, 2003.

[121] S. Micali and P. Rogaway. Secure computation. In *CRYPTO*, pages 392–404, 1991.

[122] M. Murugesan, W. Jiang, C. Clifton, L. Si, and J. Vaidya. Efficient privacy-preserving similar document detection. *VLDB Journal*, 2010.

[123] M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. In *STOC*, pages 245–254, 1999.

[124] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[125] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, 1–35(5):1254, 2006.

[126] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location Privacy via Private Proximity Testing. In *NDSS*, 2011.

[127] S. Nasserian and G. Tsudik. Revisiting oblivious signature-based envelopes. In *FC*, pages 221–235, 2006.

[128] Nokia. Ovi. `http://ovi.com/`, 2010.

[129] Nokia Qt Development Frameworks. Qt – Cross Platform application and UI framework. `http://qt.nokia.com/`, 2011.

[130] Nokia Research Center. Nokia Instant Community. `http://tinyurl.com/nokiainstcomm`, 2010.

[131] F. Olumofin and I. Goldberg. Privacy-preserving Queries over Relational Databases. In *PETS*, pages 75–92, 2010.

[132] R. Ostrovsky and W. Skeith. A survey of single-database private information retrieval: Techniques and applications. In *PKC*, pages 393–411, 2007.

[133] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[134] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, pages 387–398, 1996.

[135] K. Puttaswamy and B. Zhao. Preserving privacy in location-based mobile social applications. In *HotMobile*, pages 1–6, 2010.

[136] M. Rabin. How to exchange secrets by oblivious transfer. TR-81, Harvard Aiken Computation Lab, 1981.

[137] M. Raykova, B. Vo, S. Bellovin, and T. Malkin. Secure anonymous database search. In *CCSW*, pages 115–126, 2009.

[138] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[139] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[140] A. Shamir. How to Share a Secret. *Communications of ACM*, 22(11):612–613, 1979.

[141] A. Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, 1983.

[142] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *CRYPTO*, volume 196 of *LNCS*, pages 47–53, 1984.

[143] Sherri Davidoff. What Does DHS Know About You? `http://philosecurity.org/2009/09/07/what-does-dhs-know-about-you`.

[144] J. Shi, R. Zhang, Y. Liu, and Y. Zhang. PriSense: Privacy-Preserving Data Aggregation in People-Centric Urban Sensing Systems. In *INFOCOM*, 2010.

[145] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *S&P*, pages 44–55, 2000.

[146] G. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Transactions of the American Institute of Electrical Engineers*, 45, 1926.

[147] S. Xu and M. Yung. k-Anonymous Secret Handshakes with Reusable Credentials. In *CCS*, pages 158–167, 2004.

[148] A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[149] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *Principles and Practice of Constraint Programming*, 2006.

[150] G. Zhong. Distributed approaches for Location Privacy. Masters Thesis, University of Waterloo, 2008.

[151] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three protocols for Location Privacy. In *PET*, pages 62–76, 2007.

[152] A. Zunino and M. Campo. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 2009.

Appendix A

Performance Evaluation of Private Set Intersection Protocols

This appendix presents the experimental analysis of state-of-the-art Private Set Intersection protocols. We consider several variants, reviewed below:

- **Private Set Intersection (PSI)** involves a server and a client, on input $\mathcal{S} = \{s_1, \ldots, s_w\}$ and $\mathcal{C} = \{c_1, \ldots, c_v\}$, respectively. It results in the client outputting $\mathcal{S} \cap \mathcal{C}$.
- **Authorized Private Set Intersection (APSI)** involves a server and a client, on input $\mathcal{S} = \{s_1, \ldots, s_w\}$ and $\mathcal{C} = \{(c_1, \sigma_1), \ldots, (c_v, \sigma_v)\}$, respectively. It results in the client outputting $\{s_j \in \mathcal{S} \mid \exists\,(c_i, \sigma_i) \in \mathcal{C}$ s.t. $c_i = s_j \wedge \mathsf{Vrfy}_{\mathsf{pk}}(\sigma_i, c_i) = 1\}$, where pk is the public key of a trusted (offline) authorization authority (denoted as CA), given a digital signature scheme $\mathsf{DSIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$.
- **PSI with Data Transfer (PSI-DT)** involves a server, on input a set of items, each with associated data, $\mathcal{S} = \{(s_1, data_1), \cdots, (s_w, data_w)\}$, and a client, on input $\mathcal{C} = \{c_1, \cdots, c_v\}$. It results in the client outputting $\{(s_j, data_j) \in \mathcal{S} \mid \exists c_i \in \mathcal{C}\ s.t.\ c_i = s_j\}$.
- **Authorized PSI-DT (APSI-DT)** involves a server, on input $\mathcal{S} = \{(s_1, data_1), \cdots, (s_w, data_w)\}$, and a client, on input of a set of items with associated authorizations, $\mathcal{C} = \{(c_1, \sigma_i) \cdots, (c_v, \sigma_v)\}$. It results in client outputting $\{(s_j, data_j) \in \mathcal{S} \mid \exists (c_i, \sigma_i) \in \mathcal{C}\ s.t.\ c_i = s_j \wedge \mathsf{Vrfy}_{\mathsf{pk}}(\sigma_i, c_i) = 1\}$, where pk is the public key of a trusted (offline) authorization authority (denoted as CA), given a digital signature scheme $\mathsf{DSIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Recall from Section 8.2.3 that PSI techniques can also be distinguished based on whether or not they support pre-distribution of server inputs. Specifically, we denote as *(A)PSI-DT with pre-distribution* those protocol constructions where the server can “pre-process” its input set, independently from client input to the protocol. This way, the server can pre-distribute its (processed) set items before protocol execution. Since both pre-processing and pre-distribution can be done offline, once for all possible clients, server’s online complexity does not depend on server input size.

Implemented Protocols

We implement state-of-the-art PSI protocols listed in Table A.1. We distinguish between PSI-DT and APSI-DT variant, as well as between constructions with or without pre-distribution. We choose to implement protocols with the data transfer functionality, since they are more appealing for realistic application scenarios.

	w/o Pre-Distribution	**w/ Pre-Distribution**
PSI-DT	FNP04: [66], DT10-1: Figure 5.3	JL09: [98], JL10: [100], DT10-2: Figure 5.4
APSI-DT	DT10-APSI: Figure 5.2	-

Table A.1: Implemented PSI-DT and APSI-DT protocols.

Implementation Criteria

We develop our testing software in C++ using OpenSSL (ver. 1.0), GMP (ver. 5.01) and PBC (ver. 0.57) libraries. All measurements are performed on a Ubuntu 9.10 desktop platform with Intel Xeon E5420 CPU (2.5GHz and 6MB cache) and 8GB RAM.

In protocols supporting data transfer, the data associated with each server item can be arbitrarily long. The performance of some protocols is dominated by the size of this data, rather than sets size (e.g., in FNP04). In order to obtain a fair comparison, however, it is crucial to capture the “intrinsic” cost of each protocol, stemming from the underlying cryptographic tools. To this end, we employ the following strategy: we encrypt data associated to each set item with a distinct random symmetric key and consider these keys as the new associated data. Assuming that a different key is selected at each interaction, this technique does not violate server unlinkability. This way, the computation cost of each protocol is measured based on the same fixed-length key, regardless of data size. In our experiments, we set symmetric key size to 128 bits.

In all experiments, we use 1024-bit RSA moduli and 1024-bit cyclic-group moduli with a 160-bit subgroup order. Our goal is to compare performance of different PSI protocols, thus, we do not vary keys/moduli size as protocols exhibit the same trend. All test results are averaged over 100 independent runs. All protocols are instantiated under the assumption of *semi-honest* adversaries and in the *Random Oracle Model* (ROM).

Measurements

As discussed above, each protocol execution involves additional overhead of symmetric en-/de-cryption of records. Figure A.1 compares the resulting overhead (for variable data sizes), using either RC4 or AES-CBC (with 128-bit keys).

We assume that the client does not perform any pre-computation, while the server performs as much pre-computation on its input as possible. This reflects the reality where client input is (usually) determined in real time, while server input is pre-determined. Figure A.2 shows the pre-computation overhead for each protocol.

Next, we evaluate online computation overhead. Figures A.3 and A.4 show client online computation overhead with respect to client and server input sizes, respectively. Whereas, figures A.5 and A.6 show server online computation overhead with respect to client and server input size.

Then, Figures A.7 and A.8 evaluate protocol bandwidth complexity with respect to client and server input sizes. For protocols with pre-distribution, bandwidth consumption (since the transfer of database encryption is performed offline) does not include pre-distribution overhead. In these figures, we sometimes use the same marker for different protocols to indicate that these protocols share the same value. Client input size v (resp., server input size w) is fixed at 5,000 in figures where x-axis refers to the server (resp., the client) input size.

Performance Comparison

PSI-DT without pre-distribution. We now compare FNP04 and DT10-1 protocols. From Figures A.3-A.8, we conclude that that FNP04 is much more expensive than DT10-1 in terms of client and server online computation as well as bandwidth consumption. For each client set size, DT10-1 client overhead ranges from 460ms to 4,400ms, while FNP04 server overhead – between 1,300ms and 15,000ms. For each chosen server set size, server overhead in DT10-1 is under 1,300ms, while in FNP04 it exceeds 15,000ms.

PSI-DT with pre-distribution. Next, we compare JL09, JL10, and DT10-2. Recall that all protocols, for the sake of our experiments, are instantiated in semi-honest model, thus, ZKPK-s are not included for JL09 and JL10. Figures A.3-A.8 demonstrate that DT10-2 incurs client overhead almost two orders of magnitude lower than JL09 and JL10. In fact, DT10-2 involves two client multiplications for each item, while JL09 performs two heavy homomorphic operations and JL10 – two exponentiations. In JL10, the server online computation overhead results from v 160-bit exponentiations, whereas, in DT10-2, it results from v RSA exponentiations. Since these exponentiations can be speeded up using the Chinese Remainder Theorem, the gap (for server computation overhead) between JL10 and DT10-2 is only double. Summing up server and client computation overhead, DT10-2 results to be the most efficient. In terms of bandwidth consumption, DT10-2 and JL10 are almost the same, while JL09 is slightly more expensive.

APSI-DT without pre-distribution. The only protocol available in this context is DT10-APSI. Figure A.3-A.6 illustrates that client overhead is determined only by client set size, whereas, server overhead is determined by both client and server set sizes. Measurements obtained for APSI-DT naturally mirror those of DT10-1, as the former simply adds authorization of client inputs (by merging signatures into the protocol).

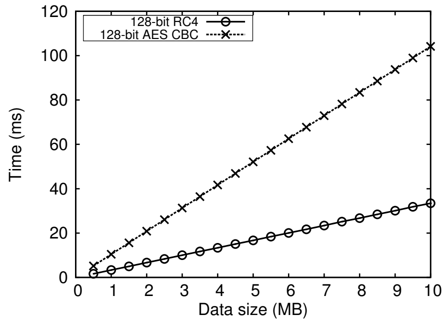


Figure A.1: Symmetric key en-/de-cryption performance.

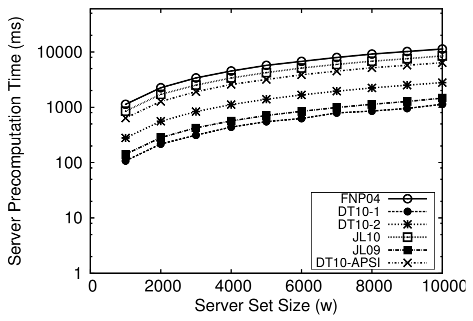


Figure A.2: Server pre-computation overhead.

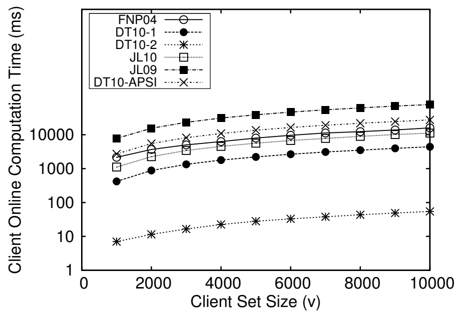


Figure A.3: Client online computation w.r.t. client set size.

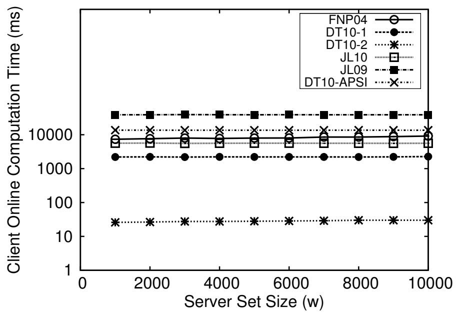


Figure A.4: Client online computation w.r.t. server set size.

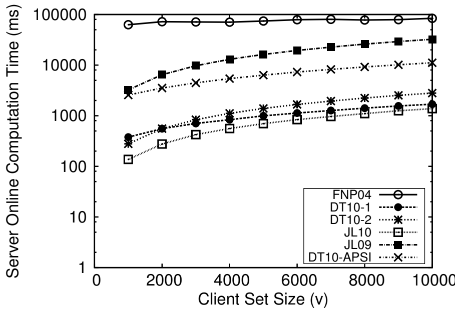


Figure A.5: Server online computation w.r.t. client set size.

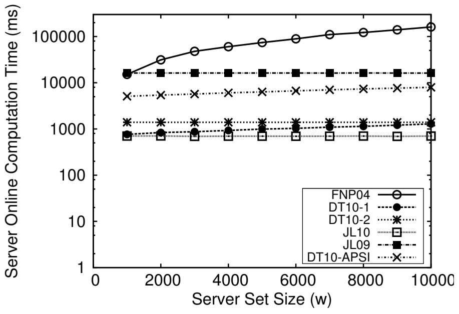


Figure A.6: Server online computation w.r.t. server set size.

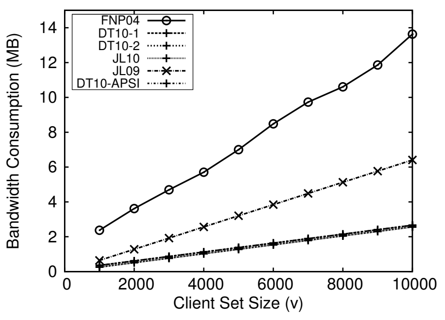


Figure A.7: Bandwidth consumption w.r.t. client set size.

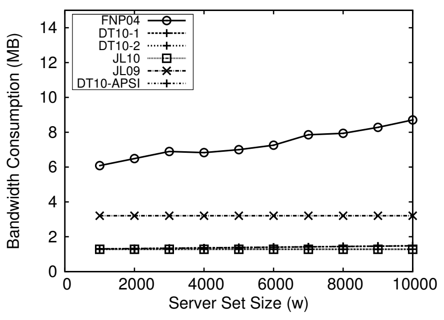


Figure A.8: Bandwidth consumption w.r.t. server set size.